Package ‘flowWorkspace’

April 19, 2019

Type Package

Title Infrastructure for representing and interacting with gated and ungated cytometry data sets.

Version 3.31.16

Date 2011-06-10

Author Greg Finak, Mike Jiang

Maintainer Greg Finak <gfinak@fhcrc.org>, Mike Jiang <wjiang2@fhcrc.org>

Description This package is designed to facilitate comparison of automated gating methods against manual gating done in flowJo. This package allows you to import basic flowJo workspaces into BioConductor and replicate the gating from flowJo using the flowCore functionality. Gating hierarchies, groups of samples, compensation, and transformation are performed so that the output matches the flowJo analysis.

License Artistic-2.0

LazyLoad yes

Imports Biobase, BiocGenerics, graph, graphics, gridDevices, lattice, methods, stats, stats4, utils, RBGL, tools, gridExtra, Rgraphviz, data.table, dplyr, latticeExtra, Rcpp, RColorBrewer, stringr, scales, flowViz, matrixStats, digest, RcppParallel


Depends R (>= 2.16.0), flowCore(>= 1.49.10), ncdfFlow(>= 2.25.4)

biocViews ImmunoOncology, FlowCytometry, DataImport, Preprocessing, DataRepresentation

Suggests testthat, flowWorkspaceData, knitr, ggcyto, parallel, CytoML

LinkingTo Rcpp, BH(>= 1.62.0-1), RProtoBufLib(>= 1.3.7), cytolib(>= 1.3.3), RcppParallel

VignetteBuilder knitr

SystemRequirements GNU make, C++11

Encoding UTF-8
RoxygenNote 6.1.1

git_url https://git.bioconductor.org/packages/flowWorkspace

git_branch master

git_last_commit 8074f4c

git_last_commit_date 2019-04-11

Date/Publication 2019-04-18

R topics documented:

<table>
<thead>
<tr>
<th>Package/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowWorkspace-package</td>
<td>4</td>
</tr>
<tr>
<td>add,GatingSet.list-method</td>
<td>4</td>
</tr>
<tr>
<td>asinhGml2_trans</td>
<td>8</td>
</tr>
<tr>
<td>asinh_Gml2</td>
<td>8</td>
</tr>
<tr>
<td>booleanFilter-class</td>
<td>9</td>
</tr>
<tr>
<td>checkRedundantNodes</td>
<td>10</td>
</tr>
<tr>
<td>clone</td>
<td>11</td>
</tr>
<tr>
<td>compensate,GatingSet,ANY-method</td>
<td>11</td>
</tr>
<tr>
<td>compute_timestep</td>
<td>12</td>
</tr>
<tr>
<td>copyNode</td>
<td>13</td>
</tr>
<tr>
<td>dropRedundantChannels</td>
<td>13</td>
</tr>
<tr>
<td>dropRedundantNodes</td>
<td>14</td>
</tr>
<tr>
<td>estimateLogicle,GatingHierarchy</td>
<td>14</td>
</tr>
<tr>
<td>extract_cluster_pop_name_from_node</td>
<td>15</td>
</tr>
<tr>
<td>filterObject,rectangleGate-method</td>
<td>16</td>
</tr>
<tr>
<td>fix_channel_slash</td>
<td>16</td>
</tr>
<tr>
<td>flowData,GatingSet-method</td>
<td>17</td>
</tr>
<tr>
<td>flowJo.fasinh</td>
<td>18</td>
</tr>
<tr>
<td>flowJo.flog</td>
<td>18</td>
</tr>
<tr>
<td>flowJoTrans</td>
<td>19</td>
</tr>
<tr>
<td>flowJo_biexp_trans</td>
<td>20</td>
</tr>
<tr>
<td>flowJo_fasinh_trans</td>
<td>21</td>
</tr>
<tr>
<td>flowWorkspace.par.init</td>
<td>21</td>
</tr>
<tr>
<td>flowWorkspace.par.set</td>
<td>22</td>
</tr>
<tr>
<td>flow_breaks</td>
<td>22</td>
</tr>
<tr>
<td>flow_trans</td>
<td>23</td>
</tr>
<tr>
<td>GatingHierarchy-class</td>
<td>24</td>
</tr>
<tr>
<td>GatingSet,flowSet,ANY-method</td>
<td>24</td>
</tr>
<tr>
<td>GatingSet-class</td>
<td>25</td>
</tr>
<tr>
<td>GatingSetList-class</td>
<td>27</td>
</tr>
<tr>
<td>getCompensationMatrices</td>
<td>29</td>
</tr>
<tr>
<td>getCompensationObj</td>
<td>30</td>
</tr>
<tr>
<td>getData,GatingHierarchy,missing-method</td>
<td>30</td>
</tr>
<tr>
<td>getDescendants</td>
<td>31</td>
</tr>
<tr>
<td>getFullPath</td>
<td>32</td>
</tr>
<tr>
<td>getGate,GatingHierarchy,character-method</td>
<td>32</td>
</tr>
<tr>
<td>getIndiceMat</td>
<td>33</td>
</tr>
<tr>
<td>getIndices,GatingHierarchy,character-method</td>
<td>34</td>
</tr>
<tr>
<td>getIndices,GatingSet,name-method</td>
<td>35</td>
</tr>
<tr>
<td>getLoglevel</td>
<td>35</td>
</tr>
<tr>
<td>getMergedStats</td>
<td>36</td>
</tr>
<tr>
<td>R topics documented:</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>getNodes,GatingSet-method</td>
<td>36</td>
</tr>
<tr>
<td>getParent,GatingSet.character-method</td>
<td>37</td>
</tr>
<tr>
<td>getProp,GatingHierarchy.character-method</td>
<td>38</td>
</tr>
<tr>
<td>getSingleCellExpression,GatingSetList.character-method</td>
<td>40</td>
</tr>
<tr>
<td>getStats</td>
<td>41</td>
</tr>
<tr>
<td>getTransformations</td>
<td>43</td>
</tr>
<tr>
<td>get_leaf_nodes</td>
<td>44</td>
</tr>
<tr>
<td>gh_check_cluster_node</td>
<td>44</td>
</tr>
<tr>
<td>gh_get_cluster_labels</td>
<td>45</td>
</tr>
<tr>
<td>groupByChannels</td>
<td>45</td>
</tr>
<tr>
<td>groupByTree</td>
<td>46</td>
</tr>
<tr>
<td>insertGate</td>
<td>46</td>
</tr>
<tr>
<td>isGated</td>
<td>47</td>
</tr>
<tr>
<td>isNcdf</td>
<td>48</td>
</tr>
<tr>
<td>keyword,GatingHierarchy.character-method</td>
<td>48</td>
</tr>
<tr>
<td>lapply,GatingSet-method</td>
<td>49</td>
</tr>
<tr>
<td>length,GatingSet-method</td>
<td>50</td>
</tr>
<tr>
<td>logicleGml2_trans</td>
<td>50</td>
</tr>
<tr>
<td>logicle_trans</td>
<td>51</td>
</tr>
<tr>
<td>logtGml2_trans</td>
<td>52</td>
</tr>
<tr>
<td>markernames,GatingHierarchy-method</td>
<td>52</td>
</tr>
<tr>
<td>mkformula</td>
<td>54</td>
</tr>
<tr>
<td>moveNode</td>
<td>54</td>
</tr>
<tr>
<td>ncFlowSet</td>
<td>55</td>
</tr>
<tr>
<td>openWorkspace</td>
<td>55</td>
</tr>
<tr>
<td>pData,GatingHierarchy-method</td>
<td>56</td>
</tr>
<tr>
<td>plot,GatingSet,missing-method</td>
<td>56</td>
</tr>
<tr>
<td>plotGate</td>
<td>57</td>
</tr>
<tr>
<td>plotPopCV,GatingHierarchy-method</td>
<td>59</td>
</tr>
<tr>
<td>plot_diff_tree</td>
<td>60</td>
</tr>
<tr>
<td>pop.MFI</td>
<td>61</td>
</tr>
<tr>
<td>prettyAxis</td>
<td>61</td>
</tr>
<tr>
<td>recompute,GatingSet-method</td>
<td>62</td>
</tr>
<tr>
<td>rotate_gate</td>
<td>63</td>
</tr>
<tr>
<td>sampleNames,GatingHierarchy-method</td>
<td>64</td>
</tr>
<tr>
<td>save_gs</td>
<td>65</td>
</tr>
<tr>
<td>scale_gate</td>
<td>66</td>
</tr>
<tr>
<td>set.count.xml</td>
<td>67</td>
</tr>
<tr>
<td>setGate,GatingHierarchy.character.filter-method</td>
<td>67</td>
</tr>
<tr>
<td>setNode,GatingHierarchy.character.character-method</td>
<td>68</td>
</tr>
<tr>
<td>shift_gate</td>
<td>69</td>
</tr>
<tr>
<td>standardize,GatingSet</td>
<td>71</td>
</tr>
<tr>
<td>subset.GatingSet</td>
<td>71</td>
</tr>
<tr>
<td>swap_data_cols</td>
<td>72</td>
</tr>
<tr>
<td>transform,GatingSet-method</td>
<td>73</td>
</tr>
<tr>
<td>transformerList</td>
<td>74</td>
</tr>
<tr>
<td>transform_gate</td>
<td>74</td>
</tr>
<tr>
<td>updateChannels</td>
<td>76</td>
</tr>
<tr>
<td>updateIndices,GatingHierarchy.character.logical-method</td>
<td>77</td>
</tr>
</tbody>
</table>

Index 79
flowWorkspace-package

Import and replicate flowJo workspaces and gating schemes using flowCore.

Description

Import flowJo workspaces into R. Generate the flowJo gating hierarchy and gates using flowCore functionality. Transform and compensate data in accordance with flowJo settings. Plot gates, gating hierarchies, population statistics, and compare flowJo vs flowCore population summaries.

Details

<table>
<thead>
<tr>
<th>Package:</th>
<th>flowWorkspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>0.5.40</td>
</tr>
<tr>
<td>Date:</td>
<td>2011-03-04</td>
</tr>
<tr>
<td>License:</td>
<td>Artistic 2.0</td>
</tr>
<tr>
<td>LazyLoad:</td>
<td>yes</td>
</tr>
<tr>
<td>Depends:</td>
<td>R (&gt;= 2.16.0), Rcpp (&gt;= 0.9.9)</td>
</tr>
</tbody>
</table>

Author(s)

Greg Finak, Mike Jiang

References

http://www.rglab.org/

add,GatingSet,list-method

Create a GatingSet and add/remove the flowCore gate(or population) to/from a GatingHierarchy/GatingSet.

Description

GatingSet method creates a gatingset from a flowSet with the ungated data as the root node. add method add the flowCore gate to a GatingHierarchy/GatingSet. setGate method update the gate of one population node in GatingHierarchy/GatingSet. Rm method Remove the population node from a GatingHierarchy/GatingSet. They are equivalent to the workFlow.add and Rm methods in flowCore package. recompute method does the actual gating after the gate is added, i.e. calculating the event indices according to the gate definition.
### Usage

```r
## S4 method for signature 'GatingSet,list'
add(wf, action, ...)

## S4 method for signature 'GatingSetList,list'
add(wf, action, ...)

## S4 method for signature 'GatingSet,filtersList'
add(wf, action, ...)

## S4 method for signature 'GatingSet,filterList'
add(wf, action, validityCheck = TRUE, ...)

## S4 method for signature 'GatingSetList,filterList'
add(wf, action, ...)

## S4 method for signature 'GatingSetList,filtersList'
add(wf, action, ...)

## S4 method for signature 'GatingSet,filter'
add(wf, action, ...)

## S4 method for signature 'GatingSet,filters'
add(wf, action, ...)

## S4 method for signature 'GatingSetList,filter'
add(wf, action, ...)

## S4 method for signature 'GatingSetList,filters'
add(wf, action, ...)

## S4 method for signature 'GatingHierarchy,filter'
add(wf, action, ...)

## S4 method for signature 'GatingHierarchy,filters'
add(wf, action, names = NULL, ...)

## S4 method for signature 'GatingHierarchy,quadGate'
add(wf, action, names = NULL, ...)

## S4 method for signature 'GatingHierarchy,logical'
add(wf, action, parent, name,
    recompute, cluster_method_name = NULL, ...)

## S4 method for signature 'GatingHierarchy,Factor'
add(wf, action, name = NULL, ...)

## S4 method for signature 'GatingHierarchy,logicalFilterResult'
add(wf, action, ...)

## S4 method for signature 'GatingHierarchy,multipleFilterResult'
add(wf, action, ...)
```
add(wf, action,  
    name = NULL, ...)  

## S4 method for signature 'character,GatingSet,character'  
Rm(symbol, envir, subSymbol, ...)  

## S4 method for signature 'character,GatingSetList,character'  
Rm(symbol, envir, subSymbol, ...)  

## S4 method for signature 'character,GatingHierarchy,character'  
Rm(symbol, envir,  
    subSymbol, ...)  

Arguments  

wf A GatingHierrarchy or GatingSet  
action A filter or a list of filters to be added to the GatingHierarchy or GatingSet.  
... some other arguments to specify how the gates are added to the gating tree.  
  • negated: a logical scalar to specify whether the gate is negated, which  
    means the the population outside of the gate will be kept as the result pop- 
    ulation. It is FALSE by default.  
validityCheck logical whether to check the consistency of tree structure across samples. de- 
  fault is TRUE. Can be turned off when speed is prefered to the robustness.  
names a character vector of length four, which specifies the population names resulted 
  by adding a quadGate.  
parent a character scalar to specify the parent node name where the new gate to be 
  added to, by default it is NULL, which indicates the root node  
name a character scalar to specify the node name of population that is generated by 
  the gate to be added.  
recompute a logical flag The order of the names is clock-wise starting from the top left 
  quadrant population.  
cluster_method_name when adding the logical vectors as the gates, the name of the cluster method can 
  be used to tag the populations as the extra meta information associated with the 
  gates.  
symbol A character identifies the population node in a GatingHierrarchy or GatingSet 
  to remove  
envir A GatingHierrarchy or GatingSet  
subSymbol Not used.  

Value  

GatingSet method returns a GatingSet object with just root node. add method returns a popu- 
lation node ID (or four population node IDs when adding a quadGate) that uniquely identify the 
population node within a GatingHierarchy.  

