Package ‘flowWorkspace’

November 9, 2019

Type Package

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

Version 3.35.0

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, grDevices, lattice, methods, stats, stats4, utils, RBGL, tools, gridExtra, Rgraphviz, data.table, dplyr, latticeExtra, Rcpp, RColorBrewer, stringr, scales, flowViz, matrixStats, digest, RcppParallel, flowCore(>= 1.51.4), ncdfFlow(>= 2.31.1)


biocViews ImmunoOncology, FlowCytometry, DataImport, Preprocessing, DataRepresentation

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

LinkingTo Rcpp, BH(>= 1.62.0-1), RProtoBufLib(>= 1.7.1), cytolib(>= 1.7.4), RcppParallel

VignetteBuilder knitr

SystemRequirements GNU make, C++11
R topics documented:

flowWorkspace-package .................................................. 4
add ................................................................. 4
asinhGml2_trans......................................................... 6
asinh_Gml2 ............................................................ 7
booleanFilter-class ..................................................... 8
checkRedundantNodes .................................................. 9
copyNode ............................................................. 9
compensate,GatingSet,ANY-method ................................. 10
compute_timestep ....................................................... 11
copyNode ............................................................. 11
dropRedundantChannels .............................................. 12
dropRedundantNodes .................................................. 13
estimateLogicle,GatingHierarchy .................................. 13
extract_cluster_pop_name_from_node ............................. 14
filterObject ......................................................... 15
fix_channel_slash ..................................................... 16
flowData ............................................................. 16
flowjo_biexp ......................................................... 17
flowjo_biexp_trans .................................................. 18
flowjo_fasinh ........................................................ 18
flowjo_fasinh_trans ................................................ 19
flowjo_log_trans ..................................................... 20
flowWorkspace-deprecated .......................................... 21
flowWorkspace.par.init .............................................. 22
flowWorkspace.par.set .............................................. 23
flow_breaks ......................................................... 23
flow_trans .......................................................... 24
GatingHierarchy-class .............................................. 25
GatingSet,flowSet,ANY-method .................................... 25
GatingSet,GatingHierarchy,character-method ................... 26
GatingSet-class ..................................................... 27
GatingSetList-class .................................................. 28
getCompensationMatrices .......................................... 30
data,GatingHierarchy-method .................................. 31
getGate,GatingHierarchy,character-method ..................... 32
getIndices,GatingHierarchy,character-method ................ 33
getNodes,GatingSet-method ....................................... 34
getParent,GatingSet,character-method .......................... 35
getPopStats,GatingHierarchy-method ............................ 36
getPopStats,GatingSet-method ................................... 37
R topics documented:

getProp .................................................. 38
getSingleCellExpression ........................................ 39
getStats .................................................. 40
getTransformations ........................................... 42
get_log_level .............................................. 43
gh_get_cluster_labels ........................................ 43
gh_pop_get_cluster_name ..................................... 44
gh_pop_get_descendants ...................................... 44
groupByChannels ............................................ 46
groupByTree ................................................ 47
gs_pop_get_cluster_name ..................................... 44
gs_pop_get_descendants ...................................... 44
gs_pop_get_full_path ....................................... 45
gs_pop_get_indices_mat ...................................... 45
gh_pop_set_xml_count ....................................... 46
groupByChannels ............................................ 46
gs_plot_diff_tree .......................................... 49
gs_plot_pop_count_cv ....................................... 49
gs_pop_get_stats_tfilter .................................... 50
isGated ................................................... 51
keyword,GatingHierarchy,character-method ................. 51
lapply,GatingSet-method .................................... 53
length,GatingSet-method .................................... 53
logicleGml2_trans .......................................... 54
logicle_trans .............................................. 54
logtGml2_trans ............................................ 55
markernames,GatingHierarchy-method ....................... 56
mkformula .................................................. 57
moveNode ................................................... 57
ncFlowSet .................................................. 58
openWorkspace ............................................. 58
isGated ................................................... 51
PData,GatingHierarchy-method ............................... 59
plot,GatingSet,missing-method ............................. 59
plotGate ................................................... 60
pop,MFI ................................................... 62
pop_add .................................................... 63
prettyAxis .................................................. 64
rbind2,GatingSetList,missing-method ....................... 64
recompute,GatingSet-method ................................ 65
rotate_gate ................................................ 66
SampleNames,GatingHierarchy-method ....................... 67
save_gs .................................................... 68
scale_gate .................................................. 69
setGate .................................................... 70
setValue,GatingHierarchy,character,character-method .......... 71
setValue,GatingHierarchy,character,logical-method ......... 72
shift_gate .................................................. 72
standardize,GatingSet ..................................... 74
subset.GatingSet .......................................... 75
swap_data_cols ............................................ 75
transform,GatingSet-method ................................ 76
transformerList ............................................ 77
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

Package: flowWorkspace
Type: Package
Version: 0.5.40
Date: 2011-03-04
License: Artistic 2.0
LazyLoad: yes
Depends: R (>= 2.16.0), Rcpp (>= 0.9.9)

Author(s)

Greg Finak, Mike Jiang

References

http://www.rglab.org/

add

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. gs_pop_set_gate 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

add(gs, gate, ...)  

### Default S3 method:
add(gs, gate, ...)  

gs_popAdd(gs, gate, validityCheck = TRUE, ...)  

gs_pop_remove(gs, node, ...)  

Rm(node, gs, ...)

Arguments

- **gs**  
  A GatingSet
- **gate**  
  A filter or a list of filters to be added to the GatingSet.
- **...**  
  some other arguments to specify how the gates are added to the gating tree.
  - names a character vector of length four, which specifies the population names resulted by adding a quadGate. The order of the names is clock-wise starting from the top left quadrant population.
  - 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
  - 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 population. It is FALSE by default.
- **validityCheck** logical whether to check the consistency of tree structure across samples. default is TRUE. Can be turned off when speed is prefered to the robustness.
- **node**  
  A character identifies the population node in a GatingHierarchy or GatingSet to remove

Value

GatingSet method returns a GatingSet object with just root node. add method returns a population 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

### 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
gs_get_pop_paths(gs[[1]]) #only contains root node

#add one gate
rg <- rectangleGate("FSC-H"=c(200,400), "SSC-H"=c(250, 400),
      filterId="rectangle")

nodeID<-gs_pop_add(gs, rg)#it is added to root node by default if parent is not specified
nodeID
gs_get_pop_paths(gs[[1]]) #the second population is named after filterId of the gate

#add a quadGate
qg <- quadGate("FL1-H"=2, "FL2-H"=4)
nodeIDs<-gs_pop_add(gs,qg,parent="rectangle")
nodeIDs #quadGate produces four population nodes

gs_get_pop_paths(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<-gs_pop_add(gs,bg,parent="rectangle")
nodeID2
gs_get_pop_paths(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
plot(gs[[1]])

#remove one node causing the removal of all the descendants
    gs_pop_remove('rectangle', gs = gs)
    gs_get_pop_paths(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

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")
```

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

**Arguments**

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

**See Also**

- `add GatingHierarchy`

**Examples**

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

```r
# 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 `gs_remove_redundant_nodes`.

