

# Package ‘flowWorkspace’

October 16, 2019

**Type** Package

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

**Version** 3.32.0

**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

**Collate** 'AllGenerics.R' 'AllClasses.R' 'GatingHierarchy\_Methods.R' 'GatingSet\_Methods.R' 'GatingSetList\_Methods.R' 'RcppExports.R' 'filterObject\_Methods.R' 'add\_Methods.R' 'copyNode.R' 'deprecated.R' 'flow\_trans.R' 'getDescendants.R' 'getSingleCellExpression.R' 'getStats.R' 'identifier.R' 'merge\_GatingSet.R' 'moveNode.R' 'setGate\_Methods.R' 'updateIndices.R' 'utils.R' 'zzz.R'

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

**biocViews** ImmunoOncology, FlowCytometry, DataImport, Preprocessing, DataRepresentation

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

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

**VignetteBuilder** knitr

**SystemRequirements** GNU make, C++11

**Encoding** UTF-8

**RoxygenNote** 6.1.1**git\_url** <https://git.bioconductor.org/packages/flowWorkspace>**git\_branch** RELEASE\_3\_9**git\_last\_commit** 89c71ca**git\_last\_commit\_date** 2019-05-02**Date/Publication** 2019-10-15**R topics documented:**

flowWorkspace-package	4
add,GatingSet,list-method	4
asinhGml2_trans	8
asinh_Gml2	8
booleanFilter-class	9
checkRedundantNodes	10
clone	11
compensate,GatingSet,ANY-method	11
compute_timestep	12
copyNode	13
dropRedundantChannels	13
dropRedundantNodes	14
estimateLogicle.GatingHierarchy	14
extract_cluster_pop_name_from_node	15
filterObject,rectangleGate-method	16
fix_channel_slash	16
flowData,GatingSet-method	17
flowJo.fasinh	18
flowJo.flog	18
flowJoTrans	19
flowJo_biexp_trans	20
flowJo_fasinh_trans	21
flowWorkspace.par.init	21
flowWorkspace.par.set	22
flow_breaks	22
flow_trans	23
GatingHierarchy-class	24
GatingSet,flowSet,ANY-method	24
GatingSet-class	25
GatingSetList-class	27
getCompensationMatrices	29
getCompensationObj	30
getData,GatingHierarchy,missing-method	30
getDescendants	31
getFullNodePath	32
getGate,GatingHierarchy,character-method	32
getIndiceMat	33
getIndices,GatingHierarchy,character-method	34
getIndices,GatingSet,name-method	35
getLoglevel	35
getMergedStats	36

getNode, GatingSet-method	36
getParent, GatingSet, character-method	37
getProp, GatingHierarchy, character-method	38
getSingleCellExpression, GatingSetList, character-method	40
getStats	41
getTransformations	43
get_leaf_nodes	44
gh_check_cluster_node	44
gh_get_cluster_labels	45
groupByChannels	45
groupByTree	46
insertGate	46
isGated	47
isNcdf	48
keyword, GatingHierarchy, character-method	48
lapply, GatingSet-method	49
length, GatingSet-method	50
logicGml2_trans	50
logic_trans	51
logtGml2_trans	52
markernames, GatingHierarchy-method	52
mkformula	54
moveNode	54
ncFlowSet	55
openWorkspace	55
pData, GatingHierarchy-method	56
plot, GatingSet, missing-method	56
plotGate	57
plotPopCV, GatingHierarchy-method	59
plot_diff_tree	60
pop.MFI	61
prettyAxis	61
recompute, GatingSet-method	62
rotate_gate	63
sampleNames, GatingHierarchy-method	64
save_gs	65
scale_gate	66
set.count.xml	67
setGate, GatingHierarchy, character, filter-method	67
setNode, GatingHierarchy, character, character-method	68
shift_gate	69
standardize-GatingSet	71
subset.GatingSet	71
swap_data_cols	72
transform, GatingSet-method	73
transformerList	74
transform_gate	74
updateChannels	76
updateIndices, GatingHierarchy, character, logical-method	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,GatingSet,list-method

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

---

### Description

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

**Usage**

```
## S4 method for signature 'GatingSet,list'  
add(wf, action, ...)  
  
## S4 method for signature 'GatingSetList,list'  
add(wf, action, ...)  
  
## S4 method for signature 'GatingSet,filtersList'  
add(wf, action, ...)  
  
## S4 method for signature 'GatingSet,filterList'  
add(wf, action, validityCheck = TRUE,  
    ...)  
  
## S4 method for signature 'GatingSetList,filterList'  
add(wf, action, ...)  
  
## S4 method for signature 'GatingSetList,filtersList'  
add(wf, action, ...)  
  
## S4 method for signature 'GatingSet,filter'  
add(wf, action, ...)  
  
## S4 method for signature 'GatingSet,filters'  
add(wf, action, ...)  
  
## S4 method for signature 'GatingSetList,filter'  
add(wf, action, ...)  
  
## S4 method for signature 'GatingSetList,filters'  
add(wf, action, ...)  
  
## S4 method for signature 'GatingHierarchy,filter'  
add(wf, action, ...)  
  
## S4 method for signature 'GatingHierarchy,filters'  
add(wf, action, names = NULL, ...)  
  
## S4 method for signature 'GatingHierarchy,quadGate'  
add(wf, action, names = NULL, ...)  
  
## S4 method for signature 'GatingHierarchy,logical'  
add(wf, action, parent, name,  
    recompute, cluster_method_name = NULL, ...)  
  
## S4 method for signature 'GatingHierarchy,factor'  
add(wf, action, name = NULL, ...)  
  
## S4 method for signature 'GatingHierarchy,logicalFilterResult'  
add(wf, action, ...)  
  
## S4 method for signature 'GatingHierarchy,multipleFilterResult'
```

```

add(wf, action,
    name = NULL, ...)

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

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

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

```

### Arguments

wf	A GatingHierrarchy or GatingSet
action	A filter or a list of filters to be added to the GatingHierarchy or GatingSet.
...	some other arguments to specify how the gates are added to the gating tree. <ul style="list-style-type: none"> <li>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.</li> </ul>
validityCheck	logical whether to check the consistency of tree structure across samples. default is TRUE. Can be turned off when speed is preferred to the robustness.
names	a character vector of length four, which specifies the population names resulted by adding a quadGate.
parent	a character scalar to specify the parent node name where the new gate to be added to, by default it is NULL, which indicates the root node
name	a character scalar to specify the node name of population that is generated by the gate to be added.
recompute	a logical flag The order of the names is clock-wise starting from the top left quadrant population.
cluster_method_name	when adding the logical vectors as the gates, the name of the cluster method can be used to tag the populations as the extra meta information associated with the gates.
symbol	A character identifies the population node in a GatingHierrarchy or GatingSet to remove
envir	A GatingHierrarchy or GatingSet
subSymbol	Not used.