See Also  

GatingSet-class
Examples

```r
## Not run:
data(GvHD)
#select raw flow data
fs<-GvHD[1:3]

#transform the raw data
tf <- transformList(colnames(fs[[1]])[3:6], asinh, transformationId="asinh")
fs_trans<-transform(fs,tf)

#add transformed data to a gatingset
gs <- GatingSet(fs_trans)
gs
getNodes(gs[[1]]) #only contains root node

#add one gate
rg <- rectangleGate("FSC-H"=c(200,400), "SSC-H"=c(250, 400),
                    filterId="rectangle")
nodeID<-add(gs, rg)#it is added to root node by default if parent is not specified
nodeID
getNodes(gs[[1]]) #the second population is named after filterId of the gate

#add a quadGate
qg <- quadGate("FL1-H"=2, "FL2-H"=4)
nodeIDs<-add(gs,qg,parent="rectangle")
nodeIDs #quadGate produces four population nodes
getNodes(gs[[1]]) #population names are named after dimensions of gate if not specified

#add a boolean Gate
bg<-booleanFilter("CD15 FITC-CD45 PE+","CD15 FITC+CD45 PE-")
bg
nodeID2<-add(gs,bg,parent="rectangle")
nodeID2
getNodes(gs[[1]])

#do the actual gating
recompute(gs)

#plot one gate for one sample
plotGate(gs[[1]],"rectangle")
plotGate(gs[[1]],nodeIDs) #may be smoothed automatically if there are not enough events after gating

#plot gates across samples using lattice plot
plotGate(gs,nodeID)

#plot all gates for one sample
plotGate(gs[[1]])#boolean gate is skipped by default
plotGate(gs[[1]],bool=TRUE)

#plot the gating hierarchy
require(Rgraphviz)
plot(gs[[1]])

#remove one node causing the removal of all the descendants
Rm('rectangle', gs)
getNodes(gs[[1]])

## End(Not run)
```
asinhtGml2_trans  
*Inverse hyperbolic sine transformation.*

**Description**

Used to construct inverse hyperbolic sine transform object.

**Usage**

asinhtGml2_trans(..., n = 6, equal.space = FALSE)

**Arguments**

- ...  
  parameters passed to asinh_Gml2
- n  
  desired number of breaks (the actual number will be different depending on the data range)
- equal.space  
  whether breaks at equal-spaced intervals

**Value**

asinhtGml2 transformation object

**Examples**

```r
trans.obj <- asinhtGml2_trans(equal.space = TRUE)
data <- 1:1e3
brks.func <- trans.obj["breaks"]
brks <- brks.func(data)
brks # fasinh space displayed at raw data scale

#transform it to verify it is equal-spaced at transformed scale
trans.func <- trans.obj["transform"]
brks.trans <- trans.func(brks)
brks.trans
```

---

asinh_Gml2  
*inverse hyperbolic sine transform function generator (GatingML 2.0 version)*

**Description**

hyperbolic sine/inverse hyperbolic sine transform function constructor. It is simply a special form of flowJo.fasinh with length set to 1 and different default values for parameters `t, m, a`.

**Usage**

asinh_Gml2(T = 262144, M = 4.5, A = 0, inverse = FALSE)
Arguments

- **T**: numeric the maximum value of input data
- **M**: numeric the full width of the transformed display in asymptotic decades
- **A**: numeric Additional negative range to be included in the display in asymptotic decades
- **inverse**: whether to return the inverse function

Value

fasinh/fsinh transform function

Examples

```r
trans <- asinh_Gml2()
data.raw <- c(1,1e2,1e3)data.trans <- trans(data.raw)data.trans
inverse.trans <- asinh_Gml2(inverse = TRUE)inverse.trans(data.trans)
```

booleanFilter-class

A class describing logical operation (& or |) of the reference populations

Description

booleanFilter class inherits class expressionFilter and exists for the purpose of methods dispatching.

Usage

```r
booleanFilter(expr, ..., filterId = "defaultBooleanFilter")
char2booleanFilter(expr, ..., filterId = "defaultBooleanFilter")

## S4 method for signature 'booleanFilter'
show(object)
```

Arguments

- **expr**: expression
- **...**: further arguments to the expression
- **filterId**: character identifier
- **object**: booleanFilter

See Also

addGatingHierarchy
checkRedundantNodes

Examples

# "4+/TNFa+" and "4+/IL2+" are two existing gates
# note: no spaces between node names and & , ! operators
booleanFilter("4+/TNFa+&!4+/IL2+")

# programmatically
n1 <- "4+/TNFa+
 n2 <- "4+/IL2+
exprs <- paste0(n1, "&", n2)
call <- substitute(booleanFilter(v), list(v = as.symbol(exprs)))
eval(call)

checkRedundantNodes try to determine the redundant terminal(or leaf) nodes that can be removed

Description

These leaf nodes make the gating trees to be different from one another and can be removed by the subsequent convenient call dropRedundantNodes.

Usage

checkRedundantNodes(x, path = "auto", ...)

Arguments

x GatingSet or list of groups(each group is a list of 'GatingSet'). When it is a list, it is usually the outcome from groupByTree.
path argumented passed to getNodes. The default value is "auto".
... other arguments passed to getNodes.

Value

a list of the character vectors indicating the nodes that are considered to be redundant for each group of GatingSets.

Examples

## Not run:
gslist <- list(gs1, gs2, gs3, gs4, gs5)
gs_groups <- groupByTree(gslist)
toRm <- checkRedundantNodes(gs_groups)

## End(Not run)
clone

**Description**

Clone a GatingSet

**Usage**

`clone(x, ...)`

**Arguments**

- `x`: A GatingSet
- `...`: `ncdfFile = NULL`: see `clone.ncdfFlowSet`

**Details**

Note that the regular R assignment operation on a `GatingSet` object does not return the copy as one would normally expect because the `GatingSet` contains environment slots (and external pointer for `GatingSet`), which require deep-copying. So make sure to use this clone method in order to make a copy of existing object.

**Value**

A copy of a given `GatingSet`.

**Examples**

```r
## Not run:
#G is a GatingSet
G1<-clone(G)
## End(Not run)
```

compensate,GatingSet,ANY-method

**Description**

Compensate the flow data associated with the GatingSet

**Usage**

- `## S4 method for signature 'GatingSet,ANY'
  compensate(x, spillover)`
- `## S4 method for signature 'GatingSetList,ANY'
  compensate(x, spillover)`

**Description**

The compensation is saved in the GatingSet and can be retrieved by `getCompensationMatrices`.
Arguments

x   GatingSet or GatingSetList
spillover compensation object or a list of compensation objects

Value

a GatingSet or GatingSetList object with the underlying flow data compensated.

Examples

## Not run:
cfile <- system.file("extdata","compdata","compmatrix", package="flowCore")
comp.mat <- read.table(cfile, header=TRUE, skip=2, check.names = FALSE)
## create a compensation object
comp <- compensation(comp.mat, compensationId="comp1")
# add it to GatingSet
gs <- compensate(gs, comp)
## End(Not run)

compute_timestep

compute time step from fcs keyword

Description

compute time step from fcs keyword

Usage

compute_timestep(kw, unit.range, timestep.source = c("TIMESTEP", "BTIM"))

Arguments

kw      list of keywords
unit.range the actual measured time unit range
timestep.source either "TIMESTEP" or "BTIM". prefer to STIMESTEP keyword when it is non NULL
copyNode

Copy a node along with all of its descendant nodes to the given ancestor

Description

Copy a node along with all of its descendant nodes to the given ancestor

Usage

copyNode(gh, node, to)

Arguments

gh GatingHierarchy
node the node to be copied

Examples

library(flowWorkspace)
dataDir <- system.file("extdata", package="flowWorkspaceData")
suppressMessages(gs <- load_gs(list.files(dataDir, pattern = "gs_manual", full = TRUE)))
gh <- gs[[1]]
old.parent <- getParent(gh, "CD4")
new.parent <- "singlets"
copyNode(gh, "CD4", new.parent)

dropRedundantChannels

Remove the channels from flow data that are not used by gates

Description

Removing these redundant channels can help standardize the channels across different GatingSet objects and make them mergable.

Usage

dropRedundantChannels(gs, ...)

Arguments

gs a GatingSet

Value

a new GatingSet object that has redundant channels removed. Please note that this new object shares the same reference (or external pointers) with the original GatingSets.
Examples

```r
## Not run:
gs_new <- dropRedundantChannels(gs)
## End(Not run)
```

dropRedundantNodes

_remove the terminal leaf nodes that make the gating trees to be different from one another.

Description

It is usually called after `groupByTree` and `checkRedundantNodes`. The operation is done in place through external pointers which means all the original GatingSets are modified.

Usage

dropRedundantNodes(x, toRemove)

Arguments

- `x` GatingSet or list of groups (each group is a list of `GatingSet`). When it is a list, it is usually the outcome from `groupByTree`.
- `toRemove` list of the node sets to be removed. Its length must equals to the length of `x`. When `x` is a list, `toRemove` is usually the outcome from `checkRedundantNodes`.

Examples

```r
## Not run:
gslist <- list(gs1, gs2, gs3, gs4, gs5)
gs_groups <- groupByTree(gslist)
toRm <- checkRedundantNodes(gs_groups)
dropRedundantNodes(gs_groups, toRm)
# Now they can be merged into a single GatingSetList.
# Note that the original gs objects are all modified in place.
GatingSetList(gslist)
## End(Not run)
```

estimateLogicle.GatingHierarchy

_Compute logicle transformation from the flowData associated with a GatingHierarchy_

Description

See details in `?flowCore::estimateLogicle`
## S3 method for class 'GatingHierarchy'

estimateLogicle(x, channels, ...)

### Arguments

- **x**: a GatingHierarchy
- **channels**: channels or markers for which the logicle transformation is to be estimated.
- **...**: other arguments

### Value

transformerList object

### Examples

```r
## Not run:
# gs is a GatingSet
trans.list <- estimateLogicle(gs[[1]], c("CD3", "CD4", "CD8"))
# trans.list is a transformerList that can be directly applied to GatinigSet
gs <- transform(gs, trans.list)
## End(Not run)
```

### Description

Extract the population name from the node path. It strips the parent path and cluster method name.

### Usage

extract_cluster_pop_name_from_node(node, cluster_method_name)

### Arguments

- **node**: population node path
- **cluster_method_name**: the name of the clustering method

### Examples

```r
extract_cluster_pop_name_from_node("cd3/flowClust_pop1", "flowClust")
# returns "pop1"
```
filterObject, rectangleGate-method

convert flowCore filter to a list It convert the flowCore gate to a list whose structure can be understood by underlying c++ data structure.

Description

convert flowCore filter to a list
It convert the flowCore gate to a list whose structure can be understood by underlying c++ data structure.

Usage

## S4 method for signature 'rectangleGate'
filterObject(x)

## S4 method for signature 'polygonGate'
filterObject(x)

## S4 method for signature 'booleanFilter'
filterObject(x)

## S4 method for signature 'ellipsoidGate'
filterObject(x)

## S4 method for signature 'logical'
filterObject(x)

Arguments

x filter a flowCore gate. Currently supported gates are: "rectangleGate", "polygonGate","ellipsoidGate" and "booleanFilter"

Value

a list

fix_channel_slash toggle the channel names between '/' and '_' character

Description

FlowJoX tends to replace '/' in the original channel names with '_' in gates and transformations. We need to do the same to the flow data but also need to change it back during the process since the channel names of the flowSet can’t be modified until the data is fully compensated.

Usage

fix_channel_slash(chnls, slash_loc = NULL)
flowData,GatingSet-method

Arguments

chnls  the channel names
slash_loc  a list that records the locations of the original slash character within each channel name so that when restoring slash it won’t tamper the the original ‘_’ character.

Value

the toggled channel names

---

flowData,GatingSet-method

*Fetch or replace the flowData object associated with a GatingSet.*

Description

Accessor method that gets or replaces the flowset/ncdfFlowSet object in a GatingSet or GatingHierarchy

Usage

```r
## S4 method for signature 'GatingSet'
flowData(x)
```

```r
## S4 replacement method for signature 'GatingSet'
flowData(x) <- value
```

Arguments

x  A GatingSet
value  The replacement flowSet or ncdfFlowSet object

Details

Accessor method that sets or replaces the ncdfFlowSet object in the GatingSet or GatingHierarchy.

Value

the object with the new flowSet in place.
flowJo.fasinh

inverse hyperbolic sine transform function

Description

hyperbolic sine/inverse hyperbolic sine (flowJo-version) transform function constructor

Usage

flowJo.fasinh(m = 4, t = 12000, a = 0.7, length = 256)
flowJo.fsinh(m = 4, t = 12000, a = 0.7, length = 256)

Arguments

m numeric the full width of the transformed display in asymptotic decades
t numeric the maximum value of input data
a numeric Additional negative range to be included in the display in asymptotic decades
length numeric the maximum value of transformed data

Value

fasinh/fsinh transform function

Examples

trans <- flowJo.fasinh()
data.raw <- c(1, 1e2, 1e3)
data.trans <- trans(data.raw)data.trans
inverse.trans <- flowJo.fsinh()
inverse.trans(data.trans)

flowJo.flog

flog transform function

Description

flog transform function constructor. It is different from flowCore version of logtGml2 in the way that it reset negative input so that no NAN will be returned.

Usage

flowJo.flog(decade = 4.5, offset = 1, max_val = 262144, min_val = 0, scale = 1, inverse = FALSE)
flowJoTrans

Arguments

- **decade**: number of decades
- **offset**: offset to the original input
- **max_val**: top of scale value
- **min_val**: lower bound of scaled value (where negative raw value gets truncated at)
- **scale**: the linear scale factor
- **inverse**: whether return the inverse function

Value

flog(or its inverse) transform function

Examples

```r
trans <- flowJo.flog()
data.raw <- c(1,1e2,1e3)
data.trans <- trans(data.raw)
data.trans

inverse.trans <- flowJo.flog(inverse = TRUE)
inverse.trans(data.trans)

# negative input
data.raw <- c(-10,1e2,1e3)
data.trans <- trans(data.raw)
data.trans
inverse.trans(data.trans)# we lose the original value at lower end since flog can't restore negative value

# different
trans <- flowJo.flog(decade = 3, max_val = 1e3)
data.trans <- trans(data.raw)
data.trans
inverse.trans <- flowJo.flog(decade = 3, max_val = 1e3, inverse = TRUE)
inverse.trans(data.trans)
```

flowJoTrans  
*construct the flowJo-type biexponential transformation function*

Description

Normally it was parsed from flowJo xml workspace. This function provides the alternate way to construct the flowJo version of logicle transformation function within R.