Usage

`checkRedundantNodes(...)`

`gs_check_redundant_nodes(x, path = "auto", ...)`

Arguments

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

Value

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

Examples

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

clone

clone a GatingSet

Description

clone a GatingSet

Usage

`clone(x, ...)`

`gs_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

## Not run:
#G is a GatingSet
G1<-gs_clone(G)

## End(Not run)

compensate,GatingSet,ANY-method

compensate the flow data associated with the GatingSet

Description

The compensation is saved in the GatingSet and can be retrieved by gh_get_compensations.

Usage

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

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

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.
compute_timestep

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 $TIMESTEP keyword when it is non NULL

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)

gh_copy_gate(gh, node, to)

Arguments

gh GatingHierarchy
node the node to be copied
to the new parent node under which the node will be copied
dropRedundantChannels

Removes redundant channels from flow data that are not used by gates.

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 <- gs_pop_get_parent(gh, "CD4")
new.parent <- "singlets"
gh_copy_gate(gh, "CD4", new.parent)
gs_get_pop_paths(gh)

dropRedundantChannels

Arguments

... other arguments passed to gs_get_pop_paths method

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 <- gs_remove_redundant_channels(gs)
## End(Not run)
```
## dropRedundantNodes

### Description

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

### Usage

```r
dropRedundantNodes(x, toRemove)
gs_remove_redundant_nodes(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 `gs_split_by_tree`.
- `toRemove` list of the node sets to be removed. Its length must equal the length of `x`. When `x` is a list, `toRemove` is usually the outcome from `gs_check_redundant_nodes`.

### Examples

```r
## Not run:
gslist <- list(gs1, gs2, gs3, gs4, gs5)
gs_groups <- gs_split_by_tree(gslist)
toRm <- gs_check_redundant_nodes(gs_groups)
gs_remove_redundant_nodes(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

### Description

Compute logicle transformation from the flowData associated with a GatingHierarchy

### Usage

```r
## S3 method for class 'GatingHierarchy'
estimateLogicle(x, channels, ...)
```

### Examples

```r
## S3 method for class 'GatingHierarchy'
estimateLogicle(x, channels, ...)
```
extract_cluster_pop_name_from_node

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

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

extract_cluster_pop_name_from_node("cd3/flowClust_pop1", "flowClust")
#returns "pop1"
filterObject

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
filterObject(x)

## Default S3 method:
filterObject(x)

filter_to_list(x)

## S3 method for class 'rectangleGate'
filter_to_list(x)

## S3 method for class 'polygonGate'
filter_to_list(x)

## S3 method for class 'booleanFilter'
filter_to_list(x)

## S3 method for class 'ellipsoidGate'
filter_to_list(x)

## S3 method for class 'logical'
filter_to_list(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)

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 original '_' character.

Value
the toggled channel names

flowData  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 GatingHier-
archy

Usage
flowData(x)

flowData(x) <- value

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

gs_cyto_data(x, ...)

### S4 method for signature 'GatingSet'
gs_cyto_data(x, inverse.transform = FALSE)

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

### S4 replacement method for signature 'GatingSet'
gs_cyto_data(x) <- value
**Arguments**

- **x** A GatingSet
- **value** The replacement flowSet or ncdfFlowSet object
- **inverse.transform** logical flag indicating whether to inverse transform the data

**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_biexp** *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
flowjo_biexp(channelRange = 4096, maxValue = 262144, pos = 4.5,
              neg = 0, widthBasis = -10, inverse = FALSE)

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 <- flowjo_biexp()
data.raw <- c(-1, 1e3, 1e5)
data.trans <- trans(data.raw)
round(data.trans)
inv <- flowjo_biexp(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)
flowJo_biexp_trans(...)
```

**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
library(flowCore)
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

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

---

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

`flowjo_fsinh(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**

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

inverse.trans <- flowjo_fsinh()
inverse.trans(data.trans)
```

---

**Description**

Used to construct the inverse hyperbolic sine transform object.

**Usage**

`flowjo_fasinh_trans(..., n = 6, equal.space = FALSE)`

`flowJo_fasinh_trans(...)`

**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**

```r
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))
```

**flowjo_log_trans**

*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**

```r
flowjo_log_trans(decade = 4.5, offset = 1, scale = 1, n = 6,
                  equal.space = FALSE)

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

**Arguments**

- `decade`: total number of decades (i.e. log(max)-log(min)
- `offset`: offset to the original input (i.e. min value)
- `scale`: the linear scale factor
- `inverse`: whether return the inverse function

**Value**

flog(or its inverse) transform function

**Examples**

```r
trans <- flowjo_log_trans()
data.raw <- c(1,1e2,1e3)
data.trans <- trans["transform"](data.raw)
data.trans

inverse.trans <- trans["inverse"]
inversetrans(data.trans)
```
```r
# negative input
data.raw <- c(-10, 1e2, 1e3)
data.trans <- trans[["transform"]](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_log_trans(decade = 3, offset = 30)
data.trans <- trans[["transform"]](data.raw)
data.trans
inverse.trans <- trans[["inverse"]]
inverse.trans(data.trans)
```

---

**flowWorkspace**

Deprecated functions in package `flowWorkspace`.

**Description**

- `getStats` -> `gs(/gh)_pop_get_stats`
- `getProp` -> `gh_pop_get_proportion`
- `getTotal` -> `gh_pop_get_count`
- `getPopStats` -> `gs(/gh)_pop_get_stats`
- `getNodes` -> `gs_get_pop_paths`
- `getParent` -> `gs_pop_get_parent`
- `getChildren` -> `gs_pop_get_children`
- `getGate` -> `gs(/gh)_get_gate`
- `getIndices` -> `gh_pop_get_indices`
- `isGated` -> `gh_pop_is_gated`
- `isNegated` -> `gh_pop_is_negated`
- `isHidden` -> `gh_pop_is_hidden`
- `getData` -> `gs(/gh)_get_data`
- `getTransformations` -> `gh_get_transformations`
- `getCompensationMatrices` -> `gh_get_compensations`
- `plotGate` -> `autoplot`
- `setName` -> `gs(/gh)_set_node_name/gs(/gh)_set_node_visible`
- `isNcdf` -> `gs_is_h5`
- `clone` -> `gs_clone`
- `recompute` -> `gs_recompute`
- `flowData` -> `gs_cyto_data`
- `flowData<` -> `gs_cyto_data<`
- `getLoglevel` -> `get_log_level`
- `setLoglevel` -> `set_log_level`
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**

```r
library(flowCore)
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))
```