### Value

GatingSet method returns a GatingSet object with just root node. add method returns a 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
  getNodes(gs[[1]]) #only contains root node

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

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

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

#add a boolean Gate
  bg<-booleanFilter(`CD15 FITC-CD45 PE+`|`CD15 FITC+CD45 PE-`)
  bg
  nodeID2<-add(gs,bg,parent="rectangle")
  nodeID2
  getNodes(gs[[1]])
#do the actual gating
  recompute(gs)

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

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

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

## End(Not run)

```

---

asinhtGml2_trans	<i>Inverse hyperbolic sine transformation.</i>
------------------	--

---

**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 # fasin 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	<i>inverse hyperbolic sine transform function generator (GatingML 2.0 version)</i>
------------	--

---

**Description**

hyperbolic sine/inverse hyperbolic sine transform function constructor. It is simply a special form of fLowJo.fasin 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**

```
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**

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

char2booleanFilter(expr, ..., filterId = "defaultBooleanFilter")

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

**Arguments**

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

**See Also**

[add GatingHierarchy](#)

**Examples**

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

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

---

checkRedundantNodes	<i>try to determine the redundant terminal(or leaf) nodes that can be removed</i>
---------------------	---

---

**Description**

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

**Usage**

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

**Arguments**

x	GatingSet or list of groups(each group is a list of 'GatingSet'). When it is a list, it is usually the outcome from <a href="#">groupByTree</a> .
path	argumented passed to <a href="#">getNode</a> s. The default value is "auto".
...	other arguments passed to <a href="#">getNode</a> s.

**Value**

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

**Examples**

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

## End(Not run)
```

---

clone	<i>clone a GatingSet</i>
-------	--------------------------

---

**Description**

clone a GatingSet

**Usage**

```
clone(x, ...)
```

**Arguments**

x	A GatingSet
...	ncdfFile = NULL: see <a href="#">clone.ncdfFlowSet</a>

**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<-clone(G)

## End(Not run)
```

---

compensate,GatingSet,ANY-method	<i>compensate the flow data associated with the GatingSet</i>
---------------------------------	---

---

**Description**

The compensation is saved in the GatingSet and can be retrieved by [getCompensationMatrices](#).

**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.

**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	<i>compute time step from fcs keyword</i>
------------------	---

---

**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

---

copyNode	<i>Copy a node along with all of its descendant nodes to the given ancestor</i>
----------	---

---

**Description**

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

**Usage**

```
copyNode(gh, node, to)
```

**Arguments**

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

**Examples**

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

---

dropRedundantChannels	<i>Remove the channels from flow data that are not used by gates</i>
-----------------------	--

---

**Description**

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

**Usage**

```
dropRedundantChannels(gs, ...)
```

**Arguments**

gs	a GatingSet
...	other arguments passed to getNodes method

**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**

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

## End(Not run)
```

---

dropRedundantNodes	<i>Remove the terminal leaf nodes that make the gating trees to be different from one another.</i>
--------------------	--

---

**Description**

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

**Usage**

```
dropRedundantNodes(x, toRemove)
```

**Arguments**

x	GatingSet or list of groups(each group is a list of 'GatingSet'). When it is a list, it is usually the outcome from <a href="#">groupByTree</a> .
toRemove	list of the node sets to be removed. its length must equals to the length of 'x'. When x is a list, toRemove is usually the outcome from <a href="#">checkRedundantNodes</a> .

**Examples**

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

#Now they can be merged into a single GatingSetList.
#Note that the original gs objects are all modified in place.
GatingSetList(gslist)

## End(Not run)
```

---

estimateLogicle.GatingHierarchy	<i>Compute logicle transformation from the flowData associated with a GatingHierarchy</i>
---------------------------------	---

---

**Description**

See details in `?flowCore::estimateLogicle`

**Usage**

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

**Arguments**

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

**Value**

transformerList object

**Examples**

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

## End(Not run)
```

---

```
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,rectangleGate-method

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

---

### Description

convert flowCore filter to a list

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

### Usage

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

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

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

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

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

### Arguments

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

### Value

a list

---

fix\_channel\_slash      *toggle the channel names between '/' and '\_' character*

---

### Description

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

### Usage

```
fix_channel_slash(chnl, 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 the original '_' character.

**Value**

the toggled channel names

---

flowData,GatingSet-method

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

---

**Description**

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

**Usage**

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

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

**Arguments**

x	A GatingSet
value	The replacement flowSet or ncdfFlowSet object

**Details**

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

**Value**

the object with the new flowSet in place.

---

flowJo.fasinh                    *inverse hyperbolic sine transform function*

---

### Description

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

### Usage

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

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

### Arguments

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

### Value

fasinh/fsinh transform function

### Examples

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

```
inverse.trans <- flowJo.fsinh()
inverse.trans(data.trans)
```

---

flowJo.flog                    *flog transform function*

---

### Description

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

### Usage

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

**Arguments**

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

**Value**

flog(or its inverse) transform function

**Examples**

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

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

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

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

---

flowJoTrans

*construct the flowJo-type biexponential transformation function*


---

**Description**

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

**Usage**

```
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**

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

---

flowJo\_biexp\_trans     *flowJo biexponential transformation.*

---

**Description**

Used for constructing biexponential transformation object.

**Usage**

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

**Arguments**

...	parameters passed to <a href="#">flowJoTrans</a>
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**

```
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\_trans     *flowJo inverse hyperbolic sine transformation.*

---

### Description

Used to construct the inverse hyperbolic sine transform object.

### Usage

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

### Arguments

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

### Value

fasinh transformation object

### Examples

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

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

---

flowWorkspace.par.init

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

---

### Description

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

### Usage

```
flowWorkspace.par.init()
```

---

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

---

### Description

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

### Usage

```
flowWorkspace.par.set(name, value)
```

```
flowWorkspace.par.get(name = NULL)
```

### Arguments

<code>name</code>	The name of a parameter category to get or set.
<code>value</code>	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)
```

**Arguments**

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

**Value**

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

**Examples**

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

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

---

flow_trans	<i>helper function to generate a trans objects Used by other specific trans constructor</i>
------------	---

---

**Description**

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

**Usage**

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

**Arguments**

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

---

GatingHierarchy-class *Class GatingHierarchy*

---

### Description

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

### Details

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

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

### See Also

[GatingSet](#)

### Examples

```
## Not run:
require(flowWorkspaceData)
d<-system.file("extdata",package="flowWorkspaceData")
wsfile<-list.files(d,pattern="A2004Analysis.xml",full=TRUE)
library(CytoML)
ws <- openWorkspace(wsfile);
G<-try(parseWorkspace(ws,path=d,name=1));
gh <- G[[1]]
getPopStats(gh);
plotPopCV(gh)
nodes <- getNodes(gh)
thisNode <- nodes[4]
plotGate(gh,thisNode);
getGate(gh,thisNode);
getData(gh,thisNode)

## End(Not run)
```

---

GatingSet, flowSet, ANY-method  
*constructors for GatingSet*

---

### Description

construct a gatingset with empty trees (just root node)