Usage

```r
flowJoTrans(channelRange = 4096, maxValue = 262144, pos = 4.5, neg = 0, widthBasis = -10, inverse = FALSE)
```
Arguments

- **channelRange**: numeric, the maximum value of transformed data
- **maxValue**: numeric, the maximum value of input data
- **pos**: numeric, the full width of the transformed display in asymptotic decades
- **neg**: numeric, additional negative range to be included in the display in asymptotic decades
- **widthBasis**: numeric, unknown.
- **inverse**: logical, whether to return the inverse transformation function.

Examples

```r
trans <- flowJoTrans()
data.raw <- c(-1, 1e3, 1e5)
data.trans <- trans(data.raw)
round(data.trans)
inv <- flowJoTrans(inverse = TRUE)
round(inv(data.trans))
```

---

**flowJo_biexp_trans**

*flowJo biexponential transformation.*

Description

Used for constructing biexponential transformation object.

Usage

```r
flowJo_biexp_trans(..., n = 6, equal.space = FALSE)
```

Arguments

- **...**: parameters passed to `flowJoTrans`
- **n**: desired number of breaks (the actual number will be different depending on the data range)
- **equal.space**: whether breaks at equal-spaced intervals

Value

biexponential transformation object

Examples

```r
data(GvHD)
fr <- GvHD[[1]]
data.raw <- exprs(fr)[, "FL1-H"]
trans.obj <- flowJo_biexp_trans(equal.space = TRUE)
brks.func <- trans.obj[["breaks"]]
brks <- brks.func(data.raw)
brks # biexp space displayed at raw data scale
```
flowJo_fasinh_trans

#transform it to verify it is equal-spaced at transformed scale
trans.func <- trans.obj[["transform"]]
print(trans.func(brks))

flowJo_fasinh_trans  
flowJo inverse hyperbolic sine transformation.

Description
Used to construct the inverse hyperbolic sine transform object.

Usage
flowJo_fasinh_trans(..., n = 6, equal.space = FALSE)

Arguments

... parameters passed to flowJo.fasinh
n desired number of breaks (the actual number will be different depending on the data range)
equal.space whether breaks at equal-spaced intervals

Value
fasinh transformation object

Examples
trans.obj <- flowJo_fasinh_trans(equal.space = TRUE)
data <- 1:1e3
brks.func <- trans.obj[["breaks"]]
brks <- brks.func(data)
brks # fasinh space displayed at raw data scale

#transform it to verify it is equal-spaced at transformed scale
trans.func <- trans.obj[["transform"]]
round(trans.func(brks))

flowWorkspace.par.init

workspace version is parsed from xml node '/Workspace/version' in flowJo workspace and matched with this list to dispatch to the one of the three workspace parsers

Description
workspace version is parsed from xml node '/Workspace/version' in flowJo workspace and matched with this list to dispatch to the one of the three workspace parsers

Usage
flowWorkspace.par.init()
flowWorkspace.par.set

flowWorkspace.par.set sets a set of parameters in the flowWorkspace package namespace.

Description

flowWorkspace.par.get gets a set of parameters in the flowWorkspace package namespace.

Usage

flowWorkspace.par.set(name, value)
flowWorkspace.par.get(name = NULL)

Arguments

name The name of a parameter category to get or set.
value A named list of values to set for category name or a list of such lists if name is missing.

Details

It is currently used to add/remove the support for a specific flowJo versions (parsed from xml node '/Workspace/version' in flowJo workspace)

Examples

# get the flowJo versions currently supported
old <- flowWorkspace.par.get("FlowJo_versions")

# add the new version
old["win"] <- c(old["win"], "1.7")
flowWorkspace.par.set("FlowJo_versions", old)

flowWorkspace.par.get("FlowJo_versions")

flow_breaks

Generate the breaks that makes sense for flow data visualization

Description

It is mainly used as helper function to construct breaks function used by 'trans_new'.

Usage

flow_breaks(x, n = 6, equal.space = FALSE, trans.fun, inverse.fun)
flow_trans

Arguments

- **x**  
  the raw data values
- **n**  
  desired number of breaks (the actual number will be different depending on the data range)
- **equal.space**  
  whether breaks at equal-spaced intervals
- **trans.fun**  
  the transform function (only needed when equal.space is TRUE)
- **inverse.fun**  
  the inverse function (only needed when equal.space is TRUE)

Value

either 10^n intervals or equal-spaced(after transformed) intervals in raw scale.

Examples

data(GvHD)
fr <- GvHD[[1]]
data.raw <- exprs(fr)[, "FL1-H"]
flow_breaks(data.raw)

trans <- logicleTransform()
inv <- inverseLogicleTransform(trans = trans)
myBrks <- flow_breaks(data.raw, equal.space = TRUE, trans = trans, inv = inv)
round(myBrks)
# to verify it is equally spaced at transformed scale
print(trans(myBrks))

flow_trans

helper function to generate a trans objects Used by other specific trans constructor

Description

helper function to generate a trans objects Used by other specific trans constructor

Usage

flow_trans(name, trans.fun, inverse.fun, equal.space = FALSE, n = 6)

Arguments

- **name**  
  transformation name
- **trans.fun**  
  the transform function (only needed when equal.space is TRUE)
- **inverse.fun**  
  the inverse function (only needed when equal.space is TRUE)
- **equal.space**  
  whether breaks at equal-spaced intervals
- **n**  
  desired number of breaks (the actual number will be different depending on the data range)
GatingHierarchy-class  

Class GatingHierarchy

Description

GatingHierarchy is a class for representing the gating hierarchy, which can be either imported from a flowJo workspace or constructed in R.

Details

There is a one-to-one correspondence between GatingHierarchy objects and FCS files in the flowJo workspace. Each sample (FCS file) is associated with its own GatingHierarchy. It is also more space efficient by storing gating results as logical/bit vector instead of copying the raw data.

Given a GatingHierarchy, one can extract the data associated with any subpopulation, extract gates, plot gates, and extract population proportions. This facilitates the comparison of manual gating methods with automated gating algorithms.

See Also

GatingSet

Examples

## Not run:
require(flowWorkspaceData)
d<-system.file("extdata",package="flowWorkspaceData")
wsfile<-list.files(d,pattern="A2004Analysis.xml",full=TRUE)
library(CytoML)
ws <- openWorkspace(wsfile);
G<-try(parseWorkspace(ws,path=d,name=1));
gh <- G[[1]]
getPopStats(gh);
plotPopCV(gh)

nodes <- getNodes(gh)

thisNode <- nodes[4]
plotGate(gh,thisNode);
getGate(gh,thisNode);
getData(gh,thisNode)

## End(Not run)

GatingSet,flowSet,ANY-method  

constructors for GatingSet

Description

construct a gatingset with empty trees (just root node)

construct object from existing gating hierarchy (gating template) and flow data
Usage

## S4 method for signature 'flowSet,ANY'
GatingSet(x)

## S4 method for signature 'GatingHierarchy,character'
GatingSet(x, y, path = ".", ...)

## S4 method for signature 'GatingSet'
identifier(object)

## S4 method for signature 'GatingSetList'
identifier(object)

## S4 replacement method for signature 'GatingSet,character'
identifier(object) <- value

## S4 replacement method for signature 'GatingSetList,character'
identifier(object) <- value

Arguments

x  GatingSet
y  GatingHierarchy
path character specifies the path to the flow data (FCS files)
... other arguments.
object GatingSet
value string

Examples

## Not run:
#fdata could be a flowSet or ncdfFlowSet
gs <- GatingSet(fdata)

## End(Not run)

GatingSet-class

Class "GatingSet"

Description

GatingSet holds a set of GatingHierarchy objects, representing a set of samples and the gating scheme associated with each.

[ subsets a GatingSet or GatingSetList using the familiar bracket notation

[[ extract a GatingHierarchy object from a GatingSet or GatingSetList


Usage

## S4 method for signature 'GatingSet,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'GatingSet,numeric'
x[[i, j, ...]]

## S4 method for signature 'GatingSetList,ANY'
x[i, j, ..., drop = TRUE]

Arguments

x GatingSet or GatingSetList
i numeric or logical or character used as sample index
j not used
... not used
drop not used

Details

Objects stores a collection of GatingHierarchies and represent a group in a flowJo workspace. A GatingSet can have two “states”. After a call to parseWorkspace(....,execute=FALSE), the workspace is imported but the data is not. Setting execute to TRUE is needed in order to load, transform, compensate, and gate the associated data. Whether or not a GatingHierarchy has been applied to data is encoded in the flag slot. Some methods will warn the user, or may not function correctly if the GatingHierarchy has not been executed. This mechanism is in place, largely for the purpose of speed when working with larger workspaces. It allows the use to load a workspace and subset desired samples before proceeding to load the data.

Slots

FCSPath: deprecated
data: Object of class "FlowSet". flow data associated with this GatingSet
flag: Object of class "logical". A flag indicating whether the gates, transformations, and compensation matrices have been applied to data, or simply imported.
axis: Object of class "list". stores the axis information used for plotGate.
pointer: Object of class "externalptr". points to the gating hierarchy stored in C data structure.
guid: Object of class "character". the unique identifier for GatingSet object.
transformation: Object of class "list". a list of transformation objects used by GatingSet.
compensation: Object of class "ANY". compensation objects.

See Also

GatingHierarchy
Examples

```r
## Not run:
require(flowWorkspaceData)
d<-system.file("extdata",package="flowWorkspaceData")
wsfile<-list.files(d,pattern="A2004Analysis.xml",full=TRUE)
library(CytoML)
ws <- openWorkspace(wsfile);
G<-try(parseWorkspace(ws,execute=TRUE,path=d,name=1));
plotPopCV(G);
## End(Not run)
```

### GatingSetList-class

Class "GatingSetList"

Description

A list of GatingSet objects. This class exists for method dispatching.

use GatingSetList constructor to create a GatingSetList from a list of GatingSet

Usage

```r
GatingSetList(x, samples = NULL)
```

## S4 method for signature 'GatingSetList,missing'
```r
rbind2(x, y = "missing", ...)
```

Arguments

- **x**: a list of GatingSet
- **samples**: character vector specifying the order of samples. if not specified, the samples are ordered as the underlying stored order.
- **y**: missing not used.
- **...**: other arguments passed to `rbind2` method for ncdfflowList

Details

Objects store a collection of GatingSets, which usually has the same gating trees and markers. Most GatingSets methods can be applied to GatingSetList.

See Also

- GatingSet
- GatingHierarchy
## Not run:

```r
#load several GatingSets from disk
gs_list <- lapply(list.files("../gs_toMerge", full=TRUE), function(this_folder){
  load.gs(this_folder)
})

#gs_list is a list
gs_groups <- merge(gs_list)
#returns a list of GatingSetList objects
gslist2 <- gs_groups[[2]]
#gslist2 is a GatingSetList that contains multiple GatingSets and they share the same gating and data structure

class(gslist2)
sampleNames(gslist2)

#reference a GatingSet by numeric index
gslist2[[1]]
#reference a GatingSet by character index
gslist2["30104.fcs"]

#loop through all GatingSets within GatingSetList
lapply(gslist2, sampleNames)

#subset a GatingSetList by [
sampleNames(gslist2[c(4,1)])
sampleNames(gslist2[c(1,4)])
gslist2[c("30104.fcs")]

#get flow data from it
getData(gslist2)
#get gated flow data from a particular population
getData(gslist2, "3+")

#extract the gates associated with one population
getGate(gslist2, "3+")
getGate(gslist2, 5)

#extract the pheno data
pData(gslist2[3:1])
#modify the pheno data
pd <- pData(gslist2)
pd$id <- 1:nrow(pd)
pData(gslist2) <- pd
pData(gslist2[3:2])

#plot the gate
plotGate(gslist2[1:2], 5, smooth=T)
plotGate_label(gslist2[3:4], 4, x="<APC Cy7-A>", y="<PE Tx RD-A>", smooth=T)

#remove certain gates by loop through GatingSets
getNodes(gslist2[[1]])
lapply(gslist2, function(gs) Rm("Excl", gs))

#extract the stats
getPopStats(gslist2)
```
getCompensationMatrices

Retrieve the compensation matrices from a GatingHierarchy.

Description

Retrieve the compensation matrices from a GatingHierarchy.

Usage

getCompensationMatrices(x)

Arguments

x A GatingHierarchy object.

Details

Return all the compensation matrices in a GatingHierarchy.

Value

A list of matrix representing the spillover matrix in GatingHierarchy.
getData,GatingHierarchy,missing-method

**Examples**

```r
## Not run:
#Assume gh is a GatingHierarchy
getCompensationMatrices(gh);

## End(Not run)
```

**getDescriptionObj**

`getDescriptionObj` **extract compensation object from GatingSet**

**Description**

extract compensation object from GatingSet

**Usage**

`getDescriptionObj(gs, sampleName)`

**Arguments**

- `gs` GatingSet
- `sampleName` sample name

**getData,GatingHierarchy,missing-method**

`getData` **get gated flow data from a GatingHierarchy/GatingSet/GatingSetList**

**Description**

get gated flow data from a GatingHierarchy/GatingSet/GatingSetList

**Usage**

```r
## S4 method for signature 'GatingHierarchy,missing'
getData(obj, y, ...)

## S4 method for signature 'GatingHierarchy,character'
getData(obj, y, ...)

## S4 method for signature 'GatingSet,missing'
getData(obj, y, ...)

## S4 method for signature 'GatingSet,character'
getData(obj, y, ...)

## S4 method for signature 'GatingSetList,ANY'
getData(obj, y, ...)
```
getDescendants

get all the descendant nodes for the given ancestor

Arguments

obj A GatingHierarchy, GatingSet or GatingSetList object.
y character the node name or full/partial) gating path. If not specified, will return the complete flowFrame/flowSet at the root node.
... arguments passed to ncdfFlow::[]

Details

Returns a flowFrame/flowSet containing the events in the gate defined at node y. Subset membership can be obtained using getIndices. Population statistics can be obtained using getPop and getPopStats. When calling getData on a GatingSet, the trees representing the GatingHierarchy for each sample in the GaingSet are presumed to have the same structure. To update the data, use flowData method.

Value

A flowFrame object if obj is a GatingHierarchy. A flowSet or ncdfFlowSet if a GatingSet. A ncdfFlowList if a GatingSetList.