**Description**

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

**Usage**

```r
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

```r
## Not run:
require(flowWorkspaceData)
d<-system.file("extdata",package="flowWorkspaceData")
wsfile<-list.files(d,pattern="A2004Analysis.xml",full=TRUE)
library(CytoML)
ws <- open_flowjo_xml(wsfile);
G<-try(flowjo_to_gatingset(ws,path=d,name=1));
gh <- G[[1]]
gh_pop_compare_stats(gh);
gh_plot_pop_count_cv(gh)
nodes <- gs_get_pop_paths(gh)
thisNode <- nodes[4]
plotGate(gh,thisNode);
gh_pop_get_gate(gh,thisNode);
gh_pop_get_data(gh,thisNode)
## End(Not run)
```

GatingSet,flowSet,ANY-method
collectors for GatingSet

Description

construct a gatingset with empty trees (just root node)
GatingSet,GatingHierarchy,character-method

Usage

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

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

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

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

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

Arguments

- `object`: GatingSet
- `value`: string

Examples

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

GatingSet,GatingHierarchy,character-method

constructors for GatingSet

Description

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

Usage

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

```r
gh_apply_to_new_fcs(x, files, ...)
```

Arguments

- `x`: GatingHierarchy
- `y`: sample names
- `path`: character specifies the path to the flow data (FCS files)
- `...`: other arguments.
GatingSet-class

files  fcs file paths
swap_cols  for internal usage

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 replacement method for signature 'GatingSet,ANY,ANY,GatingHierarchy'
x[[i, j = "missing",
    ...]] <- value

## 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 flowjo_to_gatingset(....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 <- open_flowjo_xml(wsfile);
G<-try(flowjo_to_gatingset(ws,execute=TRUE,path=d,name=1));
gs_plot_pop_count_cv(G);
## End(Not run)
```

---

GatingSetList-class

Class "GatingSetList"

Description

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

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

Usage

GatingSetList(x, samples = NULL)

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.

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

Examples

```r
## Not run:
#load several GatingSets from disk
gs_list<-lapply(list.files("../gs_toMerge",full=T) ,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[1:4])
gslist2[c("30104.fcs")]

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

#extract the gates associated with one population
gs_pop_get_gate(gslist2,"3+")
gs_pop_get_gate(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_labKey(gslist2[3:4],4,x="<APC Cy7-A>",y="<PE Tx RD-A>",smooth=T)

#remove cerntain gates by loop through GatingSets
gs_get_pop_paths(gslist2[[1]])
```
lapply(gslist2,function(gs)gs_pop_remove("Excl",gs = gs))

#extract the stats
gs_pop_get_count_fast(gslist2)
#extract statistics by using getQAStats defined in QUALIFIER package
res<-getQAStats(gslist2[c(4,2)],isMFI=F,isSpike=F,nslaves=1)

#archive the GatingSetList
save_gslist(gslist2, path="/rglab/workspace/flowIncubator/output/gslist",overwrite=T)
gslist2 <- load_gslist(path="/rglab/workspace/flowIncubator/output/gslist")

#convert GatingSetList into one GatingSet by gslist_to_gs
gs_merged2 <- gslist_to_gs(gslist2,ncdFile=path.expand(tempfile(tmpdir="/rglab/workspace/flowIncubator/output/",fileext=".nc")))

gs_merged2

## End(Not run)

## Not run:
samleNames(gsA) # return A1, A2
samleNames(gsB) # return B1, B2
gs.list <- list(gsA, gsB)
gslist<- GatingSetList(gs.list)
sampleNames(gslist) #return A1,A2,B1,B2

#set different order when create the GatingSetList
gslist<- GatingSetList(gs.list, samples = c("A1","B1", "A2", "B2"))
sampleNames(gslist) #return A1,B1,A2,B2

## End(Not run)

---

**getCompensationMatrices**

*Retrieve the compensation matrices from a GatingHierarchy*

**Description**

Retrieve the compensation matrices from a GatingHierarchy.

**Usage**

```r
getCompensationMatrices(x)
```

## S3 method for class 'GatingHierarchy'
getCompensationMatrices(x)

gh_get_compensations(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

**Examples**

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

---

**Description**

get gate flow data from a GatingHierarchy/GatingSet/GatingSetList

**Usage**

```r
## S4 method for signature 'GatingHierarchy'
getData(obj, y, ...)  
gh_pop_get_data(obj, y = "root", inverse.transform = FALSE, ...)
```

```r
## S4 method for signature 'GatingSet'
getData(obj, y, ...)  
gs_pop_get_data(obj, y = "root", inverse.transform = FALSE, ...)
```

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

**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::
- `inverse.transform` logical flag indicating whether to inverse transform the data

**Details**

Returns a flowFrame/flowSet containing the events in the gate defined at node `y`. Subset membership can be obtained using `gh_pop_get_indices`. Population statistics can be obtained using `getPop` and `gh_pop_compare_stats`. When calling `gh_pop_get_data` on a GatingSet, the trees representing the GatingHierarchy for each sample in the GatingSet are presumed to have the same structure. To update the data, use `gs_cyto_data` method.
getGate,GatingHierarchy,character-method

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

describe

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

Usage

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

gate_pop_get_gate(obj, y)

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

gate_pop_get_gate(obj, y)

## S4 method for signature 'GatingSetList,character'
gate(obj, y)

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

gh_pop_get_data gs_get_pop_paths

Examples

## Not run: # gh is a GatingHierarchy
gh_pop_get_gate(gh, "CD3") # return the gate for the fifth node in the tree, but fetch it by name.

# G is a GatingSet
gs_pop_get_gate(G, "CD3") # return a list of gates for the fifth node in each tree

## End(Not run)

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

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

gh_pop_get_indices(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.
getNodes,GatingSet-method

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 gh_pop_get_indices but the more convenient methods gh_pop_get_proportion and gh_pop_compare_stats 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

gh_pop_compare_stats

Examples

## Not run:
#G is a gating hierarchy
#Return the indices for population 5 (topological sort)
gh_pop_get_indices(G,gs_get_pop_paths(G,tsort=TRUE)[5]);
## End(Not run)

getNodes,GatingSet-method

Get the names of all nodes from a gating hierarchy.

description

gs_get_pop_paths 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, ...)
gs_get_pop_paths(x, y = NULL, order = "regular", path = "full", 
showHidden = FALSE, ...)
gh_get_pop_paths(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

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

Examples

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

## End(Not run)
Arguments

obj A `GatingHierarchy`
y a character/numeric the name or full/(partial) gating path or node indices of the node / population.
... other arguments passed to `gs_pop_get_paths` methods
showHidden logical whether to include the hidden children nodes.