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



**Usage**

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

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

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

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

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

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

**Arguments**

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

**Examples**

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

## End(Not run)
```

---

GatingSet-class	<i>Class "GatingSet"</i>
-----------------	--------------------------

---

**Description**

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

[ subsets a GatingSet or GatingSetList using the familiar bracket notation

[[ extract a GatingHierarchy object from a GatingSet or GatingSetList

**Usage**

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

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

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

**Arguments**

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

**Details**

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

**Slots**

`FCSPath`: deprecated

`data`: Object of class "flowSet". flow data associated with this GatingSet

`flag`: Object of class "logical". A flag indicating whether the gates, transformations, and compensation matrices have been applied to data, or simply imported.

`axis`: Object of class "list". stores the axis information used for `plotGate`.

`pointer`: Object of class "externalptr". points to the gating hierarchy stored in C data structure.

`guid`: Object of class "character". the unique identifier for GatingSet object.

`transformation`: Object of class "list". a list of transformation objects used by GatingSet.

`compensation`: Object of class "ANY". compensation objects.

**See Also**

[GatingHierarchy](#)

**Examples**

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

## End(Not run)
```

---

GatingSetList-class    *Class "GatingSetList"*

---

**Description**

A list of 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)
```

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

**Arguments**

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

**Details**

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

**See Also**

[GatingSet](#) [GatingHierarchy](#)

**Examples**

```

## 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
gslist2
class(gslist2)
sampleNames(gslist2)

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

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

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

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

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

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

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

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

#extract the stats
getPopStats(gslist2)

```

```

#extract statistics by using getQAStats defined in QUALIFIER package
res<-getQAStats(gslis2[c(4,2)],isMFI=F,isSpike=F,nslaves=1)

#archive the GatingSetList
save_gslis2(gslis2, path = "~/rglab/workspace/flowIncubator/output/gslis2",overwrite=T)
gslis2 <- load_gslis2(path = "~/rglab/workspace/flowIncubator/output/gslis2")

#convert GatingSetList into one GatingSet by rbind2
gs_merged2 <- rbind2(gslis2,ncdfFile=path.expand(tempfile(tmpdir="~/rglab/workspace/flowIncubator/output/gs_merged2"))
gs_merged2

## End(Not run)

## Not run:
samleNames(gsA) # return A1, A2
samleNames(gsB) # return B1, B2
gs.list <- list(gsA, gsB)
gslis2<- GatingSetList(gs.list)
samleNames(gslis2) #return A1,A2,B1,B2

#set different order when create the GatingSetList
gslis2<- GatingSetList(gs.list, samples = c("A1","B1", "A2", "B2"))
samleNames(gslis2) #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

```
getCompensationMatrices(x)

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

## Arguments

x                    A GatingHierarchy object.

## Details

Return all the compensation matrices in a GatingHierarchy.

## Value

A list of matrix representing the spillover matrix in GatingHierarchy

**Examples**

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

## End(Not run)
```

---

getCompensationObj	<i>extract compensation object from GatingSet</i>
--------------------	---

---

**Description**

extract compensation object from GatingSet

**Usage**

```
getCompensationObj(gs, sampleName)
```

**Arguments**

gs	GatingSet
sampleName	sample name

---

getData,GatingHierarchy,missing-method	<i>get gated flow data from a GatingHierarchy/GatingSet/GatingSetList</i>
--	---

---

**Description**

get gated flow data from a GatingHierarchy/GatingSet/GatingSetList

**Usage**

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

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

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

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

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

**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 ncdFlow::[[

**Details**

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

**Value**

A flowFrame object if obj is a GatingHierarchy. A flowSet or ncdFlowSet if a GatingSet. A ncdFlowList if a GatingSetList.

**See Also**

[flowData](#) [getIndices](#) [getPopStats](#)

**Examples**

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

#gh is a GatingHierarchy
getData(gh)

## End(Not run)
```

---

getDescendants	<i>get all the descendant nodes for the given ancestor</i>
----------------	--

---

**Description**

get all the descendant nodes for the given ancestor

**Usage**

```
getDescendants(gh, node, ...)
```

**Arguments**

gh	GatingHierarchy
node	the node path
...	passed to getNode call

**Examples**

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

---

getFullNodePath	<i>convert the partial gating path to the full path</i>
-----------------	---

---

**Description**

convert the partial gating path to the full path

**Usage**

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

**Arguments**

gh	GatingHierarchy object
path	the partial gating path

**Value**

the full gating path

---

getGate,GatingHierarchy,character-method	<i>Return the flowCore gate definition associated with a node in a GatingHierarchy/GatingSet.</i>
--	---

---

**Description**

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

**Usage**

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

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

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



**Arguments**

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

**Value**

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

**See Also**

[getData](#) [getNode](#)

**Examples**

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

## End(Not run)
```

---

getIndiceMat

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

---

**Description**

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

**Usage**

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

**Arguments**

gh                    GatingHierarchy object  
 y                    character node name

---

*getIndices,GatingHierarchy,character-method*

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

---

### Description

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

### Usage

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

### Arguments

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

### Details

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

### Value

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

### Note

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

### See Also

[getPopStats](#)

### Examples

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

## End(Not run)
```

---

```
getIndices,GatingSet,name-method
```

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

---

### Description

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

### Usage

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

### Arguments

obj	GatingSet
y	a quoted expression.

### Examples

```
## Not run:

getIndices(gs,quote(`4+/TNFa+|4+/IL2+`))

## End(Not run)
```

---

```
getLogLevel
```

*get/set the log level*

---

### Description

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

### Usage

```
getLogLevel()

setLogLevel(level = "none")
```

### Arguments

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

**Value**

a character that represents the internal log level

**Examples**

```
getLogLevel()
setLogLevel("Population")
getLogLevel()
```

---

getMergedStats

*Get Cell Population Statistics and Sample Metadata*

---

**Description**

Get Cell Population Statistics and Sample Metadata

**Usage**

```
getMergedStats(object, ...)
```

**Arguments**

object	a GatingSet or GatingSetList
...	additional arguments passed to getPopStats

**Value**

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

**Examples**

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

## End(Not run)
```

---

getNodes,GatingSet-method

*Get the names of all nodes from a gating hierarchy.*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

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

**Examples**

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

## End(Not run)
```

---

getParent,GatingSet,character-method

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

---

**Description**

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

**Usage**

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

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

**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 <a href="#">getNode</a> s methods
showHidden	logical whether to include the hidden children nodes.

**Value**

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

**See Also**

[getNode](#)s

**Examples**

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

## End(Not run)
```

---

getProp,GatingHierarchy,character-method

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

---

**Description**

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

**Usage**

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

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

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

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

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

### Arguments

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

### Details

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

### Value

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

### See Also

[getNode](#)s

### Examples

```
## Not run:
#gh is a GatingHierarchy
getPopStats(gh);
```

```

#proportion for the fifth population
getProp(gh,getNodes(gh)[5])
getTotal(gh,getNodes(gh,tsort=T)[5])

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

## End(Not run)

```

---

```
getSingleCellExpression,GatingSetList,character-method
```

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

---

## Description

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

## Usage

```

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

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

getSingleCellExpressionByGate(...)