See Also

flowData getIndices getPopStats

Examples

## Not run:
#G is a GatingSet
gedata(G,3) #get a flowSet constructed from the third node / population in the tree.
gedata(G,"cd4")

#gh is a GatingHierarchy
gedata(gh)

## End(Not run)
Examples

```r
library(flowWorkspace)
dataDir <- system.file("extdata", package="flowWorkspaceData")
suppressMessages(gs <- load_gs(list.files(dataDir, pattern = "gs_manual", full = TRUE)))
getDescendants(gs[[1]], "CD4")
getDescendants(gs[[1]], "CD8", path = "auto")
```

Description

convert the partial gating path to the full path

Usage

```r
getFullNodePath(gh, path)
```

Arguments

- `gh`: GatingHierarchy object
- `path`: the partial gating path

Value

the full gating path

getGate,GatingHierarchy,character-method

Return the flowCore gate definition associated with a node in a GatingHierarchy/GatingSet.

Description

Return the flowCore gate definition object associated with a node in a GatingHierarchy or GatingSet object.

Usage

```r
## S4 method for signature 'GatingHierarchy,character'
getGate(obj, y)

## S4 method for signature 'GatingSet,character'
getGate(obj, y)

## S4 method for signature 'GatingSetList,character'
getGate(obj, y)
```
getIndiceMat

Arguments

- **obj**: A GatingHierarchy or GatingSet
- **y**: A character the name or full/(partial) gating path of the node of interest.

Value

A gate object from flowCore. Usually a polygonGate, but may be a rectangleGate. Boolean gates are represented by a "BooleanGate" S3 class. This is a list boolean gate definition that references populations in the GatingHierarchy and how they are to be combined logically. If obj is a GatingSet, assuming the trees associated with each GatingHierarchy are identical, then this method will return a list of gates, one for each sample in the GatingSet corresponding to the same population indexed by y.

See Also

- `getData`
- `getNodes`

Examples

```r
## Not run: #gh is a GatingHierarchy
getGate(gh, "CD3") #return the gate for the fifth node in the tree, but fetch it by name.
#G is a GatingSet
getGate(G, "CD3") #return a list of gates for the fifth node in each tree
## End(Not run)
```

---

**getIndiceMat**  
Return the single-cell matrix of 1/0 dichotomized expression

**Description**

Return the single-cell matrix of 1/0 dichotomized expression

**Usage**

```r
getIndiceMat(gh, y)
```

**Arguments**

- **gh**: GatingHierarchy object
- **y**: character node name
getIndices, GatingHierarchy, character-method

Get the membership indices for each event with respect to a particular gate in a GatingHierarchy

Description

Returns a logical vector that describes whether each event in a sample is included or excluded by this gate.

Usage

```r
## S4 method for signature 'GatingHierarchy,character'
getIndices(obj, y)
```

Arguments

- `obj`: A `GatingHierarchy` representing a sample.
- `y`: A character giving the name or full/partial gating path of the population / node of interest.

Details

Returns a logical vector that describes whether each event in the data file is included in the given gate of this `GatingHierarchy`. The indices are for all events in the file, and do not reflect the population counts relative to the parent but relative to the root. To get population frequencies relative to the parent one cross-tabulate the indices of `y` with the indices of its parent.

Value

A logical vector of length equal to the number of events in the FCS file that determines whether each event is or is not included in the current gate.

Note

Generally you should not need to use `getIndices` but the more convenient methods `getProp` and `getPopStats` which return population frequencies relative to the parent node. The indices returned reference all events in the file and are not directly suitable for computing population statistics, unless subsets are taken with respect to the parent populations.

See Also

- `getPopStats`

Examples

```r
## Not run:
G is a gating hierarchy
#Return the indices for population 5 (topological sort)
getIndices(G, getNode(G, tsort=TRUE)[5]);

## End(Not run)
```
**getIndices, GatingSet, name-method**

routine to return the indices by specify boolean combination of reference nodes:

### Description

It adds the boolean gates and does the gating on the fly, and return the indices associated with that bool gate, and remove the bool gate the typical use case would be extracting any-cytokine-expressed cells

### Usage

```r
## S4 method for signature 'GatingSet, name'
getIndices(obj, y)
```

### Arguments

- `obj` : GatingSet
- `y` : a quoted expression.

### Examples

```r
## Not run:
getIndices(gs, quote(`4+/TNFa+|4+/IL2+`))
```

### getLoglevel

get/set the log level

### Description

It is helpful sometime to get more detailed print out for the purpose of trouble shooting

### Usage

```r
getLoglevel()
```

```r
setLoglevel(level = "none")
```

### Arguments

- `level` : a character that represents the log level, can be value of c("none", "GatingSet", "GatingHierarchy", "Population", "gate") default is "none", which does not print any information from C parser.
Value

a character that represents the internal log level

Examples

getLoglevel()
setLoglevel("Population")
getLoglevel()

getMergedStats Get Cell Population Statistics and Sample Metadata

Description

Get Cell Population Statistics and Sample Metadata

Usage

getMergedStats(object, ...)

Arguments

object

a GatingSet or GatingSetList

... additional arguments passed to getPopStats

Value

a data.table of merged population statistics with sample metadata.

Examples

## Not run:
#G is a GatingSetList
stats = getMergedStats(G)

## End(Not run)

getNodes,GatingSet-method

Get the names of all nodes from a gating hierarchy.

Description

getNodes returns a character vector of names of the nodes (populations) in the GatingSet.

Usage

## S4 method for signature 'GatingSet'
getNodes(x, y = NULL, order = "regular",
     path = "full", showHidden = FALSE, ...)


Arguments

x A GatingSet Assuming the gating hierarchy are identical within the GatingSet, the Gating tree of the first sample is used to query the node information.
y A character not used.
order order=c("regular","tsort","bfs") returns the nodes in regular, topological or breadth-first sort order. "regular" is default.
path A character or numeric scalar. when numeric, it specifies the fixed length of gating path (length 1 displays terminal name). When character, it can be either 'full' (full path, which is default) or 'auto' (display the shortest unique gating path from the bottom of gating tree).
showHidden logical whether to include the hidden nodes
... Additional arguments.

Details

integer indices of nodes are based on regular order, so whenever need to map from character node name to integer node ID, make sure to use default order which is regular.

Value

getNodes returns a character vector of node/population names, ordered appropriately.

Examples

## Not run:
#G is a gating hierarchy
getNodes(G, path = 1)#return node names (without prefix)
getNodes(G, path = "full")#return the full path
getNodes(G, path = 2)#return the path as length of two
getNodes(G, path = "auto")#automatically determine the length of path
setNode(G,"L","lymph")

## End(Not run)

getParent,GatingSet,character-method

Return the name of the parent population or a list of child populations of the current population in the GatingHierarchy

Description

Returns the name of the parent population or a character/numeric vector of all the children of the current population in the given GatingHierarchy

Usage

## S4 method for signature 'GatingSet,character'
getParent(obj, y, ...)  

## S4 method for signature 'GatingSet,character'
getChildren(obj, y, showHidden = TRUE, ...)


getProp, GatingHierarchy, character-method

Arguments

- **obj** A GatingHierarchy
- **y** a character/numeric the name or full/(partial) gating path or node indices of the node / population.
- **showHidden** logical whether to include the hidden children nodes.

... other arguments passed to getNodes methods

Value

getParent returns a character vector, the name of the parent population. getChildren returns a character or numeric vector of the node names or node indices of the child nodes of the current node. An empty vector if the node has no children.

See Also

- getNodes

Examples

```r
# Not run:
# G is a gatinghierarchy
#return the name of the parent of the fifth node in the hierarchy.
getParent(G,getNodes(G[1][5])

n<-getNodes(G,tsort=T)[4];
getChildren(G,n);#Get the names of the child nodes of the 4th node in this gating hierarchy.
getChildren(G,4);#Get the ids of the child nodes

# End(Not run)
```

getProp, GatingHierarchy, character-method

Return a table of population statistics for all populations in a GatingHierarchy/GatingSet or the population proportions or the total number of events of a node (population) in a GatingHierarchy

Description

getProp calculates the population proportion (events in the gate / events in the parent population) associated with a node in the GatingHierarchy. getPopStats is more useful than getPop. Returns a table of population statistics for all populations in a GatingHierarchy/GatingSet. Includes the xml counts, openCyto counts and frequencies. getTotal returns the total number of events in the gate defined in the GatingHierarchy object

Usage

```r
## S4 method for signature 'GatingHierarchy,character'
getProp(x, y, xml = FALSE)

## S4 method for signature 'GatingHierarchy,character'
getTotal(x, y, xml = FALSE)
```
getProp.GatingHierarchy.character-method

## S4 method for signature 'GatingHierarchy'
getPopStats(x, path = "auto", ...)

## S4 method for signature 'GatingSet'
getPopStats(x, statistic = c("freq", "count"),
            xml = FALSE, subpopulations = NULL, format = c("long", "wide"),
            path = "full", ...)

## S4 method for signature 'GatingSetList'
getPopStats(x, format = c("long", "wide"), ...)

Arguments

- **x**: A GatingHierarchy or GatingSet.
- **y**: character node name or path.
- **xml**: logical indicating whether the statistics come from xml (if parsed from xml workspace) or from openCyto.
- **path**: character see getNodes.
- **...**: Additional arguments passed to getNodes.
- **statistic**: character specifies the type of population statistics to extract. (only valid when format is "wide"). Either "freq" or "count" is currently supported.
- **subpopulations**: character vector to specify a subset of populations to return. (only valid when format is "long").
- **format**: character value of c("wide", "long") specifying whether to origanize the output in long or wide format.

Details

getPopStats returns a table population statistics for all populations in the gating hierarchy. The output is useful for verifying that the import was successful, if the xml and openCyto derived counts don’t differ much (i.e. if they have a small coefficient of variation.) for a GatingSet, returns a matrix of proportions for all populations and all samples getProp returns the proportion of cells in the gate, relative to its parent. getTotal returns the total number of events included in this gate. The contents of "thisTot" variable in the "metadata" environment of the nodeData element associated with the gating tree and gate / population.

Value

getPopStats returns a data.frame with columns for the population name, xml derived counts, openCyto derived counts, and the population proportions (relative to their parent pouplation). getProp returns a population frequency numeric. getTotal returns a numeric value of the total number of elements in the population.

See Also

- getNodes

Examples

## Not run:

#gh is a GatingHierarchy
getPopStats(gh);
#proportion for the fifth population
getProp(gh,getNodes(gh)[5])
getTotal(gh,getNodes(gh,tsort=T)[5])

#gs is a GatingSet
getPopStats(gs)
#optionally output in long format as a data.table
getPopStats(gs, format = "long", path = "auto")
#only get stats for a subset of populations
getPopStats(gs, format = "long", subpopulations = getNodes(gs)[4:6])

## End(Not run)

getSingleCellExpression,GatingSetList,character-method

Return the cell events data that express in any of the single populations defined in y

Description

Returns a list of matrix containing the events that expressed in any one of the populations defined in y

Usage

## S4 method for signature 'GatingSetList,character'
getSingleCellExpression(x, nodes, ...)

## S4 method for signature 'GatingSet,character'
getSingleCellExpression(x, nodes, ...)

getSingleCellExpressionByGate(...)

Arguments

x A GatingSet or GatingSetList object.
nodes character vector specifying different cell populations
... other arguments
other.markers character vector specifying the extra markers/channels to be returned besides the ones derived from "nodes" and "map" argument. It is only valid when threshold is set to FALSE.
swap logical indicates whether channels and markers of flow data are swapped.
threshold logical indicates whether to threshold the flow data by setting intensity value to zero when it is below the gate threshold.
marginal logical indicates whether to the gate is treaded as 1d marginal gate. Default is TRUE, which means markers are determined either by node name or by "map" argument explained below. When FALSE, the markers are determined by the gate dimensions. and node name and "map" argument are ignored.
map a named list providing the mapping between node names (as specified in the gating hierarchy of the gating set) and channel names (as specified in either
getStats

Exact MFI from populations(or nodes) for all the markers

Description

It calculates the MFI for each marker.
getStats

Usage

getStats(x, ...)

## S3 method for class 'GatingSetList'
getStats(x, ...)

## S3 method for class 'GatingSet'
getStats(x, ...)

## S3 method for class 'GatingHierarchy'
getStats(x, nodes = NULL, type = "count",
    inverse.transform = FALSE, stats.fun.arg = list(), ...)

Arguments

x          a GatingSet or GatingHierarchy
...
   arguments passed to getNodes method.
nodes       the character vector specifies the populations of interest. default is all available
            nodes
type       the character vector specifies the type of pop stats or a function used to compute
           population stats. when character, it is expected to be either "count" or "percent".
           Default is "count" (total number of events in the populations). when a function,
           it takes a flowFrame object through 'fr' argument and return the stats as a named
           vector.
inverse.transform
            logical flag . Whether inverse transform the data before computing the stats.
stats.fun.arg       a list of arguments passed to 'type' when 'type' is a function.

Value

da data.table that contains MFI values for each marker per column along with 'pop' column and
'sample' column (when used on a 'GatingSet')

Examples

## Not run:
dataDir <- system.file("extdata",package="flowWorkspaceData")
suppressMessages(gs <- load_gs(list.files(dataDir, pattern = "gs_manual",full = TRUE)))

# get stats all nodes
dt <- getStats(gs) #default is "count"

nodes <- c("CD4", "CD8")
getStats(gs, nodes, "percent")

# pass a build-in function
getStats(gs, nodes, type = pop.MFI)

# compute the stats based on the raw data scale
getStats(gs, nodes, type = pop.MFI, inverse.transform = TRUE)

# supply user-defined stats fun
pop.quantiles <- function(fr){
getTransformations <- function(x, channel = NULL, inverse = FALSE, only.function = TRUE, ...) {
  if (is.null(channel)) {
    # Return a list of all the transformations or a transformation in the flowJo workspace.
    # The list is of length L, where L is the number of distinct transformations applied to samples in the flowJo Workspace.
    # Each element of L is itself a list of length M, where M is the number of parameters that were transformed for a sample or group of samples in a flowJoWorkspace. For example, if a sample has 10 parameters, and 5 are transformed during analysis, using two different sets of transformations, then L will be of length 2, and each element of L will be of length 5. The elements of L represent channel- or parameter-specific transformation functions that map from raw intensity values to channel-space used by flowJo.
    getTransformations(x, channel = NULL, inverse = FALSE, only.function = TRUE, ...) 
  }

  names(res) <- chnls
  res
}

getStats(gs, nodes, type = pop.quantiles)
## End(Not run)
Examples

```r
## Not run:
# Assume gh is a GatingHierarchy
getTransformations(gh); # return a list transformation functions
getTransformations(gh, inverse = TRUE); # return a list inverse transformation functions
getTransformations(gh, channel = "FL1-H"); # only return the transformation associated with given channel
getTransformations(gh, channel = "FL1-H", only.function = FALSE) # return the entire transform object

## End(Not run)
```

get_leaf_nodes

get all the leaf nodes

**Description**

get all the leaf nodes

**Usage**

```r
get_leaf_nodes(x, ...)
```

**Arguments**

- `x` GatingHierarchy/GatingSet object
- `...` arguments passed to `getNodes` method

**Value**

the leaf nodes

---

**gh_check_cluster_node**

check if a node is clustering node

**Description**

check if a node is clustering node

**Usage**

```r
gh_check_cluster_node(gh, node)
```

**Arguments**

- `gh` GatingHierarchy
- `node` the population/node name or path

**Value**

the name of the clustering method. If it is not cluster node, returns NULL
gh_get_cluster_labels  Retrieve the cluster labels from the cluster nodes

Description

Clustering results are stored as individual gated nodes. This helper function collect all the gating indices from the same clustering run (identified by ’parent’ node and ’cluster_method_name” and merge them as a single factor.