Value

gs_pop_get_parent returns a character vector, the name of the parent population. gs_pop_get_children 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

`gs_pop_get_paths`

Examples

```r
## Not run:
#G is a gatinghierarchy
#return the name of the parent of the fifth node in the hierarchy.
gs_pop_get_parent(G,gs_get_pop_paths(G[[1]][5]))
n<-gs_get_pop_paths(G,tsort=T)[4];
gs_pop_get_children(G,n);#Get the names of the child nodes of the 4th node in this gating hierarchy. gs_pop_get_children(G,4);#Get the ids of the child nodes
## End(Not run)
```

Description

Compare the stats(count/freq) between the version parsed from xml and the one recalculated/gated from R.

Usage

```r
## S4 method for signature 'GatingHierarchy'
getPopStats(x, path = "auto", ...)
gh_pop_compare_stats(x, path = "auto", ...)
gh_plot_pop_count_cv(x, path = "auto", ...)
```
**Arguments**

- `x` GatingHierarchy
- `path` see `gs_get_pop_paths`
- `...` not used

**Description**

`gs_pop_get_count_fast` 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.

**Usage**

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

**Arguments**

- `x` A GatingHierarchy or GatingSet
- `statistic` character specifies the type of population statistics to extract. (only valid when format is "wide"). Either "freq" or "count" is currently supported.
- `xml` logical indicating whether the statistics come from xml (if parsed from xml workspace) or from openCyto.
- `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
- `path` character see `gs_get_pop_paths`
- `...` Additional arguments passed to `gs_get_pop_paths`
- `x` a GatingSet or GatingSetList
- `...` additional arguments passed to `gs_pop_get_count_fast`
getProp

Get count or proportion from populations

Description

Get count or proportion from populations

Usage

getProp(x, y, xml = FALSE)

gh_pop_get_proportion(x, y, xml = FALSE)

getTotal(x, y, xml = FALSE)

gh_pop_get_count(x, y, xml = FALSE)
getSingleCellExpression

Arguments

- **x**  
  GatingHierarchy

- **y**  
  character node name or path

- **xml**  
  whether to extract xml stats or openCyto stats

getSingleCellExpression

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

getSingleCellExpression(...)

```r
gs_get_singlecell_expression(x, nodes, other.markers = NULL, 
  swap = FALSE, threshold = TRUE, marginal = TRUE, 
  mc.cores = getOption("mc.cores", 1L), ...)
```

getSingleCellExpressionByGate(...)

```r
gs_get_singlecell_expression_by_gate(...)
```

Arguments

- **...**  
  other arguments 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 the desc or name columns of the parameters of the associated flowFrames in the GatingSet). see examples.

- **ignore.case**  
  whether to ignore case when match the marker names. Default is FALSE.

- **x**  
  A GatingSet or GatingSetList object.

- **nodes**  
  character vector specifying different cell populations

- **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 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.

- **mc.cores**  
  passed to mclapply. Default is 1, which means the process runs in serial mode. When it is larger than 1, parallel mode is enabled.
getStats

Extract stats from populations(or nodes)

Description

Extract stats from populations(or nodes)

Usage

getStats(x, ...)  

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

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

gs_pop_get_stats(x, ...)

# S3 method for class 'GatingHierarchy'

Value

A list of numerici matrices

Author(s)

Mike Jiang <wjiang2@fhcrc.org>

See Also

gh_pop_get_indices gs_pop_get_count_fast

Examples

### Not run:

#G is a GatingSet
nodes <- c("4+/TNFa+", "4+/IL2+")
res <- gs_get_singlecell_expression(gs, nodes)
res[[1]]

# if it fails to match the given nodes to the markers, then try to provide the mapping between node and marker explicitly
res <- gs_get_singlecell_expression(gs, nodes, map = list("4+/TNFa+" = "TNFa", "4+/IL2+" = "IL2"))

# It can also operate on the 2d gates by setting marginal to FALSE
# The markers are no longer deduced from node names or supplied by map
# Instead, it retrieves the markers that are associated with the gates
nodes <- c("4+/TNFa+IFNg+", "4+/IL2+IL3+")
res <- gs_get_singlecell_expression(gs, nodes, marginal = FALSE)
# or simply call convenient wrapper
gs_get_singlecell_expression_by_gate(gs, nodes)

### End(Not run)
getStats(...)

gh_pop_get_stats(x, nodes = NULL, type = "count", xml = FALSE, inverse.transform = FALSE, stats.fun.arg = list(), ...)

Arguments

- **x**
  - a GatingSet or GatingHierarchy
- **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.
- **xml**
  - whether to extract xml stats or openCyto stats
- **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

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

Examples

```r
## 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 <- gs_pop_get_stats(gs) #default is "count"

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

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

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

# supply user-defined stats fun
pop.quantiles <- function(fr){
  chnls <- colnames(fr)
  res <- matrixStats::colQuantiles(exprs(fr), probs = 0.75)
  names(res) <- chnls
  res
}
gs_pop_get_stats(gs, nodes, type = pop.quantiles)

## End(Not run)
```
getTransformations

Return a list of transformations or a transformation in a GatingHierarchy

Description

Return a list of all the transformations or a transformation in a GatingHierarchy

Usage

getTransformations(x, ...)

## S3 method for class 'GatingHierarchy'
getTransformations(...)

gh_get_transformations(x, channel = NULL, inverse = FALSE,
    only.function = TRUE, ...)

Arguments

x A GatingHierarchy object
...
orther arguments equal spaced logical passed to the breaks functio to determine
whether to break at 10^n or equally spaced intervals
channel character channel name
inverse logical whether to return the inverse transformation function. Valid when
only.function is TRUE
only.function logical whether to return the function or the entire transformer object(see
scales package) that contains transform and inverse and breaks function.

Details

Returns a list of 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 trans-
formed for a sample or group of samples in a flowjo_workspace. 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 repre-
sent channel- or parameter-specific transformation functions that map from raw intensity values to
channel-space used by flowJo.

Value

lists of functions(or transform objects when only.function is FALSE), with each element of the list
representing a transformation applied to a specific channel/parameter of a sample.