```

## Arguments

x	A GatingSet or GatingSetList object .
nodes	character vector specifying different cell populations
...	other arguments

other.markers character vector specifying the extra markers/channels to be returned besides the ones derived from "nodes" and "map" argument.It is only valid when threshold is set to FALSE.

swap logical indicates whether channels and markers of flow data are swapped.

threshold logical indicates whether to threshold the flow data by setting intensity value to zero when it is below the gate threshold.

marginal logical indicates whether to the gate is treaded as 1d marginal gate. Default is TRUE, which means markers are determined either by node name or by 'map' argument explained below. When FALSE, the markers are determined by the gate dimensions. and node name and 'map' argument are ignored.

map a named list providing the mapping between node names (as specified in the gating hierarchy of the gating set) and channel names (as specified in either



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.

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.

### Value

A list of numeric matrices

### Author(s)

Mike Jiang <wjiang2@fhcrc.org>

### See Also

[getIndices](#) [getPopStats](#)

### Examples

```
## Not run:
#G is a GatingSet
nodes <- c("4+/TNFa+", "4+/IL2+")
res <- getSingleCellExpression(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 ex
res <- getSingleCellExpression(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 <- getSingleCellExpression(gs, nodes, marginal = FALSE)
#or simply call convenient wrapper
getSingleCellExpressionByGate(gs, nodes)

## End(Not run)
```

---

getStats

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

---

### Description

It calculates the MFI for each marker.

**Usage**

```

getStats(x, ...)

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

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

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

```

**Arguments**

<code>x</code>	a <code>GatingSet</code> or <code>GatingHierarchy</code>
<code>...</code>	arguments passed to <code>getNode</code> method.
<code>nodes</code>	the character vector specifies the populations of interest. default is all available nodes
<code>type</code>	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 <code>flowFrame</code> object through 'fr' argument and return the stats as a named vector.
<code>inverse.transform</code>	logical flag . Whether inverse transform the data before computing the stats.
<code>stats.fun.arg</code>	a list of arguments passed to 'type' when 'type' is a function.

**Value**

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

**Examples**

```

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

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

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

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

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

# supply user-defined stats fun
pop.quantiles <- function(fr){

```

```

    chnls <- colnames(fr)
    res <- matrixStats::colQuantiles(exprs(fr), probs = 0.75)
    names(res) <- chnls
    res
  }
  getStats(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(x, channel = NULL,
  inverse = FALSE, only.function = TRUE, ...)

```

### Arguments

x	A GatingHierarchy object
...	other arguments equal.spaced logical passed to the breaks function 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 flowJoWorkspace. Each element of L is itself a list of length M, where M is the number of parameters that were transformed for a sample or group of samples in a flowJoWorkspace. For example, if a sample has 10 parameters, and 5 are transformed during analysis, using two different sets of transformations, then L will be of length 2, and each element of L will be of length 5. The elements of L represent channel- or parameter-specific transformation functions that map from raw intensity values to channel-space used by flowJo.

### 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
getTransformations(gh); # return a list transformation functions
  getTransformations(gh, inverse = TRUE); # return a list inverse transformation functions
  getTransformations(gh, channel = "FL1-H") # only return the transformation associated with given channel
  getTransformations(gh, channel = "FL1-H", only.function = FALSE) # return the entire transform object

## End(Not run)
```

---

get_leaf_nodes	<i>get all the leaf nodes</i>
----------------	-------------------------------

---

**Description**

get all the leaf nodes

**Usage**

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

**Arguments**

x	GatingHierarchy/GatingSet object
...	arguments passed to 'getNode'" method

**Value**

the leaf nodes

---

gh_check_cluster_node	<i>check if a node is clustering node</i>
-----------------------	---

---

**Description**

check if a node is clustering node

**Usage**

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

**Arguments**

gh	GatingHierarchy
node	the population/node name or path

**Value**

the name of the clustering method. If it is not cluster node, returns NULL

---

gh\_get\_cluster\_labels *Retrieve the cluster labels from the cluster nodes*

---

### Description

Clustering results are stored as individual gated nodes. This helper function collect all the gating indices from the same clustering run (identified by 'parent' node and 'cluster\_method\_name" and merge them as a single factor.

### Usage

```
gh_get_cluster_labels(gh, parent, cluster_method_name)
```

### Arguments

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

---

groupByChannels *split GatingSets into groups based on their flow channels*

---

### Description

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

### Usage

```
groupByChannels(x)
```

### Arguments

x	a list of GatingSets
---	----------------------

### Examples

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

## End(Not run)
```

---

groupByTree	<i>split GatingSets into groups based on their gating schemes Be careful that the splitted resluts still points to the original data set!!</i>
-------------	--

---

### 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 `getNode(gh, path = "auto", showHidden = F)` returns the same set

### Usage

```
groupByTree(x)
```

### Arguments

x a list of GatingSets or one GatingSet

### Value

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

### Examples

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

## End(Not run)
```

---

insertGate	<i>insert a dummy gate to the GatingSet</i>
------------	---

---

### Description

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

### Usage

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

### Arguments

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

**Value**

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

**Examples**

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

## End(Not run)
```

---

isGated	<i>The flags of gate nodes isGated checks if a node is already gated isNegated checks if a node is negated. isHidden checks if a node is hidden.</i>
---------	--

---

**Description**

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

**Usage**

```
isGated(obj, y, ...)
```

## S4 method for signature 'GatingHierarchy,character'

```
isGated(obj, y)
```

```
isNegated(obj, y, ...)
```

## S4 method for signature 'GatingHierarchy,character'

```
isNegated(obj, y)
```

```
isHidden(obj, y, ...)
```

## S4 method for signature 'GatingHierarchy,character'

```
isHidden(obj, y)
```

**Arguments**

obj	GatingHierarchy
y	node/gating path
...	not used

---

isNcdf	<i>determine the flow data associated with a Gating Hierarchy is based on 'ncdfFlowSet' or 'flowSet'</i>
--------	--

---

**Description**

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

**Usage**

```
isNcdf(x)
```

**Arguments**

x	GatingHierarchy object
---	------------------------

**Value**

logical

---

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 <a href="#">keyword-methods</a>

**Details**

See keyword in Package 'flowCore'

**See Also**

[keyword-methods](#)

**Examples**

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

## End(Not run)
```

---

lapply,GatingSet-method

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

---

**Description**

sample names are used for names of the returned list

**Usage**

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

**Arguments**

X	GatingSet
FUN	function to be applied to each sample in 'GatingSet'
...	other arguments to be passed to 'FUN'

---

length, GatingSet-method

*Methods to get the length of a GatingSet*

---

### Description

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

### Usage

```
## 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

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

### Arguments

T, M, W, A	see <a href="#">logicleGml2</a>
n	desired number of breaks (the actual number will be different depending on the data range)
equal.space	whether breaks at equal-spaced intervals