Usage

gh_get_cluster_labels(gh, parent, cluster_method_name)

Arguments

- gh: GatingHierarchy
- parent: the parent population/node name or path
- cluster_method_name: the name of the clustering method

groupByChannels  split GatingSets into groups based on their flow channels

Description

Sometime it is gates are defined on the different dimensions across different GatingSets, (e.g. ‘FSC-W’ or ‘SSC-H’ may be used for Y axis for cytokines) These difference in dimensions may not be critical since they are usually just used for visualization(instead of thresholding events) But this prevents the gs from merging because they may not be collected across batches Thus we have to separate them if we want to visualize the gates.

Usage

groupByChannels(x)

Arguments

- x: a list of GatingSets

Examples

## Not run:
gslist <- list(gs1, gs2, gs3, gs4, gs5)
gs_groups <- groupByChannels(gslist)

## End(Not run)
**groupByTree**

*split GatingSets into groups based on their gating schemes Be careful that the splitted results still points to the original data set!!*

**Description**

It allows isomorphism in Gating tree and ignore difference in hidden nodes i.e. tree is considered to be the same as long as `getNodes(gh, path = "auto", showHidden = F)` returns the same set.

**Usage**

```r
groupByTree(x)
```

**Arguments**

- `x` a list of GatingSets or one GatingSet

**Value**

when `x` is a GatingSet, this function returns a list of sub-GatingSets When `x` is a list of GatingSets, it returns a list of list, each list itself is a list of GatingSets, which share the same gating tree.

**Examples**

```r
## Not run:
gslist <- list(gs1, gs2, gs3, gs4, gs5)
gs_groups <- groupByTree(gslist)
## End(Not run)
```

---

**insertGate**

*insert a dummy gate to the GatingSet*

**Description**

Is is useful trick to make the tree structure of GatingSet same with other so that they can be combined into a `GatingSetList` object. (deprecated by `moveNode`)

**Usage**

```r
insertGate(gs, gate, parent, children)
```

**Arguments**

- `gs` GatingSet to work with
- `gate` filter a dummy gate to be inserted, its `filterId` will be used as the population name
- `parent` character full path of parent node where the new dummy gate to be added to
- `children` character full path of children nodes that the new dummy gate to be parent of
isGated

Value

A new GatingSet object with the new gate added but share the same flow data with the input 'GatingSet'

Examples

```r
## Not run:
# construct a dummy singlet gate
dummyGate <- rectangleGate("FSC-A" = c(-Inf, Inf), "FSC-H" = c(-Inf, Inf), filterId = "singlets")
# insert it between the 'not debris' node and "lymph" node
gs_clone <- insertGate(gs, dummyGate, "not debris", "lymph")
## End(Not run)
```

isGated

The flags of gate nodes isGated checks if a node is already gated isNegated checks if a node is negated. isHidden checks if a node is hidden.

Description

The flags of gate nodes isGated checks if a node is already gated isNegated checks if a node is negated. isHidden checks if a node is hidden.

Usage

```r
isGated(obj, y, 
## S4 method for signature 'GatingHierarchy,character'
isGated(obj, y)

isNegated(obj, y, 
## S4 method for signature 'GatingHierarchy,character'
isNegated(obj, y)

isHidden(obj, y, 
## S4 method for signature 'GatingHierarchy,character'
isHidden(obj, y)
```

Arguments

- `obj` GatingHierarchy
- `y` node/gating path
- `...` not used
isNcdf  
determine the flow data associated with a Gating Hierarchy is based on ‘ncdfFlowSet' or ‘flowSet'

Description

determine the flow data associated with a Gating Hierarchy is based on ‘ncdfFlowSet' or ‘flowSet'

Usage

isNcdf(x)

Arguments

x  
GatingHierarchy object

Value

logical

description

Retrieve a specific keyword for a specific sample in a GatingHierarchy or or set of samples in a GatingSet or GatingSetList

Usage

## S4 method for signature 'GatingHierarchy,character'
keyword(object, keyword)

## S4 method for signature 'GatingHierarchy,missing'
keyword(object, keyword = "missing", ...)

## S4 method for signature 'GatingSet,missing'
keyword(object, keyword = "missing", ...)

## S4 method for signature 'GatingSet,character'
keyword(object, keyword)

## S4 method for signature 'GatingSetList,missing'
keyword(object, keyword = "missing", ...)

## S4 method for signature 'GatingSetList,character'
keyword(object, keyword)
Arguments

object  GatingHierarchy or GatingSet or GatingSetList
keyword  character specifying keyword name. When missing, extract all keywords.
...  other arguments passed to keyword-methods

Details

See keyword in Package ‘flowCore’

See Also

keyword-methods

Examples

## Not run:
# get all the keywords from all samples
keyword(G)
# get all the keywords from one sample
keyword(G[[1]])
# filter the instrument setting
keyword(G[[1]], compact = TRUE)
# get single keyword from all samples
keyword(G, "FILENAME")
# get single keyword from one sample
keyword(G[[1], "FILENAME")

## End(Not run)

---

lapply,GatingSet-method

apply FUN to each sample (i.e. GatingHierarchy)

Description

sample names are used for names of the returned list

Usage

## S4 method for signature 'GatingSet'
lapply(X, FUN, ...)

Arguments

X  GatingSet
FUN  function to be applied to each sample in 'GatingSet'
...  other arguments to be passed to 'FUN'
length,GatingSet-method

Methods to get the length of a GatingSet

Description

Return the length of a GatingSet or GatingSetList object (number of samples).

Usage

```r
## S4 method for signature 'GatingSet'
length(x)
```

```r
## S4 method for signature 'GatingSet'
show(object)
```

Arguments

- `x` GatingSet
- `object` object

logicleGml2_trans

GatingML2 version of logicle transformation.

Description

The only difference from logicle_trans is it is scaled to c(0,1) range.

Usage

```r
logicleGml2_trans(T = 262144, M = 4.5, W = 0.5, A = 0, n = 6,
equal.space = FALSE)
```

Arguments

- `T`, `M`, `W`, `A` see logicleGml2
- `n` desired number of breaks (the actual number will be different depending on the data range)
- `equal.space` whether breaks at equal-spaced intervals

Value

a logicleGml2 transformation object
Examples

trans.obj <- logicleGml2_trans(equal.space = TRUE)
data <- 1:1e3
brks.func <- trans.obj["breaks"]
brks <- brks.func(data)
brks # logicle space displayed at raw data scale
#transform it to verify the equal-spaced breaks at transformed scale
print(trans.obj["transform"](brks))

Description

Used for construct logicle transform object.

Usage

logicle_trans(..., n = 6, equal.space = FALSE)

Arguments

... arguments passed to logicleTransform.
n desired number of breaks (the actual number will be different depending on the
data range)
equal.space whether breaks at equal-spaced intervals

Value

a logicle transformation object

Examples

trans.obj <- logicle_trans(equal.space = TRUE)
data <- 1:1e3
brks.func <- trans.obj["breaks"]
brks <- brks.func(data)
brks # logicle space displayed at raw data scale
#transform it to verify the equal-spaced breaks at transformed scale
print(trans.obj["transform"](brks))
Description

Used to construct flog transformer object. (which uses a specialized flowJo.flog)

Usage

logtGml2_trans(M = 4.5, T = 262144, n = 6, equal.space = FALSE)

Arguments

M  number of decades
T  top scale value
n  desired number of breaks (the actual number will be different depending on the
data range)
equal.space  whether breaks at equal-spaced intervals

Value

logtGml2 transformation object

Examples

trans.obj <- logtGml2_trans(M = 1, T = 1e3, equal.space = TRUE)
data <- 1:1e3
brks.func <- trans.obj["breaks"]
brks <- brks.func(data)
brks # fasinh space displayed at raw data scale

#transform it to verify it is equal-spaced at transformed scale
trans.func <- trans.obj["transform"]
brks.trans <- trans.func(brks)
brks.trans

Description

Get/set the column(channel) or marker names

It simply calls the methods for the underlying flow data (flowSet/ncdfFlowSet/ncdfFlowList).
Usage

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

## S4 replacement method for signature 'GatingHierarchy'
markernames(object) <- value

## S4 method for signature 'GatingHierarchy'
colnames(x, do.NULL = "missing",
          prefix = "missing")

## S4 replacement method for signature 'GatingHierarchy'
colnames(x) <- value

## S4 method for signature 'GatingSet'
markernames(object)

## S4 replacement method for signature 'GatingSet'
markernames(object) <- value

## S4 method for signature 'GatingSet'
colnames(x, do.NULL = "missing",
          prefix = "missing")

## S4 replacement method for signature 'GatingSet'
colnames(x) <- value
```

Arguments

- `value`: named character vector for markernames<-, regular character vector for colnames<-
- `x, object`: GatingHierarchy/GatingSet/GatingSetList
- `do.NULL, prefix`: not used.

Examples

```r
## Not run:
markers.new <- c("CD4", "CD8")
chnls <- c("<B710-A>", "<R780-A>")
names(markers.new) <- chnls
markernames(gs) <- markers.new

chnls <- colnames(gs)
chnls.new <- chnls
chnls.new[c(1,4)] <- c("fsc", "ssc")
colnames(gs) <- chnls.new

## End(Not run)
```
**mkformula**

*make a formula from a character vector*

**Description**

construct a valid formula to be used by flowViz::xyplot

**Usage**

```r
mkformula(dims, isChar = FALSE)
```

**Arguments**

- `dims` a character vector that contains y, x axis, if it is unnamed, then treated as the order of c(y,x)
- `isChar` logical flag indicating whether to return a formula or a pasted string

**Value**

when `isChar` is TRUE, return a character, otherwise coerce it as a formula

**Examples**

```r
all.equal(mkformula(c("SSC-A", "FSC-A")), 'SSC-A' ~ 'FSC-A') # unnamed vector
all.equal(mkformula(c(x = "SSC-A", y = "FSC-A")), 'FSC-A' ~ 'SSC-A') # named vector
```

**moveNode**

*move a node along with all of its descendant nodes to the given ancestor*

**Description**

move a node along with all of its descendant nodes to the given ancestor

**Usage**

```r
moveNode(gh, node, to)
```

**Arguments**

- `gh` GatingHierarchy
- `node` the node to be moved
- `to` the new parent node under which the node will be moved to
Examples

```r
library(flowWorkspace)
dataDir <- system.file("extdata", package="flowWorkspaceData")
suppressMessages(gs <- load_gs(list.files(dataDir, pattern = "gs_manual", full = TRUE)))
gh <- gs[[1]]
old.parent <- getParent(gh, "CD4")
new.parent <- "singlets"
moveNode(gh, "CD4", new.parent)
getParent(gh, "CD4")
```

ncFlowSet

_Fetch the flowData object associated with a GatingSet._

Description

Deprecated by flowData method

Deprecated by flowData method

openWorkspace

_It is now moved along with entire flowJo parser to CytoML package_

Description

_It is now moved along with entire flowJo parser to CytoML package_

Usage

```r
openWorkspace(file, ...)
```

## Default S3 method:
```r
openWorkspace(file, ...)
```

Arguments

- `file` xml file
- `...` other arguments
**pData,GatingHierarchy-method**

*read/set pData of flow data associated with GatingSet or GatingSetList*

---

**Description**

Accessor method that gets or replaces the pData of the flowset/ncdfFlowSet object in a GatingSet or GatingSetList.

**Usage**

```r
## S4 method for signature 'GatingHierarchy'
pData(object)
## S4 method for signature 'GatingSet'
pData(object)
## S4 replacement method for signature 'GatingSet,data.frame'
pData(object) <- value
## S4 replacement method for signature 'GatingSetList,data.frame'
pData(object) <- value
```

**Arguments**

- `object` : GatingSet or GatingSetList
- `value` : data.frame The replacement of pData for flowSet or ncdfFlowSet object

**Value**

a data.frame

---

**plot,GatingSet,missing-method**

*plot a gating tree*

---

**Description**

Plot a tree/graph representing the GatingHierarchy.

**Usage**