Examples

## Not run:
#Assume gh is a GatingHierarchy
gh_get_transformations(gh); # return a list transformation functions
gh_get_transformations(gh, inverse = TRUE); # return a list inverse transformation functions
get_log_level

get_log_level
get/set the log level

Description

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

Usage

get_log_level()
set_log_level(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

get_log_level()
set_log_level("Population")
get_log_level()

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

---

**gh_pop_get_cluster_name**

*check if a node is clustering node*

**Description**

check if a node is clustering node

**Usage**

gh_pop_get_cluster_name(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_pop_get_descendants**

*get all the descendant nodes for the given ancestor*

**Description**

get all the descendant nodes for the given ancestor

**Usage**

gh_pop_get_descendants(gh, node, ...)

getDescendants(...)

**Arguments**

gh : GatingHierarchy

node : the node path

... : passed to getNode call
**Examples**

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

**gh_pop_get_full_path**

convert the partial gating path to the full path

**Description**

convert the partial gating path to the full path

**Usage**

`gh_pop_get_full_path(gh, path)`

**Arguments**

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

**Value**

the full gating path

**gh_pop_get_indices_mat**

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

**Description**

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

**Usage**

`gh_pop_get_indices_mat(gh, y)`

**Arguments**

- `gh` GatingHierarchy object
- `y` character string containing the boolean or of node names e.g. `"cd4|cd8"`
**gh_pop_set_xml_count**  
save the event counts parsed from xml into c++ tree structure

---

**Description**

It is for internal use by the diva parser

**Usage**

gh_pop_set_xml_count(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:
gh_pop_set_xml_count(gh, "CD3", 10000)
## End(Not run)
```

---

**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)

gs_split_by_channels(x)

**Arguments**

- x: a list of GatingSets
Examples

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

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 gs_get_pop_paths(gh, path = "auto", showHidden = F) returns the same set.

Usage

groupByTree(x)
gs_split_by_tree(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

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

gs_get_compensation_internal

extract compensation object from GatingSet

Description

extract compensation object from GatingSet

Usage

gs_get_compensation_internal(gs, sampleName)
Arguments

\[ gs \rightarrow \text{GatingSet} \]

\[ \text{sampleName} \rightarrow \text{sample name} \]

\[
gs\_get\_leaf\_nodes \quad \text{get all the leaf nodes}
\]

Description

get all the leaf nodes

Usage

\[
gs\_get\_leaf\_nodes(x, \ldots)
\]

\[
g\_get\_leaf\_nodes(x, \ldots)
\]

Arguments

\[ x \rightarrow \text{GatingHierarchy/GatingSet object} \]

\[ \ldots \rightarrow \text{arguments passed to 'gs\_get\_pop\_paths' method} \]

Value

the leaf nodes

\[
gs\_is\_h5 \quad \text{determine the flow data associated with a Gating Hierarchy is based on 'ncdfFlowSet' or 'flowSet'}
\]

Description

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

Usage

\[
gs\_is\_h5(x)
\]

\[
isNcdf(x)
\]

Arguments

\[ x \rightarrow \text{GatingHierarchy object} \]

Value

logical
gs_plot_diff_tree

visualize the tree structure difference among the GatingSets

Description

visualize the tree structure difference among the GatingSets

Usage

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

Arguments

x list of groups (each group is a list of ‘GatingSet’). It is usually the outcome from gs_split_by_tree.

path passed to getNodes

... passed to getNodes

Examples

## Not run:

gslist <- list(gs1, gs2, gs3, gs4, gs5)
gs_groups <- gs_split_by_tree(gslist)
gs_plot_diff_tree(gs_groups)

## End(Not run)

gs_plot_pop_count_cv

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


gs_plot_pop_count_cv(x, scales = list(x = list(rot = 90)), path = "auto", ...)

Arguments

x A GatingHierarchy from or a GatingSet.

scales list see barchart

path character see gs_get_pop_paths

... Additional arguments to the barplot methods.
Details

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

Value

Nothing is returned.

See Also

gs_pop_get_count_fast

Examples

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

---

**gs_pop_get_stats_tfilter**

*Extract stats from populations(or nodes) within a restricted time window*

**Description**

Extract stats from populations(or nodes) within a restricted time window

**Usage**

```r
gs_pop_get_stats_tfilter(x, ...) gh_pop_get_stats_tfilter(x, nodes = NULL, type = "count", inverse.transform = FALSE, stats.fun.arg = list(), tfilter = NULL, ...)
```

**Arguments**

- `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.
Either a list \((t_{\text{min}}, t_{\text{max}})\) specifying the minimum and maximum of a the time window filter or a GatingHierarchy, whose minimum and maximum time will be used to determine the window. For both \(x\) and the reference GatingHierarchy in \(t\)filter, the only channels that will match this filter are "Time" or "time" and the filter will be applied to each event such that only events with time value \(t\) where \(t_{\text{min}} \leq t \leq t_{\text{max}}\) will be evaluated.

---

**isGated**

The flags of gate nodes \(\text{gh\_pop\_is\_gated}\) checks if a node is already gated \(\text{gh\_pop\_is\_negated}\) checks if a node is negated. \(\text{gh\_pop\_is\_hidden}\) checks if a node is hidden.

---

**Description**

The flags of gate nodes \(\text{gh\_pop\_is\_gated}\) checks if a node is already gated \(\text{gh\_pop\_is\_negated}\) checks if a node is negated. \(\text{gh\_pop\_is\_hidden}\) checks if a node is hidden.

**Usage**

\[
isGated(obj, y) \\
g_{\text{h\_pop\_is\_gated}}(obj, y) \\
isNegated(obj, y) \\
g_{\text{h\_pop\_is\_negated}}(obj, y) \\
isHidden(obj, y) \\
g_{\text{h\_pop\_is\_hidden}}(obj, y) \\
g_{\text{h\_pop\_is\_bool\_gate}}(obj, y)
\]

**Arguments**

- **obj**: GatingHierarchy
- **y**: node/gating path
- **...**: not used

---

**keyword,GatingHierarchy,character-method**

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

---

**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)
Description

Sample names are used for names of the returned list.

Usage

```r
## 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'

Description

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

Usage

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

## 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**

```r
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))
```

---

logicle_trans

*logicle transformation.*

**Description**

Used for construct logicle transform object.