### Value

a logicleGml2 transformation object

**Examples**

```
trans.obj <- logicleGm12_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	<i>logicle transformation.</i>
---------------	--------------------------------

---

**Description**

Used for construct logicle transform object.

**Usage**

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

**Arguments**

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

**Value**

a logicle transformation object

**Examples**

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

---

logtGml2_trans	<i>Gating-ML 2.0 Log transformation.</i>
----------------	--

---

### Description

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

### Usage

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

### Arguments

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

### Value

logtGml2 transformation object

### Examples

```
trans.obj <- logtGml2_trans(M = 1, T = 1e3, equal.space = TRUE)
data <- 1:1e3
brks.func <- trans.obj[["breaks"]]
brks <- brks.func(data)
brks # fasin 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
```

---

markernames,GatingHierarchy-method	<i>Get/set the column(channel) or marker names</i>
------------------------------------	--

---

### Description

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

**Usage**

```
## 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**

```
## 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	<i>make a formula from a character vector</i>
-----------	---

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

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

**Examples**

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

---

moveNode	<i>move a node along with all of its descendant nodes to the given ancestor</i>
----------	---

---

**Description**

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

**Usage**

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

**Arguments**

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

**Examples**

```

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

```

---

ncFlowSet

*Fetch the flowData object associated with a GatingSet .*


---

**Description**

Deprecated by flowData method

Deprecated by flowData method

---

openWorkspace

*It is now moved along with entire flowJo parser to CytoML package*


---

**Description**

It is now moved along with entire flowJo parser to CytoML package

**Usage**

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

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

**Arguments**

file	xml file
...	other arguments

---

pData,GatingHierarchy-method

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

---

### Description

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

### Usage

```
## S4 method for signature 'GatingHierarchy'
pData(object)

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

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

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

### Arguments

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

### Value

a data.frame

---

plot,GatingSet,missing-method

*plot a gating tree*

---

### Description

Plot a tree/graph representing the GatingHierarchy

### Usage

```
## 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 subtree rooted from 'CD4'
plot(gs, "CD4")

## End(Not run)
```

---

plotGate	<i>Plot gates and associated cell population contained in a GatingHierarchy or GatingSet</i>
----------	--

---

**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 [getNode](#).
- ...
- `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.ligcal` 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 `getNodePath` method (used to control how the gating/node path is displayed)  
... The other additional arguments to be passed to `xyplot`.

### Value

a trellis object if arrange is FALSE,

### References

<http://www.rglab.org/>

### Examples

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

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

## End(Not run)
```

---

plotPopCV,GatingHierarchy-method

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

---

### Description

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

**Usage**

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

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

**Arguments**

x	A GatingHierarchy from or a GatingSet.
m	numeric The number of rows in the panel plot. Now deprecated, uses lattice.
n	numeric The number of columns in the panel plot. Now deprecated, uses lattice.
path	character see <a href="#">getNode</a> s
...	Additional arguments to the barplot methods.
scales	list see <a href="#">barchart</a>

**Details**

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

**Value**

Nothing is returned.

**See Also**

[getPopStats](#)

**Examples**

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

## End(Not run)
```

---

plot\_diff\_tree

*visualize the tree structure difference among the GatingSets*

---

**Description**

visualize the tree structure difference among the GatingSets

**Usage**

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

**Arguments**

x	list of groups(each group is a list of 'GatingSet'). it is usually the outcome from <a href="#">groupByTree</a> .
path	passed to getNodes
...	passed to getNodes

**Examples**

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

## End(Not run)
```

---

pop.MFI *built-in stats functions.*

---

**Description**

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

**Usage**

```
pop.MFI(fr)
```

**Arguments**

fr a flowFrame represents a gated population

**Value**

a named numeric vector

---

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

---

**Description**

Determine tick mark locations and labels for a given channel axis

**Usage**

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

**Arguments**

gh	GatingHiarchy
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**

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

## End(Not run)
```

---

```
recompute,GatingSet-method
```

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

---

**Description**

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

**Usage**

```
## 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 [setGate](#) calls.

---

 rotate\_gate

*Simplified geometric rotation of gates associated with nodes*


---

## Description

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

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

## Usage

```
## 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

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

## Details

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

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

**See Also**

transform\_gate [flowCore::rotate\\_gate](#)

**Examples**

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

## End(Not run)
```

---

sampleNames,GatingHierarchy-method

*Get/update sample names in a GatingSet*

---

**Description**

Return a sample names contained in a GatingSet

**Usage**

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

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

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

**Arguments**

object	or a GatingSet
value	character new sample names

**Details**

The sample names comes from pdata of fs.

**Value**

A character vector of sample names

**Examples**

```
## Not run:
#G is a GatingSet
sampleNames(G)

## End(Not run)
```



---

save_gs	<i>save/load a GatingSet/GatingSetList to/from disk.</i>
---------	--

---

### Description

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

### Usage

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

load_gs(path)

save_gslist(gslist, path, ...)

load_gslist(path)
```

### Arguments

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

### Value

load\_gs returns a GatingSet object load\_gslist returns a GatingSetList object

### See Also

[GatingSet-class](#),[GatingSetList-class](#)

### Examples

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

#G is a GatingSet

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

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

---

scale_gate	<i>Simplified geometric scaling of gates associated with nodes</i>
------------	--

---

### Description

Simplified geometric scaling of gates associated with nodes

### Usage

```
## 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 an accompanying scale transformation applied to its underlying data, see `?ggcyto::rescale_gate`.

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

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

### See Also

transform\_gate [flowCore::scale\\_gate](#)

**Examples**

```
## 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)
```

---

set.count.xml	<i>save the event counts parsed from xml into c++ tree structure</i>
---------------	--

---

**Description**

It is for internal use by the diva parser

**Usage**

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

**Arguments**

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

**Examples**

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

## End(Not run)
```

---

setGate,GatingHierarchy,character,filter-method	<i>update the gate</i>
---	------------------------

---

**Description**

update the population node with a flowCore-compatible gate object

**Usage**

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

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

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

**Arguments**

obj	GatingHierarchy or GatingSet
y	character node name or path
value	filter or filterList or list of filter objects
negated	logical see <a href="#">add</a>
...	other arguments

**Details**

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

**Examples**

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

## End(Not run)
```

---

setNode,GatingHierarchy,character,character-method

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

---

**Description**

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

**Usage**

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

## S4 method for signature 'GatingHierarchy,character,logical'
```

```
setNode(x, y, value)
```

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

### Arguments

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

### Examples

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

## End(Not run)
## Not run:
setNode(gh, 4, FALSE) # hide a node
setNode(gh, 4, TRUE) # unhide a node

## End(Not run)
```

---

 shift\_gate

---

*Simplified geometric translation of gates associated with nodes*


---

### Description

Shift the location of a gate associated with a node of a GatingHierarchy or GatingSet. This method is a wrapper for [shift\\_gate](#) that enables updating of the gate associated with a node of a GatingHierarchy or GatingSet.

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

### Usage

```
## 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

[groupByTree\(x\)](#)  
[groupByChannels\(x\)](#)  
[checkRedundantNodes\(x\)](#)  
[dropRedundantNodes\(x, toRemove\)](#)  
[dropRedundantChannels\(gs\)](#)  
[updateChannels\(gs, map, all = TRUE\)](#)  
[insertGate\(gs, gate, parent, children\)](#)  
[setNode\(x, y, FALSE\)](#)

## 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.

[groupByTree](#) splits the GatingSets into groups based on the gating tree structures.