```r
## S4 method for signature 'GatingSet,missing'
plot(x, y, ...)
## S4 method for signature 'GatingSet,character'
plot(x, y, ...)
```
Arguments

x  GatingHierarchy or GatingSet. If GatingSet, the first sample will be used to extract gating tree.
y  missing or character specifies.
...
other arguments:
  • boolean: TRUE|FALSE logical specifying whether to plot boolean gate nodes. Defaults to FALSE.
  • showHidden: TRUE|FALSE logical whether to show hidden nodes
  • layout: See layoutGraph in package Rgraphviz
  • width: See layoutGraph in package Rgraphviz
  • height: See layoutGraph in package Rgraphviz
  • fontsize: See layoutGraph in package Rgraphviz
  • labelfontsize: See layoutGraph in package Rgraphviz
  • fixedsize: See layoutGraph in package Rgraphviz

Examples

## Not run:
#gs is a GatingSet
plot(gs) # the same as plot(gs[[1]])
#plot a substree rooted from 'CD4'
plot(gs, "CD4")

## End(Not run)

plotGate

Plot gates and associated cell population contained in a GatingHierarchy or GatingSet

Description

When applied to a GatingHierarchy, arrange is set as TRUE, then all the gates associated with it are plotted as different panel on the same page. If arrange is FALSE, then it plots one gate at a time. By default, merge is set as TRUE, plot multiple gates on the same plot when they share common parent population and axis. When applied to a GatingSet, if lattice is TRUE, it plots one gate (multiple samples) per page, otherwise, one sample (with multiple gates) per page.

Usage

plotGate(x, y, ...)

## S4 method for signature 'GatingHierarchy,numeric'
plotGate(x, y, ...)

## S4 method for signature 'GatingSet,missing'
plotGate(x, y, ...)

## S4 method for signature 'GatingSetList,character'
plotGate(x, y, ...)

plotGate

Plot gates and associated cell population contained in a GatingHierarchy or GatingSet

Description

When applied to a GatingHierarchy, arrange is set as TRUE, then all the gates associated with it are plotted as different panel on the same page. If arrange is FALSE, then it plots one gate at a time. By default, merge is set as TRUE, plot multiple gates on the same plot when they share common parent population and axis. When applied to a GatingSet, if lattice is TRUE, it plots one gate (multiple samples) per page, otherwise, one sample (with multiple gates) per page.

Usage

plotGate(x, y, ...)

## S4 method for signature 'GatingHierarchy,numeric'
plotGate(x, y, ...)

## S4 method for signature 'GatingSet,missing'
plotGate(x, y, ...)

## S4 method for signature 'GatingSetList,character'
plotGate(x, y, ...)

plotGate

Plot gates and associated cell population contained in a GatingHierarchy or GatingSet

Description

When applied to a GatingHierarchy, arrange is set as TRUE, then all the gates associated with it are plotted as different panel on the same page. If arrange is FALSE, then it plots one gate at a time. By default, merge is set as TRUE, plot multiple gates on the same plot when they share common parent population and axis. When applied to a GatingSet, if lattice is TRUE, it plots one gate (multiple samples) per page, otherwise, one sample (with multiple gates) per page.

Usage

plotGate(x, y, ...)

## S4 method for signature 'GatingHierarchy,numeric'
plotGate(x, y, ...)

## S4 method for signature 'GatingSet,missing'
plotGate(x, y, ...)

## S4 method for signature 'GatingSetList,character'
plotGate(x, y, ...)

plotGate
Arguments

x  \textit{GatingSet} or \textit{GatingHierarchy} object
y  character the node name or full/(partial) gating path or numeric representing the node index in the \textit{GatingHierarchy}, or missing which will plot all gates and one gate per page. It is useful for generating plots in a multi-page pdf. Nodes can be accessed with \texttt{getNodes}.

• bool logical specifying whether to plot boolean gates.
• arrange.main character The title of the main page of the plot. Default is the sample name. Only valid when \(x\) is \textit{GatingHierarchy}
• arrange logical indicating whether to arrange different populations/nodes on the same page via \texttt{arrangeGrob} call.
• merge logical indicating whether to draw multiple gates on the same plot if these gates share the same parent population and same x,y dimensions/parameters;
• projections list of character vectors used to customize x,y axis. By default, the x,y axis are determined by the respective gate parameters. The elements of the list are named by the population name or path (see \(y\)). Each element is a pair of named character specifying the channel name (or marker name) for x, y axis. Short form of channel or marker names (e.g. "APC" or "CD3") can be used as long as they can be uniquely matched to the dimensions of flow data. For example, \texttt{projections} = list("lymph" = c(x = "SSC-A", y = "FSC-A"), "CD3" = c(x = "CD3", y = "SSC-A"))
• par.settings list of graphical parameters passed to \texttt{lattice};
• gpar list of grid parameters passed to \texttt{grid.layout};
• lattice logical deprecated;
• formula formula a formula passed to \texttt{xyplot} function of \texttt{flowViz}, by default it is NULL, which means the formula is generated according to the x,y parameters associated with gate.
• cond character the conditioning variable to be passed to lattice plot.
• overlayNode names. These populations are plotted on top of the existing gates (defined by \(y\) argument) as the overlaid dots.
• overlay.symbol A named (lattice graphic parameter) list that defines the symbol color and size for each overlaid population. If not given, we automatically assign the colors.
• keyLattice legend parameter for overlay symbols.
• default.y character specifying y channel for \texttt{xyplot} when plotting a 1d gate. Default is "SSC-A" and session-wise setting can be stored by 'flowWorkspace.par.set("plotGate", list(default.y = "FSC-A"))'
• type character either "xyplot" or "densityplot". Default is "xyplot" and session-wise setting can be stored by 'flowWorkspace.par.set("plotGate", list(type = "xyplot"))'
• fitGate used to disable behavior of plotting the gate region in 1d density-plot. Default is FALSE and session-wise setting can be stored by 'flowWorkspace.par.set("plotGate", list(fitGate = FALSE))'
• strip logical specifies whether to show pop name in strip box, only valid when \(x\) is \textit{GatingHierarchy}
• strip.text either "parent" (the parent population name) or "gate" (the gate name).
• raw.scale logical whether to show the axis in raw(untransformed) scale. Default is TRUE and can be stored as session-wise setting by 'flowWorkspace.par.set("plotGate", list(raw.scale = TRUE))'

• xlim, ylim character can be either "instrument" or "data" which determines the x, y axis scale either by instrument measurement range or the actual data range. or numeric which specifies customized range. They can be stored as session-wise setting by 'flowWorkspace.par.set("plotGate", list(xlim = "instrument"))'

• path A character or numeric scalar passed to getNodes method (used to control how the gating/node path is displayed)

... The other additional arguments to be passed to xyplot.

**Value**

a trellis object if arrange is FALSE,

**References**

http://www.rglab.org/

**Examples**

```r
## Not run:
projections <- list("cd3" = c(x = "cd3", y = "AViD"),
                   "cd4" = c(x = "cd8", y = "cd4"),
                   "cd4/IL2" = c(x = "IL2", y = "IFNg"),
                   "cd4/IFNg" = c(x = "IL2", y = "IFNg"))
plotGate(gh, c("cd3", "cd4", "cd4/IL2", "cd4/IFNg"), path = "auto", projections = projections, gpar = c(nrow = 2))

## End(Not run)
```

## Not run:

```r
#G is a GatingHierarchy
plotGate(G, getNodes(G)[5]); #plot the gate for the fifth node

## End(Not run)

```

---

**plotPopCV, GatingHierarchy-method**

*Plot the coefficient of variation between xml and openCyto population statistics for each population in a gating hierarchy.*

**Description**

This function plots the coefficient of variation calculated between the xml population statistics and the openCyto population statistics for each population in a gating hierarchy extracted from a xml Workspace.
Usage

```r
## S4 method for signature 'GatingHierarchy'
plotPopCV(x, m = 2, n = 2, path = "auto",
          ...)  

## S4 method for signature 'GatingSet'
plotPopCV(x, scales = list(x = list(rot = 90)),
          path = "auto", ...)  
```

Arguments

- `x`: A `GatingHierarchy` from or a `GatingSet`.
- `m`: numeric. The number of rows in the panel plot. Now deprecated, uses lattice.
- `n`: numeric. The number of columns in the panel plot. Now deprecated, uses lattice.
- `path`: character. See `getNodes`.
- `...`: Additional arguments to the `barplot` methods.
- `scales`: list. See `barchart`.

Details

The CVs are plotted as barplots across panels on a grid of size `m` by `n`.

Value

Nothing is returned.

See Also

- `getPopStats`

Examples

```r
## Not run:
#G is a GatingHierarchy
plotPopCV(G,4,4);
## End(Not run)
```

plot_diff_tree

visualize the tree structure difference among the GatingSets

Description

visualize the tree structure difference among the GatingSets

Usage

```r
plot_diff_tree(x, path = "auto", ...)  
```
Arguments

- `x`: list of groups (each group is a list of 'GatingSet'). It is usually the outcome from `groupByTree`.
- `path`: passed to `getNodes`
- `...`: passed to `getNodes`

Examples

```r
## Not run:
gslist <- list(gs1, gs2, gs3, gs4, gs5)
gs_groups <- groupByTree(gslist)
plot_diff_tree(gs_groups)
## End(Not run)
```

Description

`pop.MFI` computes and returns the median fluorescence intensity for each marker. They are typically used as the arguments passed to `getStats` method to perform the sample-wise population stats calculations.

Usage

`pop.MFI(fr)`

Arguments

- `fr`: a `flowFrame` represents a gated population

Value

a named numeric vector

prettyAxis

* Determine tick mark locations and labels for a given channel axis*

Description

Determine tick mark locations and labels for a given channel axis

Usage

`prettyAxis(gh, channel)`
recompute,GatingSet-method

Arguments

<table>
<thead>
<tr>
<th>gh</th>
<th>GatingHierarchy</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel</td>
<td>character channel name</td>
</tr>
</tbody>
</table>

Value

when there is transformation function associated with the given channel, it returns a list of that contains positions and labels to draw on the axis other wise returns NULL.

Examples

```r
## Not run:
prettyAxis(gh, "<B710-A>")
## End(Not run)
```

recompute,GatingSet-method

*Compute the cell events by the gates stored within the gating tree.*

Description

Compute each cell event to see if it falls into the gate stored within the gating tree and store the result as cell count.

Usage

```r
## S4 method for signature 'GatingSet'
recompute(x, y = "root", alwaysLoadData = FALSE, ...)

## S4 method for signature 'GatingSetList'
recompute(x, ...)
```

Arguments

<table>
<thead>
<tr>
<th>x</th>
<th>GatingSet</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>character node name or node path. Default &quot;root&quot;. Optional.</td>
</tr>
<tr>
<td>alwaysLoadData</td>
<td>logical. Specifies whether to load the flow raw data for gating boolean gates. Default 'FALSE'. Optional. Sometime it is more efficient to skip loading the raw data if all the reference nodes and parent are already gated. 'FALSE' will check the parent node and reference to determine whether to load the data. This check may not be sufficient since the further upstream ancestor nodes may not be gated yet. In that case, we allow the gating to fail and prompt user to recompute those nodes explicitly. When TRUE, then it forces data to be loaded to guarantee the gating process to be uninterrupted at the cost of unnecessary data IO.</td>
</tr>
<tr>
<td>...</td>
<td>other arguments leaf.bool whether to compute the leaf boolean gate, default is TRUE</td>
</tr>
</tbody>
</table>

Details

It is usually used immediately after add or setGate calls.
rotate_gate

Simplified geometric rotation of gates associated with nodes

Description

Rotate a gate associated with a node of a GatingHierarchy or GatingSet. This method is a wrapper for rotate_gate that enables updating of the gate associated with a node of a GatingHierarchy or GatingSet.

rotate_gate calls setGate to modify the provided GatingHierarchy or GatingSet directly so there is no need to re-assign its output. The arguments will be essentially identical to the flowCore method, except for the specification of the target gate. Rather than being called on an object of type flowCore:filter, here it is called on a GatingHierarchy or GatingSet object with an additional character argument for specifying the node whose gate should be transformed. The rest of the details below are taken from the flowCore documentation.

Usage

```r
## S3 method for class 'GatingHierarchy'
rotate_gate(obj, y, deg = NULL,
    rot_center = NULL, ...)

## S3 method for class 'GatingSet'
rotate_gate(obj, y, deg = NULL, rot_center = NULL,
    ...)
```

Arguments

- `obj` A GatingHierarchy or GatingSet object
- `y` A character specifying the node whose gate should be modified
- `deg` An angle in degrees by which the gate should be rotated in the counter-clockwise direction
- `rot_center` A separate 2-dimensional center of rotation for the gate, if desired. By default, this will be the center for ellipsoidGate objects or the centroid for polygonGate objects. The rot_center argument is currently only supported for polygonGate objects.
- `...` not used

Details

This method allows for geometric rotation of filter types defined by simple geometric gates (ellipsoidGate, and polygonGate). The method is not defined for rectangleGate or quadGate objects, due to their definition as having 1-dimensional boundaries.

The angle provided in the deg argument should be in degrees rather than radians. By default, the rotation will be performed around the center of an ellipsoidGate or the centroid of the area encompassed by a polygonGate. The rot_center argument allows for specification of a different center of rotation for polygonGate objects (it is not yet implemented for ellipsoidGate objects) but it is usually simpler to perform a rotation and a translation individually than to manually specify the composition as a rotation around a shifted center.
See Also

transform_gate flowCore::rotate_gate

Examples

## Not run:
#
# Rotates the original gate 15 degrees counter-clockwise
scale_gate(gs, node, deg = 15)
# Rotates the original gate 270 degrees counter-clockwise
scale_gate(gs, node, 270)
## End(Not run)

### sampleNames,GatingHierarchy-method

Get(update) sample names in a GatingSet

Description

Return a sample names contained in a GatingSet

Usage

## S4 method for signature 'GatingHierarchy'
sampleNames(object)

## S4 method for signature 'GatingSet'
sampleNames(object)

## S4 replacement method for signature 'GatingSet'
sampleNames(object) <- value

Arguments

object or a GatingSet

value character new sample names

Details

The sample names comes from pdata of fs.

Value

A character vector of sample names

Examples

## Not run:
#G is a GatingSet
sampleNames(G)
## End(Not run)
save_gs

save/load a GatingSet/GatingSetList to/from disk.

Description
Save/load a GatingSet/GatingSetList which is the gated flow data including gates and populations to/from the disk. The GatingSet object The internal C data structure (gating tree), ncdfFlowSet object(if applicable)

Usage
save_gs(G, path, overwrite = FALSE, cdf = c("copy", "move", "skip", "symlink", "link"), ...)
load_gs(path)

save_gslist(gslist, path, ...)
load_gslist(path)

Arguments
G A GatingSet
path A character scalar giving the path to save/load the GatingSet to/from.
overwrite A logical scalar specifying whether to overwrite the existing folder.
cdf a character scalar. The valid options are :"copy","move","skip","symlink","link" specifying what to do with the cdf data file. Sometime it is more efficient to move or create a link of the existing cdf file to the archived folder. It is useful to "skip" archiving cdf file if raw data has not been changed.
...
other arguments: not used.
gslist A GatingSetList

Value
load_gs returns a GatingSet object load_gslist returns a GatingSetList object

See Also
GatingSet-class, GatingSetList-class

Examples
## Not run:
#G is a GatingSet
save_gs(G,path="tempFolder")
G<-load_gs(path="tempFolder")

#G is a GatingSet

save_gslist(gslist1,path="tempFolder")
gslist2<-load_gslist(path="tempFolder")
### Description

Simplified geometric scaling of gates associated with nodes

### Usage