**Usage**

```r
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
logtGml2_trans

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))

logtGml2_trans

Gating-ML 2.0 Log transformation.

Description

Used to construct flog transformer object.

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 # fasinb 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**

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**

`mkformula` is a utility function used to 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 unnamed, then treated as the order of c(y,x).
- `isChar`: A 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")), func ~ var)
all.equal(mkformula(c(x = "SSC-A", y = "FSC-A")), func ~ var)
```

---

**moveNode**

`moveNode` is a utility function used to 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)
gh_pop_move(gh, node, to)
```
openWorkspace

Arguments

gh | GatingHierarchy
node | the node to be moved
to | the new parent node under which the node will be moved to

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 <- gs_pop_get_parent(gh, "CD4")
new.parent <- "singlets"
gh_pop_move(gh, "CD4", new.parent)
gs_pop_get_parent(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

openWorkspace(file, ...)

Arguments

file | xml file
... | other arguments
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

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, ...)
Arguments

- **x**: GatingSet or GatingHierarchy object
- **y**: character the node name or full/(partial) gating path or numeric representing the node index in the 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 `gs_get_pop_paths`.

...• 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 GatingHierarchy
• `arrange logical` indicating whether to arrange different populations/nodes on the same page via 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, `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 lattice;
• `gpar list` of grid parameters passed to grid.layout;
• lattice logical deprecated;
• formula formula a formula passed to `xyplot` function of 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 `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 densityplot. 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 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 `gs_get_pop_paths` 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


### 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))
# Not run:
#G is a GatingHierarchy
plotGate(G, gs_get_pop_paths(G)[5]);#plot the gate for the fifth node
```

### Description

`pop.MFI` computes and returns the median fluorescence intensity for each marker. They are typically used as the arguments passed to `gh_pop_get_stats` 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

Description

Add populations to a GatingHierarchy

Usage

```r
pop_add(gate, gh, ...)  
## S3 method for class 'filter'
pop_add(gate, gh, ...)  
## S3 method for class 'filters'
pop_add(gate, gh, names = NULL, ...)  
## S3 method for class 'quadGate'
pop_add(gate, gh, names = NULL, ...)  
## S3 method for class 'logical'
pop_add(gate, gh, parent, name, recompute,  
        cluster_method_name = NULL, ...)  
## S3 method for class 'factor'
pop_add(gate, gh, name = NULL, ...)  
## S3 method for class 'logicalFilterResult'
pop_add(gate, gh, ...)  
## S3 method for class 'multipleFilterResult'
pop_add(gate, gh, name = NULL, ...)  
gh_pop_remove(gh, node, ...)  
```

Arguments

*gate*  
a gate object that extends flowCore::filter or flowCore::filters

*gh*  
GatingHierarchy

*...*  
other arguments

*names*  
a character vector of length four, which specifies the population names resulted by adding a quadGate. The order of the names is clock-wise starting from the top left quadrant population.
**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

```r
prettyAxis(gh, channel)
```

### Arguments

- `gh` GatingHierarchy
- `channel` character channel name

### 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)
```
### Description

Merge a GatingSetList into a single GatingSet

### Usage

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

### Arguments

- **x**: GatingSetList
- **y**: missing not used.
- **...**: other arguments passed to gslist_to_gs method for ncdfFlowList

### Description

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

### Usage

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

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

### Arguments

- **x**: GatingSet
- **y**: character node name or node path. Default "root". Optional.
- **alwaysLoadData**: 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.

... other arguments leaf.bool whether to compute the leaf boolean gate, default is TRUE

Details

It is usually used immediately after add or gs_pop_set_gate calls.

---

**rotate_gate**  
*Simplied 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 gs_pop_set_gate 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
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 come from pdata of fs.

Value

A character vector of sample names

Examples

```r
## 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

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

load_gs(path)

save_gsl(gslist, path, ...)

load_gsl(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. Sometimes 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_gsl` returns a GatingSetList object
## scale_gate

Simplified geometric scaling of gates associated with nodes

### Description

Simplified geometric scaling of gates associated with nodes

### Usage

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

## 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 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

### Examples

```r
## Not run:
# G is a GatingSet
save_gs(G, path="tempFolder")
G1 <- load_gs(path="tempFolder")

# G is a GatingSet
save_gsl(gslist1, path="tempFolder")
gslist2 <- load_gsl(path="tempFolder")

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

---

**setGate**

*update the gate*

**Description**

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

**Usage**

```r
setGate(obj, y, value, ...)

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

gs_pop_set_gate(obj, y, value, negated = FALSE, ...)

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

gs_pop_set_gate(obj, y, value, ...)
```
Arguments

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

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

```r
## 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])
gs_pop_set_gate(gs[1:2], "lymph", flist)
recompute(gs[1:2], "lymph")

## End(Not run)
```

Description

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

Usage

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

gh_pop_set_name(x, y, value)

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

gs_pop_set_name(x, y, value)
```

Arguments

- **x**: GatingHierarchy
- **y**: pop name/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
gs_get_pop_paths(G[[1]])#return node names
gh_pop_set_name(G,"L","lymph")

## End(Not run)
```

**setNode, GatingHierarchy, character, logical-method**

hide/unhide a node

Description

hide/unhide a node

Usage

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

gh_pop_set_visibility(x, y, value)
gs_pop_set_visibility(x, y, value)
```

Arguments

- `x` : GatingHierarchy object
- `y` : character node name or path
- `value` : TRUE/FALSE

Examples

```r
## Not run:
gh_pop_set_visibility(gh, 4, FALSE) # hide a node
gs_pop_set_visibility(gh, 4, TRUE) # unhide a node

## End(Not run)
```

**shift_gate**

Simplified geometric translation of gates associated with nodes
shift_gate

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 gs_pop_set_gate 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)
- `...`: not used

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 flowCore::shift_gate

Examples

## 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)

standardize-GatingSet  The tools to standardize the tree structures and channel names.

Description

gs_split_by_tree(x)
gs_split_by_channels(x)
gs_check_redundant_nodes(x)
gs_remove_redundant_nodes(x, toRemove)
gs_remove_redundant_channels(gs)
gs_update_channels(gs, map, all = TRUE)
gh_pop_move(gh, node, to)
gs_pop_set_visibility(x, y, FALSE)

Details

In order to merge multiple GatingSets into single GatingSetList, the gating trees and channel names must be consistent. These functions help removing the discrepancies and standardize the GatingSets so that they are mergable.

gs_split_by_tree splits the GatingSets into groups based on the gating tree structures.
gs_split_by_channels split GatingSets into groups based on their flow channels.
gs_check_redundant_nodes returns the terminal(or leaf) nodes that makes the gating trees to be different among GatingSets and thus can be considered to remove as redundant nodes.
gs_remove_redundant_nodes removes the terminal (or leaf) nodes that are detected as redundant by gs_check_redundant_nodes.
gs_remove_redundant_channels remove the redundant channels that are not used by any gate defined in the GatingSet.
gs_update_channels modifies the channel names in place. (Usually used to standardize the channels among GatingSets due to the letter case discrepancies or typo).
gh_pop_move inserts a dummy gate to the GatingSet. It is useful trick to deal with the extra non-leaf node in some GatingSets that can not be simply removed by gs_remove_redundant_nodes.
gs_pop_set_visibility hide a node/gate in a GatingSet. It is useful to deal with the non-leaf node that causes the tree structure discrepancy.