[groupByChannels](#) split GatingSets into groups based on their flow channels.

[checkRedundantNodes](#) 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.

[dropRedundantNodes](#) removes the terminal(or leaf) nodes that are detected as redundant by [checkRedundantNodes](#).

[dropRedundantChannels](#) remove the redundant channels that are not used by any gate defined in the GatingSet.

[updateChannels](#) modifies the channel names in place. (Usually used to standardize the channels among GatingSets due to the letter case discrepancies or typo).

[insertGate](#) inserts a dummy gate to the GatingSet. Is is useful trick to deal with the extra non-leaf node in some GatingSets that can not be simply removed by [dropRedundantNodes](#)

[setNode](#) 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 <a href="#">subset</a>
...	other arguments. (not used)

**Value**

a codeGatingSet or GatingSetList object

---

swap_data_cols	<i>Swap the colnames Perform some validity checks before returning the updated colnames</i>
----------------	---

---

**Description**

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

**Usage**

```
swap_data_cols(cols, swap_cols)
```

**Arguments**

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

**Value**

the new colname vector that has some colnames swapped

**Examples**

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



---

transform,GatingSet-method

*transform the flow data associated with the GatingSet*

---

## Description

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

## Usage

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

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

## Arguments

<code>_data</code>	GatingSet or GatingSetList
<code>translist</code>	expect a transformList object or a list of transformList objects(with names matched to sample names)
<code>...</code>	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	<i>Constructor for transformerList object</i>
-----------------	---

---

### Description

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

### Usage

```
transformerList(from, trans)
```

### Arguments

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

### Examples

```
library(scales)
#create tranformer 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 <- asinhtGm12_trans()
transformerList(c("FL1-H", "FL2-H"), list(trans.obj, trans.obj2))
```

---

transform_gate	<i>Simplified geometric transformations of gates associated with nodes</i>
----------------	--

---

### Description

Perform geometric transformations of a gate associated with a node of a [GatingHierarchy](#) or [GatingSet](#). This method is a wrapper for [transform\\_gate](#) that enables updating of the gate associated with a node of a [GatingHierarchy](#) or [GatingSet](#).

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

**Usage**

```
## 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). <b>Important:</b> For ellipsoidGate objects, scale determines scale factors for the major and minor axes of the ellipse, in that order.
deg	An angle in degrees by which the gate should be rotated in the counter-clockwise direction.
rot_center	A separate 2-dimensional center of rotation for the gate, if desired. By default, this will be the center for ellipsoidGate objects or the centroid for polygonGate objects. The rot_center argument is currently only supported for polygonGate objects. It is also usually simpler to perform a rotation and a translation individually than to manually specify the composition as a rotation around a shifted center.
dx	Either a numeric scalar or numeric vector. If it is scalar, this is just the desired shift of the gate in its first dimension. If it is a vector, it specifies both dx and dy as (dx, dy). This provides an alternate syntax for shifting gates, as well as allowing shifts of ellipsoidGate objects in more than 2 dimensions.
dy	A numeric scalar specifying the desired shift of the gate in its second dimension.
center	A numeric vector specifying where the center or centroid should be moved (rather than specifying dx and/or dy)
...	Assignments made to the slots of the particular Gate-type filter object in the form "<slot_name> = <value>"

**Details**

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

First, `transform_gate` will apply any direct alterations to the slots of the supplied Gate-type filter object. For example, if `"mean = c(1, 3)"` is present in the argument list when `transform_gate` is called on 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

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

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

## End(Not run)
```

---

updateChannels

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

---

### Description

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

### Usage

```
updateChannels(gs, map, all = TRUE)
```

### Arguments

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

**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**

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

## End(Not run)
```

---

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

---

**Description**

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

**Usage**

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

**Arguments**

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

**Examples**

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

# subsample 30% cell events at CD3+ node
total.cd3 <- pop.stats[pop == "CD3+", count]
gInd <- seq_len(total.cd3) #create integer index for cd3
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
updateIndices(gh, "CD3+", gInd.logical)
#check the updated pop counts
getStats(gs[[1]], nodes = c("CD3+", "CD4", "CD8")) #note that CD4, CD8 are not updated
#update all the descendants of CD3+
nodes <- getDescendants(gh, "CD3+")
for (node in nodes) suppressMessages(recompute(gh, node))
getStats(gs[[1]], nodes = c("CD3+", "CD4", "CD8")) #now all are update to date
```

# Index

- [, GatingSet, ANY-method  
(GatingSet-class), 25
- [, GatingSetList, ANY-method  
(GatingSet-class), 25
- [[, GatingSet, character-method  
(GatingSet-class), 25
- [[, GatingSet, logical-method  
(GatingSet-class), 25
- [[, GatingSet, numeric-method  
(GatingSet-class), 25
  
- add, 9, 62, 68
- add (add, GatingSet, list-method), 4
- add, GatingHierarchy, factor-method  
(add, GatingSet, list-method), 4
- add, GatingHierarchy, filter-method  
(add, GatingSet, list-method), 4
- add, GatingHierarchy, filters-method  
(add, GatingSet, list-method), 4
- add, GatingHierarchy, logical-method  
(add, GatingSet, list-method), 4
- add, GatingHierarchy, logicalFilterResult-method  
(add, GatingSet, list-method), 4
- add, GatingHierarchy, multipleFilterResult-method  
(add, GatingSet, list-method), 4
- add, GatingHierarchy, quadGate-method  
(add, GatingSet, list-method), 4
- add, GatingSet, filter-method  
(add, GatingSet, list-method), 4
- add, GatingSet, filterList-method  
(add, GatingSet, list-method), 4
- add, GatingSet, filters-method  
(add, GatingSet, list-method), 4
- add, GatingSet, filtersList-method  
(add, GatingSet, list-method), 4
- add, GatingSet, list-method, 4
- add, GatingSetList, filter-method  
(add, GatingSet, list-method), 4
- add, GatingSetList, filterList-method  
(add, GatingSet, list-method), 4
- add, GatingSetList, filters-method  
(add, GatingSet, list-method), 4
- add, GatingSetList, filtersList-method  
(add, GatingSet, list-method), 4
  
- add, GatingSetList, list-method  
(add, GatingSet, list-method), 4
- asinh\_Gml2, 8
- asinhGml2\_trans, 8
  