```r
## S3 method for class 'GatingHierarchy'
scale_gate(obj, y, scale = NULL, ...)
```

```r
## S3 method for class 'GatingSet'
scale_gate(obj, y, scale = NULL, ...)
```

### Arguments

- `obj` A `GatingHierarchy` or `GatingSet` object
- `y` A character specifying the node whose gate should be modified
- `scale` Either a numeric scalar (for uniform scaling in all dimensions) or numeric vector specifying the factor by which each dimension of the gate should be expanded (absolute value > 1) or contracted (absolute value < 1). Negative values will result in a reflection in that dimension.
- `...` not used

### Details

This method allows uniform or non-uniform geometric scaling of filter types defined by simple geometric gates (`quadGate`, `rectangleGate`, `ellipsoidGate`, and `polygonGate`) Note that these methods are for manually altering the geometric definition of a gate. To easily transform the definition of a gate with an accompanying scale transformation applied to its underlying data, see `?ggcyto::rescale_gate`.

The `scale` argument passed to `scale_gate` should be either a scalar or a vector of the same length as the number of dimensions of the gate. If it is scalar, all dimensions will be multiplicatively scaled uniformly by the scalar factor provided. If it is a vector, each dimension will be scaled by its corresponding entry in the vector.

The scaling behavior of `scale_gate` depends on the type of gate passed to it. For `rectangleGate` and `quadGate` objects, this amounts to simply scaling the values of the 1-dimensional boundaries. For `polygonGate` objects, the values of `scale` will be used to determine scale factors in the direction of each of the 2 dimensions of the gate (`scale_gate` is not yet defined for higher-dimensional polytopeGate objects). **Important**: For `ellipsoidGate` objects, `scale` determines scale factors for the major and minor axes of the ellipse, *in that order*. Scaling by a negative factor will result in a reflection in the corresponding dimension.

### See Also

`transform_gate flowCore::scale_gate`
Examples

```r
## Not run:
# Scales both dimensions by a factor of 5
scale_gate(gs, node, 5)

# Shrinks the gate in the first dimension by factor of 1/2
# and expands it in the other dimension by factor of 3
scale_gate(gs, node, c(0.5, 3))

## End(Not run)
```

Description

It is for internal use by the diva parser

Usage

```r
set.count.xml(gh, node, count)
```

Arguments

- `gh`: GatingHierarchy
- `node`: the unique gating path that uniquely identifies a population node
- `count`: integer number that is events count for the respective gating node directly parsed from xml file

Examples

```r
## Not run:
set.count.xml(gh, "CD3", 10000)

## End(Not run)
```

Description

update the population node with a flowCore-compatible gate object
Usage

## S4 method for signature 'GatingHierarchy,character,filter'
setGate(obj, y, value,
   negated = FALSE, ...)

## S4 method for signature 'GatingSet,character,list'
setGate(obj, y, value, ...)

## S4 method for signature 'GatingSet,character,filterList'
setGate(obj, y, value, ...)

Arguments

obj GatingHierarchy or GatingSet
y character node name or path
value filter or filterList or list of filter objects
negated logical see add
... other arguments

Details

Usually recompute is followed by this call since updating a gate doesn’t re-calculating the cell events within the gate automatically. see filterObject for the gate types that are currently supported.

Examples

## Not run:
rg1 <- rectangleGate("FSC-H"=c(200,400), "SSC-H"=c(250, 400), filterId="rectangle")
rg2 <- rectangleGate("FSC-H"=c(200,400), "SSC-H"=c(250, 400), filterId="rectangle")
flist <- list(rg1,rg2)
names(flist) <- sampleNames(gs[1:2])
setGate(gs[1:2], "lymph", flist)
recompute(gs[1:2], "lymph")

## End(Not run)

setNode,GatingHierarchy,character,character-method

Update the name of one node in a gating hierarchy/GatingSet.

Description

setNode update the name of one node in a gating hierarchy/GatingSet.

Usage

## S4 method for signature 'GatingHierarchy,character,character'
setNode(x, y, value)

## S4 method for signature 'GatingHierarchy,character,logical'
setNode(x, y, value)

## S4 method for signature 'GatingSet,character,ANY'
setNode(x, y, value)

### Arguments

- **x**: GatingHierarchy object
- **y**: character node name or path
- **value**: A character the name of the node. or logical to indicate whether to hide a node

### Examples

```r
## Not run:
#G is a gating hierarchy
getNodes(G[[1]])#return node names
setNode(G,"L","lymph")

## End(Not run)
## Not run:
## End(Not run)
```

---

**shift_gate**

*Simplified geometric translation of gates associated with nodes*

### Description

Shift the location of a gate associated with a node of a GatingHierarchy or GatingSet. This method is a wrapper for `shift_gate` that enables updating of the gate associated with a node of a GatingHierarchy or GatingSet.

`shift_gate` calls `setGate` to modify the provided GatingHierarchy or GatingSet directly so there is no need to re-assign its output. The arguments will be essentially identical to the `flowCore` method, except for the specification of the target gate. Rather than being called on an object of type `flowCore::filter`, here it is called on a GatingHierarchy or GatingSet object with an additional character argument for specifying the node whose gate should be transformed. The rest of the details below are taken from the FlowCore documentation.

### Usage

```r
## S3 method for class 'GatingHierarchy'
shift_gate(obj, y, dx = NULL, dy = NULL, center = NULL, ...)

## S3 method for class 'GatingSet'
shift_gate(obj, y, dx = NULL, dy = NULL, center = NULL, ...)
```
Arguments

- `obj`: A `GatingHierarchy` or `GatingSet` object.
- `y`: A character specifying the node whose gate should be modified.
- `dx`: Either a numeric scalar or numeric vector. If it is scalar, this is just the desired shift of the gate in its first dimension. If it is a vector, it specifies both `dx` and `dy` as `(dx, dy)`. This provides an alternate syntax for shifting gates, as well as allowing shifts of `ellipsoidGate` objects in more than 2 dimensions.
- `dy`: A numeric scalar specifying the desired shift of the gate in its second dimension.
- `center`: A numeric vector specifying where the center or centroid should be moved (rather than specifying `dx` and/or `dy`).

Details

This method allows for geometric translation of filter types defined by simple geometric gates (`rectangleGate`, `quadGate`, `ellipsoidGate`, or `polygonGate`). The method provides two approaches to specify a translation. For `rectangleGate` objects, this will shift the min and max bounds by the same amount in each specified dimension. For `quadGate` objects, this will simply shift the dividing boundary in each dimension. For `ellipsoidGate` objects, this will shift the center (and therefore all points of the ellipse). For `polygonGate` objects, this will simply shift all of the points defining the polygon.

The method allows two different approaches to shifting a gate. Through the `dx` and/or `dy` arguments, a direct shift in each dimension can be provided. Alternatively, through the `center` argument, the gate can be directly moved to a new location in relation to the old center of the gate. For `quadGate` objects, this center is the intersection of the two dividing boundaries (so the value of the `boundary` slot). For `rectangleGate` objects, this is the center of the rectangle defined by the intersections of the centers of each interval. For `ellipsoidGate` objects, it is the center of the ellipsoid, given by the `mean` slot. For `polygonGate` objects, the centroid of the old polygon will be calculated and shifted to the new location provided by `center` and all other points on the polygon will be shifted by relation to the centroid.

See Also

- `transform_gate`

Examples

```r
## Not run:
# Moves the entire gate +500 in its first dimension and 0 in its second dimension
shift_gate(gs, node, dx = 500)

# Moves the entire gate +250 in its first dimension and +700 in its second dimension
shift_gate(gs, node, dx = 500, dy = 700)

# Same as previous
shift_gate(gs, node, c(500, 700))

# Move the gate based on shifting its center to (700, 1000)
shift_gate(gs, node, center = c(700, 1000))

## End(Not run)
```
The tools to standardize the tree structures and channel names.

**Description**

- `groupByTree(x)`: groups GatingSets based on their gating tree structures.
- `groupByChannels(x)`: groups GatingSets based on their flow channels.
- `checkRedundantNodes(x)`: identifies terminal (or leaf) nodes that make the gating trees different among GatingSets and can be considered redundant.
- `dropRedundantNodes(x, toRemove)`: removes the terminal (or leaf) nodes identified as redundant.
- `dropRedundantChannels(gs)`: removes redundant channels not used by any gate.
- `updateChannels(gs, map, all = TRUE)`: modifies channel names in place to standardize.
- `insertGate(gs, gate, parent, children)`: inserts a dummy gate to handle unlisted non-leaf nodes.
- `setNode(x, y, FALSE)`: hides a node/gate in GatingSets.

**Details**

In order to merge multiple GatingSets into a single GatingSetList, their gating trees and channel names must be consistent. These functions help remove discrepancies and standardize GatingSets so that they can be merged.

- `groupByTree` splits GatingSets into groups based on their gating trees.
- `groupByChannels` splits GatingSets into groups based on their flow channels.
- `checkRedundantNodes` returns terminal (or leaf) nodes that make the gating trees different among GatingSets and can be considered redundant.
- `dropRedundantNodes` removes detected redundant nodes.
- `dropRedundantChannels` removes redundant channels not used by any gate.
- `updateChannels` modifies channel names in place (usually used to standardize channel differences due to case or typo issues).
- `insertGate` inserts a dummy gate to handle extra non-leaf nodes.
- `setNode` hides a node/gate in GatingSets.

**subset.GatingSet**

Subset the GatingSet/GatingSetList based on `pData`.

**Description**

Subset the GatingSet/GatingSetList based on `pData`.

**Usage**

```r
## S3 method for class 'GatingSet'
subset(x, subset, ...)```

---

**standardize-GatingSet**

The tools to standardize the tree structures and channel names.

**Description**

- `groupByTree(x)`: groups GatingSets based on their gating tree structures.
- `groupByChannels(x)`: groups GatingSets based on their flow channels.
- `checkRedundantNodes(x)`: identifies terminal (or leaf) nodes that make the gating trees different among GatingSets and can be considered redundant.
- `dropRedundantNodes(x, toRemove)`: removes the terminal (or leaf) nodes identified as redundant.
- `dropRedundantChannels(gs)`: removes redundant channels not used by any gate.
- `updateChannels(gs, map, all = TRUE)`: modifies channel names in place to standardize.
- `insertGate(gs, gate, parent, children)`: inserts a dummy gate to handle unlisted non-leaf nodes.
- `setNode(x, y, FALSE)`: hides a node/gate in GatingSets.

**Details**

In order to merge multiple GatingSets into a single GatingSetList, their gating trees and channel names must be consistent. These functions help remove discrepancies and standardize GatingSets so that they can be merged.

- `groupByTree` splits GatingSets into groups based on their gating trees.
- `groupByChannels` splits GatingSets into groups based on their flow channels.
- `checkRedundantNodes` returns terminal (or leaf) nodes that make the gating trees different among GatingSets and can be considered redundant.
- `dropRedundantNodes` removes detected redundant nodes.
- `dropRedundantChannels` removes redundant channels not used by any gate.
- `updateChannels` modifies channel names in place (usually used to standardize channel differences due to case or typo issues).
- `insertGate` inserts a dummy gate to handle extra non-leaf nodes.
- `setNode` hides a node/gate in GatingSets.

**subset.GatingSet**

Subset the GatingSet/GatingSetList based on `pData`.

**Description**

Subset the GatingSet/GatingSetList based on `pData`.

**Usage**

```r
## S3 method for class 'GatingSet'
subset(x, subset, ...)```
swap_data_cols

Arguments
x  GatingSet or GatingSetList
subset logical expression (within the context of pData) indicating samples to keep. see subset

Value
a codeGatingSet or GatingSetList object

Description
Swap the colnames Perform some validity checks before returning the updated colnames

Usage
swap_data_cols(cols, swap_cols)

Arguments
cols the original colname vector
swap_cols a named list specifying the pairs to be swapped

Value
the new colname vector that has some colnames swapped

Examples

data(GvHD)
fr <- GvHD[[1]]
colnames(fr)
new <- swap_data_cols(colnames(fr), list("FSC-H" = "SSC-H", "FL2-H" = "FL2-A"))
colnames(fr) <- new
**transform,GatingSet-method**

Transform the flow data associated with the GatingSet

---

**Description**

The transformation functions are saved in the GatingSet and can be retrieved by `getTransformations`. Currently only flowJo-type biexponential transformation (either returned by `getTransformations` or constructed by `flowJoTrans`) is supported.

**Usage**

```r
## S4 method for signature 'GatingSet'
transform(.data, translist, ...)