subset.GatingSet

subset the GatingSet/GatingSetList based on 'pData'

Description
subset the GatingSet/GatingSetList based on 'pData'

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

Arguments
x GatingSet or GatingSetList
subset logical expression (within the context of pData) indicating samples to keep. see subset
... other arguments. (not used)

Value
a codeGatingSet or GatingSetList object

swap_data_cols

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

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

Usage
swap_data_cols(cols, swap_cols)
transform,GatingSet-method

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

library(flowCore)
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 asociated with the GatingSet

Description

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

Usage

## 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 underling flow data transformed.
Examples

## 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(flowCore)
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 `gs_pop_set_gate` 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'
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 `polytopeGate` 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.
transform_gate

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 a 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**

```r
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

**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 <- gs_update_channels(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)
gh_pop_set_indices(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 <- gh_pop_get_stats(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
gInd <- sample.int(total.cd3, size = total.cd3 * 0.3) # randomly select 30%
# convert it to logicle index
gInd.logical <- rep(FALSE, total.cd3)
gInd.logical[gInd] <- TRUE
# replace the original index stored at GatingHierarchy
gh_pop_set_indices(gh, "CD3+", gInd.logical)
# check the updated pop counts
gh_pop_get_stats(gs[[1]], nodes = c("CD3+", "CD4", "CD8")) # note that CD4, CD8 are not updated
# update all the descendants of CD3+
nodes <- gh_pop_get_descendants(gh, "CD3+")
for (node in nodes) suppressMessages(recompute(gh, node))
gh_pop_get_stats(gs[[1]], nodes = c("CD3+", "CD4", "CD8")) # now all are update to date
```
Index

[,GatingSet,ANY-method
  (GatingSet-class), 27
[,GatingSetList,ANY-method
  (GatingSet-class), 27
[[,GatingSet,character-method
  (GatingSet-class), 27
[[,GatingSet,logical-method
  (GatingSet-class), 27
[[,GatingSet,numeric-method
  (GatingSet-class), 27
[[-,GatingSet,ANY,ANY,GatingHierarchy-method
  (GatingSet-class), 27
add, 4, 8, 66, 71
asinh_Gml2, 7
asinhGml2_trans, 6