- barchart, 60
- booleanFilter (booleanFilter-class), 9
- booleanFilter-class, 9
  
- char2booleanFilter  
(booleanFilter-class), 9
- checkRedundantNodes, 10, 14, 71
- clone, 11
- clone, GatingSet-method (clone), 11
- clone-methods (clone), 11
- clone.ncdfFlowSet, 11
- colnames, GatingHierarchy-method  
(markernames, GatingHierarchy-method), 52
- colnames, GatingSet-method  
(markernames, GatingHierarchy-method), 52
- colnames<-, GatingHierarchy-method  
(markernames, GatingHierarchy-method), 52
- colnames<-, GatingSet-method  
(markernames, GatingHierarchy-method), 52
- compensate, GatingSet, ANY-method, 11
- compensate, GatingSetList, ANY-method  
(compensate, GatingSet, ANY-method), 11
- compute\_timestep, 12
- copyNode, 13
  
- dropRedundantChannels, 13, 71
- dropRedundantNodes, 10, 14, 71
  
- ellipsoidGate, 63, 66, 70, 75
- estimateLogicle.GatingHierarchy, 14
- expressionFilter, 9
- extract\_cluster\_pop\_name\_from\_node, 15
  
- filterObject, 68

- filterObject
  - (filterObject,rectangleGate-method), 16
- filterObject,booleanFilter-method
  - (filterObject,rectangleGate-method), 16
- filterObject,ellipsoidGate-method
  - (filterObject,rectangleGate-method), 16
- filterObject,logical-method
  - (filterObject,rectangleGate-method), 16
- filterObject,polygonGate-method
  - (filterObject,rectangleGate-method), 16
- filterObject,rectangleGate-method, 16
- fix\_channel\_slash, 16
- flow\_breaks, 22
- flow\_trans, 23
- flowCore::rotate\_gate, 64
- flowCore::scale\_gate, 66
- flowCore::shift\_gate, 70
- flowCore::transform\_gate, 76
- flowData, 31
- flowData(flowData,GatingSet-method), 17
- flowData,GatingSet-method, 17
- flowData<- (flowData,GatingSet-method), 17
- flowData<-,GatingSet-method
  - (flowData,GatingSet-method), 17
- flowJo.fasinh, 18
- flowJo.flog, 18, 52
- flowJo.fsinh (flowJo.fasinh), 18
- flowJo\_biexp\_trans, 20
- flowJo\_fasinh\_trans, 21
- flowJoTrans, 19, 20, 73
- flowWorkspace (flowWorkspace-package), 4
- flowWorkspace-package, 4
- flowWorkspace.par.get
  - (flowWorkspace.par.set), 22
- flowWorkspace.par.init, 21
- flowWorkspace.par.set, 22
  
- GatingHierarchy, 9, 26, 27, 58, 74
- GatingHierarchy-class, 24
- GatingSet, 24, 27, 58, 74
- GatingSet (GatingSet-class), 25
- GatingSet,flowSet,ANY-method, 24
- GatingSet,GatingHierarchy,character-method
  - (GatingSet,flowSet,ANY-method), 24
- GatingSet-class, 25
- GatingSetList, 71
  
- GatingSetList (GatingSetList-class), 27
- GatingSetList-class, 27
- get\_leaf\_nodes, 44
- getChildren
  - (getParent,GatingSet,character-method), 37
- getChildren,GatingSet,character-method
  - (getParent,GatingSet,character-method), 37
- getCompensationMatrices, 11, 29
- getCompensationObj, 30
- getData, 33
- getData
  - (getData,GatingHierarchy,missing-method), 30
- getData,GatingHierarchy,character-method
  - (getData,GatingHierarchy,missing-method), 30
- getData,GatingHierarchy,missing-method, 30
- getData,GatingHierarchy,missing-method, 30
- getData,GatingSet,character-method
  - (getData,GatingHierarchy,missing-method), 30
- getData,GatingSet,missing-method
  - (getData,GatingHierarchy,missing-method), 30
- getData,GatingSetList,ANY-method
  - (getData,GatingHierarchy,missing-method), 30
- getDescendants, 31
- getFullPath, 32
- getGate
  - (getGate,GatingHierarchy,character-method), 32
- getGate,GatingHierarchy,character-method, 32
- getGate,GatingSet,character-method
  - (getGate,GatingHierarchy,character-method), 32
- getGate,GatingSetList,character-method
  - (getGate,GatingHierarchy,character-method), 32
- getIndiceMat, 33
- getIndices, 31, 41
- getIndices
  - (getIndices,GatingHierarchy,character-method), 34
- getIndices,GatingHierarchy,character-method, 34
- getIndices,GatingSet,name-method, 35
- getLogLevel, 35
- getMergedStats, 36



- getNodeNodes, [10](#), [33](#), [38](#), [39](#), [42](#), [58–60](#)
- getNodeNodes (getNodeNodes, GatingSet-method), [36](#)
- getNodeNodes, GatingSet-method, [36](#)
- getParent
  - (getParent, GatingSet, character-method), [37](#)
- getParent, GatingSet, character-method, [37](#)
- getPopStats, [31](#), [34](#), [41](#), [60](#)
- getPopStats
  - (getProp, GatingHierarchy, character-method), [38](#)
- getPopStats, GatingHierarchy-method
  - (getProp, GatingHierarchy, character-method), [38](#)
- getPopStats, GatingSet-method
  - (getProp, GatingHierarchy, character-method), [38](#)
- getPopStats, GatingSetList-method
  - (getProp, GatingHierarchy, character-method), [38](#)
- getProp
  - (getProp, GatingHierarchy, character-method), [38](#)
- getProp, GatingHierarchy, character-method, [38](#)
- getSingleCellExpression
  - (getSingleCellExpression, GatingSetList, character-method), [40](#)
- getSingleCellExpression, GatingSet, character-method
  - (getSingleCellExpression, GatingSetList, character-method), [40](#)
- getSingleCellExpression, GatingSetList, character-method
  - (keyword, GatingSet, missing-method), [48](#)
- getSingleCellExpressionByGate
  - (getSingleCellExpression, GatingSetList, character-method), [40](#)
- getStats, [41](#)
- getTotal
  - (getProp, GatingHierarchy, character-method), [38](#)
- getTotal, GatingHierarchy, character-method
  - (getProp, GatingHierarchy, character-method), [38](#)
- getTransformations, [43](#), [73](#)
- gh\_check\_cluster\_node, [44](#)
- gh\_get\_cluster\_labels, [45](#)
- grid.layout, [58](#)
- groupByChannels, [45](#), [71](#)
- groupByTree, [10](#), [14](#), [46](#), [61](#), [71](#)
- identifier, GatingSet-method
  - (GatingSet, flowSet, ANY-method), [24](#)
- identifier, GatingSetList-method
  - (GatingSet, flowSet, ANY-method), [24](#)
- identifier<-, GatingSet, character-method
  - (GatingSet, flowSet, ANY-method), [24](#)
- identifier<-, GatingSetList, character-method
  - (GatingSet, flowSet, ANY-method), [24](#)
- isGated, [46](#), [71](#)
- isGated, [47](#)
- isGated, GatingHierarchy, character-method
  - (isGated), [47](#)
- isHidden (isGated), [47](#)
- isHidden, GatingHierarchy, character-method
  - (isGated), [47](#)
- isNcdf, [48](#)
- isNegated (isGated), [47](#)
- isNegated, GatingHierarchy, character-method
  - (isGated), [47](#)
- keyword
  - (keyword, GatingHierarchy, character-method), [48](#)
- keyword, GatingHierarchy, character-method, [48](#)
- keyword, GatingSet, missing-method
  - (keyword, GatingHierarchy, character-method), [48](#)
- keyword, GatingSetList, character-method
  - (keyword, GatingHierarchy, character-method), [48](#)
- keyword, GatingSet, missing-method
  - (keyword, GatingHierarchy, character-method), [48](#)
- keyword, GatingSetList, character-method
  - (keyword, GatingHierarchy, character-method), [48](#)
- keyword, GatingSetList, missing-method
  - (keyword, GatingHierarchy, character-method), [48](#)
- keyword, GatingHierarchy, character-method, [48](#)
- lapply, GatingSet-method, [49](#)
- lattice, [58](#)
- layoutGraph, [57](#)
- length (length, GatingSet-method), [50](#)
- length, GatingSet-method, [50](#)
- load\_gs (save\_gs), [65](#)
- load\_gslist (save\_gs), [65](#)
- logicle\_trans, [50](#), [51](#)
- logicleGml2\_trans, [50](#)
- logicletGml2, [50](#)