## S4 method for signature 'GatingSetList'
transform(.data, ...)
```

**Arguments**

- `.data` GatingSet or GatingSetList
- `translist` expect a transformList object or a list of transformList objects (with names matched to sample names)
- `...` other arguments passed to `transform` method for `ncdfFlowSet`. (e.g. `ncdf-File`)

**Value**

A GatingSet or GatingSetList object with the underlying flow data transformed.

**Examples**

```r
## Not run:
data(GvHD)
fs <- GvHD[1:2]
gs <- GatingSet(fs)

# construct biexponential transformation function
biexpTrans <- flowJo_biexp_trans(channelRange=4096, maxValue=262144, pos=4.5, neg=0, widthBasis=-10)

# make a transformList object
chnls <- c("FL1-H", "FL2-H")
transList <- transformerList(chnls, biexpTrans)

# add it to GatingSet
gs_trans <- transform(gs, transList)

## End(Not run)
```
transformerList  Constructor for transformerList object

Description

Similar to transformList function, it constructs a list of transformer objects generated by trans_new method from scales so that the inverse and breaks functions are also included.

Usage

transformerList(from, trans)

Arguments

from  channel names
trans  a trans object or a list of trans objects constructed by trans_new method.

Examples

library(scales)
#create transformer object from scratch
trans <- logicleTransform(w = 0.5, t = 262144, m = 4.5, a = 0)
inv <- inverseLogicleTransform(trans = trans)
trans.obj <- flow_trans(“logicle”, trans, inv, n = 5, equal.space = FALSE)

#or simply use convenient constructor
#trans.obj <- logicle_trans(n = 5, equal.space = FALSE, w = 0.5, t = 262144, m = 4.5, a = 0)

transformerList(c(“FL1-H”, “FL2-H”), trans.obj)

#use different transformer for each channel
trans.obj2 <- asinhGml2_trans()
transformerList(c(“FL1-H”, “FL2-H”), list(trans.obj, trans.obj2))

transform_gate  Simplified geometric transformations of gates associated with nodes

Description

Perform geometric transformations of a gate associated with a node of a GatingHierarchy or GatingSet. This method is a wrapper for transform_gate that enables updating of the gate associated with a node of a GatingHierarchy or GatingSet.

transform_gate calls setGate to modify the provided GatingHierarchy or GatingSet directly so there is no need to re-assign its output. The arguments will be essentially identical to the flowCore method, except for the specification of the target gate. Rather than being called on an object of type flowCore::filter, here it is called on a GatingHierarchy or GatingSet object with an additional character argument for specifying the node whose gate should be transformed. The rest of the details below are taken from the flowCore documentation.
transform_gate

Usage

```r
## S3 method for class 'GatingHierarchy'
transform_gate(obj, y, scale = NULL,
               deg = NULL, rot_center = NULL, dx = NULL, dy = NULL,
               center = NULL, ...)
```

Arguments

- **obj**: A GatingHierarchy or GatingSet object
- **y**: A character specifying the node whose gate should be modified
- **scale**: Either a numeric scalar (for uniform scaling in all dimensions) or numeric vector specifying the factor by which each dimension of the gate should be expanded (absolute value > 1) or contracted (absolute value < 1). Negative values will result in a reflection in that dimension.
  
  For `rectangleGate` and `quadGate` objects, this amounts to simply scaling the values of the 1-dimensional boundaries. For `polygonGate` objects, the values of `scale` will be used to determine scale factors in the direction of each of the 2 dimensions of the gate (`scale_gate` is not yet defined for higher-dimensional `polytopyeGate` objects). **Important**: For `ellipsoidGate` objects, `scale` determines scale factors for the major and minor axes of the ellipse, in that order.

- **deg**: An angle in degrees by which the gate should be rotated in the counter-clockwise direction.

- **rot_center**: A separate 2-dimensional center of rotation for the gate, if desired. By default, this will be the center for `ellipsoidGate` objects or the centroid for `polygonGate` objects. The `rot_center` argument is currently only supported for `polygonGate` objects. It is also usually simpler to perform a rotation and a translation individually than to manually specify the composition as a rotation around a shifted center.

- **dx**: Either a numeric scalar or numeric vector. If it is scalar, this is just the desired shift of the gate in its first dimension. If it is a vector, it specifies both `dx` and `dy` as `(dx, dy)`. This provides an alternate syntax for shifting gates, as well as allowing shifts of `ellipsoidGate` objects in more than 2 dimensions.

- **dy**: A numeric scalar specifying the desired shift of the gate in its second dimension.

- **center**: A numeric vector specifying where the center or centroid should be moved (rather than specifying `dx` and/or `dy`)

- **...**: Assignments made to the slots of the particular Gate-type filter object in the form "<slot_name> = <value>"

Details

This method allows changes to the four filter types defined by simple geometric gates (`quadGate`, `rectangleGate`, `ellipsoidGate`, and `polygonGate`) using equally simple geometric transformations (shifting/translation, scaling/dilation, and rotation). The method also allows for directly resetting the slots of each Gate-type object. Note that these methods are for manually altering the geometric definition of a gate. To easily transform the definition of a gate with an accompanying scale transformation applied to its underlying data, see `ggcyto::rescale_gate`.

First, `transform_gate` will apply any direct alterations to the slots of the supplied Gate-type filter object. For example, if "mean = c(1,3)" is present in the argument list when `transform_gate` is called on an `ellipsoidGate` object, the first change applied will be to shift the mean slot to (1,3).
The method will carry over the dimension names from the gate, so there is no need to provide column or row names with arguments such as mean or cov for ellipsoidGate or boundaries for polygonGate.

transform_gate then passes the geometric arguments (dx, dy, deg, rot_center, scale, and center) to the methods which perform each respective type of transformation: shift_gate, scale_gate, or rotate_gate. The order of operations is to first scale, then rotate, then shift. The default behavior of each operation follows that of its corresponding method but for the most part these are what the user would expect. A few quick notes:

- rotate_gate is not defined for rectangleGate or quadGate objects, due to their definition as having 1-dimensional boundaries.
- The default center for both rotation and scaling of a polygonGate is the centroid of the polygon. This results in the sort of scaling most users expect, with a uniform scale factor not distorting the shape of the original polygon.

See Also
flowCore::transform_gate

Examples

```r
## Not run:
# Scale the original gate non-uniformly, rotate it 15 degrees, and shift it
transform_gate(gs, node, scale = c(2,3), deg = 15, dx = 500, dy = -700)

# Scale the original gate (in this case an ellipsoidGate) after moving its center to (1500, 2000)
transform_gate(gs, node, scale = c(2,3), mean = c(1500, 2000))

## End(Not run)
```

updateChannels
Update the channel information of a GatingSet (c++ part)

Description
It updates the channels stored in gates, compensations and transformations based on given mapping between the old and new channel names.

Usage
updateChannels(gs, map, all = TRUE)

Arguments

- **gs**: a GatingSet object
- **map**: data.frame contains the mapping from old (case insensitive) to new channel names Note: Make sure to remove the ‘<’ or ‘>’ characters from ‘old’ name because the API tries to only look at the raw channel name so that the gates with both prefixed and non-prefixed names could be updated.
- **all**: logical whether to update the flow data as well
updateIndices, GatingHierarchy, character, logical-method

**Value**

when 'all' is set to TRUE, it returns a new GatingSet but it still shares the same underlying C++ tree structure with the original GatingSet otherwise it returns nothing (less overhead.)

**Examples**

```r
## Not run:
## this will update both "Qdot 655-A" and "<Qdot 655-A>"
gs <- updateChannels(gs, map = data.frame(old = c("Qdot 655-A"),
                                          new = c("QDot 655-A"))
)
## End(Not run)
```

updateIndices, GatingHierarchy, character, logical-method
directly update event indices without changing gates

**Description**

It is useful when we want to alter the population at events level yet without removing or adding the existing gates.

**Usage**

```r
## S4 method for signature 'GatingHierarchy, character, logical'
updateIndices(obj, y, z)
```

**Arguments**

- **obj** GatingHierarchy object
- **y** character node name or path
- **z** logical vector as local event indices relative to node y

**Examples**

```r
library(flowWorkspace)
dataDir <- system.file("extdata", package = "flowWorkspaceData")
suppressMessages(gs <- load_gs(list.files(dataDir, pattern = "gs_manual", full = TRUE)))
gh <- gs[[1]]
# get pop counts
pop.stats <- getStats(gh, nodes = c("CD3+", "CD4", "CD8"))
pop.stats

# subsample 30% cell events at CD3+ node
total.cd3 <- pop.stats[pop == "CD3+", count]
gInd <- seq_len(total.cd3) # create integer index for cd3
rand <- sample.int(total.cd3, size = total.cd3 * 0.3) # randomly select 30%
# convert it to logical index
ghInd.logical <- rep(FALSE, total.cd3)
ghInd.logical[gInd] <- TRUE
```
#replace the original index stored at GatingHierarchy
updateIndices(gh, "CD3+", gInd.logical)
#check the updated pop counts
getStats(gs[[1]], nodes = c("CD3+, "CD4", "CD8")) #note that CD4, CD8 are not updated
#update all the descendants of CD3+
nodes <- getDescendants(gh, "CD3+")
for (node in nodes) suppressMessages(recompute(gh, node))
getStats(gs[[1]], nodes = c("CD3+, "CD4", "CD8")) #now all are update to date
Index

[,GatingSet, ANY-method (GatingSet-class), 25
[,GatingSetList, ANY-method (GatingSet-class), 25
[,GatingSet, character-method (GatingSet-class), 25
[,GatingSet, logical-method (GatingSet-class), 25
[,GatingSet, numeric-method (GatingSet-class), 25
add, 9, 62, 68
add (add, GatingSet, list-method), 4
add, GatingHierarchy, factor-method (add, GatingSet, list-method), 4
add, GatingHierarchy, filter-method (add, GatingSet, list-method), 4
add, GatingHierarchy, filters-method (add, GatingSet, list-method), 4
add, GatingHierarchy, logical-method (add, GatingSet, list-method), 4
add, GatingHierarchy, logicalFilterResult-method (add, GatingSet, list-method), 4
add, GatingHierarchy, multipleFilterResult-method (add, GatingSet, list-method), 4
add, GatingHierarchy, quadGate-method (add, GatingSet, list-method), 4
add, GatingSet, filter-method (add, GatingSet, list-method), 4
add, GatingSet, filterList-method (add, GatingSet, list-method), 4
add, GatingSet, filters-method (add, GatingSet, list-method), 4
add, GatingSet, filtersList-method (add, GatingSet, list-method), 4
add, GatingSet, list-method, 4
add, GatingSetList, filter-method (add, GatingSet, list-method), 4
add, GatingSetList, filterList-method (add, GatingSet, list-method), 4
add, GatingSetList, filters-method (add, GatingSet, list-method), 4
add, GatingSetList, filtersList-method (add, GatingSet, list-method), 4
add, GatingSetList, list-method (add, GatingSet, list-method), 4
asin_Gml2, 8
asinhtGml2_trans, 8
barchart, 60
booleanFilter (booleanFilter-class), 9
booleanFilter-class, 9
char2booleanFilter (booleanFilter-class), 9
checkRedundantNodes, 10, 14, 71
collect, 11
collectNodes (clone), 11
collect-methods (clone), 11
collect, ncdfFlowSet, 11
colnames, GatingHierarchy-method
(markernames, GatingHierarchy-method), 52
colnames, GatingSet-method (markernames, GatingHierarchy-method), 52
colnames<-, GatingHierarchy-method (markernames, GatingHierarchy-method), 52
colnames<-, GatingSet-method
(markernames, GatingHierarchy-method), 52
compensate, GatingSet, ANY-method, 11
compensate, GatingSetList, ANY-method (compensate, GatingSet, ANY-method), 11
compute_timestep, 12
copyNode, 13
dropRedundantChannels, 13, 71
dropRedundantNodes, 10, 14, 71
ellipsoidGate, 63, 66, 70, 75
estimateLogicle.GatingHierarchy, 14
expressionFilter, 9
extract_cluster_pop_name_from_node, 15
filterObject, 68

79
INDEX

logtGml2, 18
logtGml2_trans, 52
markernames,GatingHierarchy-method, 52
markernames,GatingSet-method
(markernames,GatingHierarchy-method), 52
markernames<-,GatingHierarchy-method
(markernames,GatingHierarchy-method), 52
markernames<-,GatingSet-method
(markernames,GatingHierarchy-method), 52
merge-GatingSet
(standardize-GatingSet), 71
mkformula, 54
moveNode, 54
ncFlowSet, 55
ncFlowSet,GatingSet-method (ncFlowSet), 55
ncFlowSet<-(ncFlowSet), 55
ncFlowSet<-,GatingSet-method
(ncFlowSet), 55
openWorkspace, 55
pData (pData,GatingHierarchy-method), 56
pData,GatingHierarchy-method, 56
pData,GatingSet-method
(pData,GatingHierarchy-method), 56
pData<-(pData,GatingHierarchy-method), 56
pData<-,GatingSet,data.frame-method
(pData,GatingHierarchy-method), 56
pData<-,GatingSetList,data.frame-method
(pData,GatingHierarchy-method), 56
plot (plot,GatingSet,missing-method), 56
plot,GatingSet,character-method
(plot,GatingSet,missing-method), 56
plot,GatingSet,missing-method, 56
plot_diff_tree, 60
plotGate, 57
plotGate,GatingHierarchy,character-method
(plotGate), 57
plotGate,GatingHierarchy,missing-method
(plotGate), 57
plotGate,GatingHierarchy,numerical-method
(plotGate), 57
plotGate,GatingSet,character-method
(plotGate), 57
plotGate,GatingSet,missing-method
(plotGate), 57
plotGate,GatingSet,numerical-method
(plotGate), 57
plotPopCV
(plotPopCV,GatingHierarchy-method), 59
plotPopCV,GatingHierarchy-method, 59
plotPopCV,GatingSet-method
(plotPopCV,GatingHierarchy-method), 59
polygonGate, 63, 66, 70, 75
pop.MFI, 61
prettyAxis, 61
quadGate, 66, 75
rbind2,GatingSetList,missing-method
(GatingSetList-class), 27
recompute, 68
recompute (recompute,GatingSet-method), 62
recompute,GatingSet-method, 62
recompute,GatingSetList-method
(recompute,GatingSet-method), 62
rectangleGate, 66, 75
Rm (add,GatingSet,list-method), 4
Rm,character,GatingHierarchy,character-method
(add,GatingSet,list-method), 4
Rm,character,GatingSet,character-method
(add,GatingSet,list-method), 4
Rm,character,GatingSetList,character-method
(add,GatingSet,list-method), 4
rotate_gate, 63, 63, 76
sampleNames
(sampleNames,GatingHierarchy-method), 64
sampleNames,GatingHierarchy-method, 64
sampleNames,GatingSet-method
(sampleNames,GatingHierarchy-method), 64
sampleNames<-
(sampleNames,GatingHierarchy-method), 64
sampleNames<-,GatingSet,ANY-method
(sampleNames,GatingHierarchy-method), 64
INDEX

sampleNames<-,GatingSet-method
 (sampleNames,GatingHierarchy-method)
64

set.count.xml
67

setGate
 (setGate,GatingHierarchy,character,filter-method),
67

setLoglevel (getLoglevel), 35

setNode
 (setNode,GatingHierarchy,character,character-method),
68

setNode,GatingHierarchy,character,logical-method
 (setNode,GatingHierarchy,character,character-method),
68

setNode,GatingSet,character,ANY-method
 (setNode,GatingHierarchy,character,character-method),
68

shift_gate, 69, 69, 76

show,booleanFilter-method
 (booleanFilter-class), 9

show,GatingHierarchy-method
 (GatingHierarchy-class), 24

show,GatingSet-method
 (length,GatingSet-method), 50

standardize-GatingSet

subset, 72

subset,GatingSet, 71

swap_data_cols, 72

transform,GatingSet-method, 73

transform,GatingSetList-method
 (transform,GatingSet-method),
73

transform_gate, 74, 74

transformerList, 74

updateChannels, 71, 76

xymplot, 59