barchart, 49
booleanFilter (booleanFilter-class), 8
booleanFilter-class, 8
char2booleanFilter
  (booleanFilter-class), 8
checkRedundantNodes, 9
close, 9
close,GatingSet-method (clone), 9
close-methods (clone), 9
close.ncdfFlowSet, 10
colnames,GatingHierarchy-method
  (markernames,GatingHierarchy-method), 56
colnames,GatingSet-method
  (markernames,GatingHierarchy-method), 56
colnames<-,GatingHierarchy-method
  (markernames,GatingHierarchy-method), 56
colnames<-,GatingSet-method
  (markernames,GatingHierarchy-method), 56
compensate,GatingSet,ANY-method, 10
compensate,GatingSetList,ANY-method
  (compensate,GatingSet,ANY-method), 10
compute_timestep, 11
copyNode, 11
dropRedundantChannels, 12
dropRedundantNodes, 13
ellipsoidGate, 67, 69, 73, 79
estimateLogicle.GatingHierarchy, 13
expressionFilter, 8
extract_cluster_pop_name_from_node, 14
filter_to_list (filterObject), 15
filterObject, 15, 71
fix_channel_slash, 16
flow_breaks, 23
flow_trans, 24
flowCore::rotate_gate, 67
flowCore::scale_gate, 70
flowCore::shift_gate, 74
flowCore::transform_gate, 79
flowData, 16
flowData,GatingSet-method (flowData), 16
flowData<-(flowData), 16
flowJo.fasinh (flowjo_fasinh), 18
flowJo.flog (flowjo_log_trans), 20
flowJo.fsinh (flowjo_fasinh), 18
flowjo_biexp, 17
flowjo_biexp_trans
  (flowjo_fasinh_trans), 18
flowjo_fasinh, 18
flowJo_fasinh, 18
flowjo_fasinh_trans, 19
flowjo_fasinh_trans, 19
flowjo_fasinh, 18
flowjo_fasinh, 18
flowjo_fasinh_trans, 19
flowJo_fasinh, 18
flowjo_fasinh_trans, 19
flowJo_fasinh, 18
flowjo_biexp, 17
flowjo_biexp_trans
  (flowjo_fasinh_trans), 18
flowjo_biexp_trans, 18
flowjo_biexp_trans, 18
flowjo_fasinh, 18
flowJo_fasinh, 18
flowjo_fasinh_trans, 19
flowjo_fasinh_trans, 19
flowjo_fasinh, 18
flowjo_fasinh, 18
flowjo_biexp, 17
flowjo_biexp_trans
  (flowjo_fasinh_trans), 18
flowjo_biexp_trans, 18
flowjo_biexp_trans, 18
flowJoTrans, 18, 76
flowJoTrans (flowjo_biexp), 17
flowWorkspace (flowWorkspace-package), 4
flowWorkspace-deprecated, 21
flowWorkspace-package, 4
flowWorkspace.par.get
  (flowWorkspace.par.set), 23
flowWorkspace.par.init, 22
82
flowWorkspace.par.set, 23
GatingHierarchy, 8, 28, 29, 61, 78
GatingHierarchy-class, 25
GatingSet, 25, 29, 61, 78
GatingSet(GatingSet-class), 27
GatingSet, flowSet, ANY-method, 25
GatingSet, GatingHierarchy, character-method, 26
GatingSet-class, 27
GatingSetList, 74
GatingSetList(GatingSetList-class), 28
get_leaf_nodes (gs_get_leaf_nodes), 48
get_log_level, 43
getChildren, GatingSet, character-method
(getParent, GatingSet, character-method), 35
getCompensationMatrices, 30
getData, GatingHierarchy-method, 31
dataGet, GatingSet-method
(getData, GatingHierarchy-method), 31
dataGet, GatingSetList-method
(getData, GatingHierarchy-method), 31
getDescendants
(gh_pop_get_descendants), 44
getGate, GatingHierarchy, character-method, 32
getGate, GatingSet, character-method
(getGate, GatingHierarchy, character-method), 32
getGate, GatingSetList, character-method
(getGate, GatingHierarchy, character-method), 32
getIndices, GatingHierarchy, character-method
(getGate, GatingHierarchy, character-method), 33
getNodes, GatingSet-method, 34
getParent, GatingSet, character-method, 35
getPopStats, GatingHierarchy-method, 36
getPopStats, GatingSet-method, 37
getProp, 38
getSingleCellExpression, 39
getSingleCellExpressionByGate
(getSingleCellExpression), 39
getStats, 40
getTotal (getProp), 38
getTransformations, 42
gh_apply_to_new_fcs
(GatingSet, GatingHierarchy, character-method), 26
gh_copy_gate (copyNode), 11
gh_get_cluster_labels, 43
gh_get_compensations, 10
gh_get_compensations
(getCompensationMatrices), 30
gh_get_leaf_nodes (gs_get_leaf_nodes), 48
gh_get_pop_paths
(getNodes, GatingSet-method), 34
gh_get_transformations, 76
gh_get_transformations
(getTransformations), 42
gh_plot_pop_count_cv
(getPopStats, GatingHierarchy-method), 36
gh_pop_compare_stats, 32, 34
gh_pop_compare_stats
(getPopStats, GatingHierarchy-method), 36
gh_pop_get_cluster_name, 44
gh_pop_get_count (getProp), 38
gh_pop_get_data, 33
gh_pop_get_data
(getData, GatingHierarchy-method), 31
gh_pop_get_descendants, 44
gh_pop_get_full_path, 45
gh_pop_get_gate
(getGate, GatingHierarchy, character-method), 32
gh_pop_get_indices, 32, 40
gh_pop_get_indices
(getIndices, GatingHierarchy, character-method), 33
gh_pop_get_indices_mat, 45
gh_pop_get_parent
(getParent, GatingSet, character-method), 35
gh_pop_get_proportion (getProp), 38
gh_pop_get_stats (getStats), 40
gh_pop_get_stats_tfilter
(gs_pop_get_stats_tfilter), 50
gh_pop_is_bool_gate (isGated), 51
gh_pop_is_gated (isGated), 51
gh_pop_is_hidden (isGated), 51
gh_pop_is_negated (isGated), 51
gh_pop_move, 75
method, move (moveNode), 57
gh_pop_remove (pop_add), 63
gh_pop_set_gate (setGate), 70
gh_pop_set_indices
(updateIndices, GatingHierarchy, character, logical-method), 80
gh_pop_set_name
(setNode, GatingHierarchy, character, character-method), 71
gh_pop_set_visibility
(setNode, GatingHierarchy, character, logical-method), 72
gs_pop_set_xml_count, 46
group.layout, 61
groupByChannels, 46
groupByTree, 47
gs_check_redundant_nodes, 13, 74
gs_check_redundant_nodes
(checkRedundantNodes), 9
gs_clone (clone), 9
gs_cyto_data, 32
gs_cyto_data (flowData), 16
gs_cyto_data, GatingSet-method
(flowData), 16
gs_cyto_data<-, GatingSet-method
(flowData), 16
gs_get_compensation_internal, 47
gs_get_leaf_nodes, 48
gs_get_pop_paths, 9, 33, 36–38, 41, 49, 61, 62
gs_get_pop_paths
(getNodes, GatingSet-method), 34
gs_get_singlecell_expression
(getSingleCellExpression), 39
gs_get_singlecell_expression_by_gate
(getSingleCellExpression), 39
gs_is_h5, 48
gs_plot_diff_tree, 49
gs_plot_pop_count_cv, 49
gs_pop_add (add), 4
gs_pop_get_children
(getParent, GatingSet, character-method), 35
gs_pop_get_count_fast, 40, 50
gs_pop_get_count_fast
(getPopStats, GatingSet-method), 37
gs_pop_get_count_with_meta
(getPopStats, GatingSet-method), 37
gs_pop_get_data
(getData, GatingHierarchy-method), 31
gs_pop_get_gate
(getGate, GatingHierarchy, character-method), 32
ingo_pop_get_method
(getParent, GatingSet, character-method), 35
gs_pop_get_stats
(getStats), 40
gs_pop_remove (add), 4
gs_pop_set_gate, 66, 73, 78
gs_pop_set_gate (setGate), 70
gs_pop_set_name
(setNode, GatingHierarchy, character, character-method), 71
gs_pop_set_visibility, 75
gs_pop_set_visibility
(setNode, GatingHierarchy, character, logical-method), 72
gs_remove_redundant_channels, 75
gs_remove_redundant_channels
(dropRedundantChannels), 12
gs_remove_redundant_nodes, 9, 75
gs_remove_redundant_nodes
(dropRedundantNodes), 13
gs_split_by_channels, 74
gs_split_by_channels (groupByChannels), 46
gs_split_by_tree, 9, 13, 49, 74
gs_split_by_tree (groupByTree), 47
gs_update_channels, 75
gs_update_channels (updateChannels), 80
gslist_to_gs
(rbinder2, GatingSetList, missing-method), 65
identifier, GatingSet-method
(GatingSet, flowSet, ANY-method), 25
identifier, GatingSetList-method
(GatingSet, flowSet, ANY-method), 25
identifier<-, GatingSet, character-method
(GatingSet, flowSet, ANY-method), 25
identifier<-, GatingSetList, character-method
(GatingSet, flowSet, ANY-method), 25
isGated, 51
isHidden (isGated), 51
isNcdf (gs_is_h5), 48
isNegated (isGated), 51
keyword
(keyword, GatingHierarchy, character-method),
INDEX

Rm (add), 4
rotate_gate, 66, 66, 79
sampleNames
  (sampleNames, GatingHierarchy-method), 67
  sampleNames, GatingHierarchy-method, 67
sampleNames <- (sampleNames, GatingHierarchy-method), 67
save_gs, 68
save_gslist (save_gs), 68
scale_gate, 69, 79
set_log_level (get_log_level), 43
setGate, 70
setGate, GatingHierarchy, character, filter-method
  (setGate), 70
setGate, GatingSet, character, ANY-method
  (setGate), 70
setNode
  (setNode, GatingHierarchy, character, character-method), 71
  setNode, GatingHierarchy, character, character-method, 71
  setNode, GatingHierarchy, character, logical-method, 72
setNode, GatingSet, character, ANY-method
  (setNode, GatingHierarchy, character, character-method), 71
shift_gate, 72, 73, 79
show, booleanFilter-method
  (booleanFilter-class), 8
  show, GatingHierarchy-method
  (GatingHierarchy-class), 25
  show, GatingSet-method
  (length, GatingSet-method), 53
standardize-GatingSet, 74
subset, 75
subset.GatingSet, 75
swap_data_cols, 75
transform, GatingSet-method, 76
transform, GatingSetList-method
  (transform, GatingSet-method),
transformerList, 77
updateChannels, 80
updateIndices, GatingHierarchy, character, logical-method, 80
xyplot, 62