- logtGml2, [18](#)
- logtGml2\_trans, [52](#)
- markernames, GatingHierarchy-method, [52](#)
- markernames, GatingSet-method  
(markernames, GatingHierarchy-method),  
[52](#)
- markernames<- , GatingHierarchy-method  
(markernames, GatingHierarchy-method),  
[52](#)
- markernames<- , GatingSet-method  
(markernames, GatingHierarchy-method),  
[52](#)
- merge-GatingSet  
(standardize-GatingSet), [71](#)
- mkformula, [54](#)
- moveNode, [54](#)
- ncFlowSet, [55](#)
- ncFlowSet, GatingSet-method (ncFlowSet),  
[55](#)
- ncFlowSet<- (ncFlowSet), [55](#)
- ncFlowSet<- , GatingSet-method  
(ncFlowSet), [55](#)
- openWorkspace, [55](#)
- pData (pData, GatingHierarchy-method), [56](#)
- pData, GatingHierarchy-method, [56](#)
- pData, GatingSet-method  
(pData, GatingHierarchy-method),  
[56](#)
- pData<- (pData, GatingHierarchy-method),  
[56](#)
- pData<- , GatingSet, data.frame-method  
(pData, GatingHierarchy-method),  
[56](#)
- pData<- , GatingSetList, data.frame-method  
(pData, GatingHierarchy-method),  
[56](#)
- plot (plot, GatingSet, missing-method), [56](#)
- plot, GatingSet, character-method  
(plot, GatingSet, missing-method),  
[56](#)
- plot, GatingSet, missing-method, [56](#)
- plot\_diff\_tree, [60](#)
- plotGate, [57](#)
- plotGate, GatingHierarchy, character-method  
(plotGate), [57](#)
- plotGate, GatingHierarchy, missing-method  
(plotGate), [57](#)
- plotGate, GatingHierarchy, numeric-method  
(plotGate), [57](#)
- plotGate, GatingSet, character-method  
(plotGate), [57](#)
- plotGate, GatingSet, missing-method  
(plotGate), [57](#)
- plotGate, GatingSet, numeric-method  
(plotGate), [57](#)
- plotGate, GatingSetList, character-method  
(plotGate), [57](#)
- plotGate-methods (plotGate), [57](#)
- plotPopCV  
(plotPopCV, GatingHierarchy-method),  
[59](#)
- plotPopCV, GatingHierarchy-method, [59](#)
- plotPopCV, GatingSet-method  
(plotPopCV, GatingHierarchy-method),  
[59](#)
- polygonGate, [63](#), [66](#), [70](#), [75](#)
- pop.MFI, [61](#)
- prettyAxis, [61](#)
- quadGate, [66](#), [75](#)
- rbind2, GatingSetList, missing-method  
(GatingSetList-class), [27](#)
- recompute, [68](#)
- recompute (recompute, GatingSet-method),  
[62](#)
- recompute, GatingSet-method, [62](#)
- recompute, GatingSetList-method  
(recompute, GatingSet-method),  
[62](#)
- rectangleGate, [66](#), [75](#)
- Rm (add, GatingSet, list-method), [4](#)
- Rm, character, GatingHierarchy, character-method  
(add, GatingSet, list-method), [4](#)
- Rm, character, GatingSet, character-method  
(add, GatingSet, list-method), [4](#)
- Rm, character, GatingSetList, character-method  
(add, GatingSet, list-method), [4](#)
- rotate\_gate, [63](#), [63](#), [76](#)
- sampleNames  
(sampleNames, GatingHierarchy-method),  
[64](#)
- sampleNames, GatingHierarchy-method, [64](#)
- sampleNames, GatingSet-method  
(sampleNames, GatingHierarchy-method),  
[64](#)
- sampleNames<-  
(sampleNames, GatingHierarchy-method),  
[64](#)
- sampleNames<- , GatingSet, ANY-method  
(sampleNames, GatingHierarchy-method),  
[64](#)

sampleNames<- ,GatingSet-method  
     (sampleNames,GatingHierarchy-method), 64  
     77  
 save\_gs, 65  
 save\_gslist (save\_gs), 65  
 scale\_gate, 66, 76  
 set.count.xml, 67  
 setGate, 62, 63, 69, 74  
 setGate  
     (setGate,GatingHierarchy,character,filter-method),  
     67  
 setGate,GatingHierarchy,character,filter-method,  
     67  
 setGate,GatingSet,character,filterList-method  
     (setGate,GatingHierarchy,character,filter-method),  
     67  
 setGate,GatingSet,character,list-method  
     (setGate,GatingHierarchy,character,filter-method),  
     67  
 setLogLevel (getLogLevel), 35  
 setNode, 71  
 setNode  
     (setNode,GatingHierarchy,character,character-method),  
     68  
 setNode,GatingHierarchy,character,character-method,  
     68  
 setNode,GatingHierarchy,character,logical-method  
     (setNode,GatingHierarchy,character,character-method),  
     68  
 setNode,GatingSet,character,ANY-method  
     (setNode,GatingHierarchy,character,character-method),  
     68  
 shift\_gate, 69, 69, 76  
 show,booleanFilter-method  
     (booleanFilter-class), 9  
 show,GatingHierarchy-method  
     (GatingHierarchy-class), 24  
 show,GatingSet-method  
     (length,GatingSet-method), 50  
 standardize-GatingSet, 71  
 subset, 72  
 subset.GatingSet, 71  
 swap\_data\_cols, 72  
  
 transform,GatingSet-method, 73  
 transform,GatingSetList-method  
     (transform,GatingSet-method),  
     73  
 transform\_gate, 74, 74  
 transformerList, 74  
  
 updateChannels, 71, 76  
 updateIndices,GatingHierarchy,character,logical-method,