Package 'flowWorkspace'

October 12, 2021

Type Package

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

Version 4.4.0

Date 2011-06-10

Author Greg Finak, Mike Jiang

Maintainer Greg Finak <greg@ozette.ai>, Mike Jiang <mike@ozette.ai>

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 file LICENSE

License_restricts_use yes

LazyLoad yes

Imports Biobase, BiocGenerics, cytolib (>= 2.3.9), lattice, latticeExtra, XML, ggplot2, graph, graphics, grDevices, methods, stats, stats4, utils, RBGL, tools, Rgraphviz, data.table, dplyr, Rcpp, scales, matrixStats, RcppParallel, RProtoBufLib, digest, aws.s3, aws.signature, flowCore(>= 2.1.1), ncdfFlow(>= 2.25.4), DelayedArray, S4Vectors

Collate 'cytoframe.R' 'cytoset.R' 'AllClasses.R' 'getStats.R' 'GatingHierarchy_Methods.R' 'GatingSet_Methods.R' 'GatingSetList_Methods.R' 'RcppExports.R' 'filterObject_Methods.R' 'add_Methods.R' 'copyNode.R' 'cytoctx.R' 'deprecated.R' 'flow_trans.R' 'getDescendants.R' 'getSingleCellExpression.R' 'identifier.R' 'load_fcs.R' 'load_gs.R' 'merge_GatingSet.R' 'merge_gslist.R' 'moveNode.R' 'parse_transformer.R' 'setGate_Methods.R' 'updateIndices.R' 'utils.R' 'zzz.R'

Suggests testthat, flowWorkspaceData (>= 2.23.2), knitr, ggcyto, parallel, CytoML, openCyto

LinkingTo Rcpp, BH(>= 1.62.0-1), RProtoBufLib(>= 1.99.4), cytolib (>= 2.3.7),Rhdf5lib, RcppArmadillo, RcppParallel(>= 4.4.2-1)

VignetteBuilder knitr

biocViews ImmunoOncology, FlowCytometry, DataImport, Preprocessing, DataRepresentation

SystemRequirements GNU make, C++11

Encoding UTF-8

RoxygenNote 7.1.1

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

git_branch RELEASE_3_13

git_last_commit 8b26faf

git_last_commit_date 2021-05-19

Date/Publication 2021-10-12

R topics documented:

`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/

`asinhtGml2_trans` *Inverse hyperbolic sine transformation.*

Description

Used to construct inverse hyperbolic sine transform object.

Usage

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

Arguments

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

Value

asinhtGml2 transformation object

Examples

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

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

---

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

Description

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

Usage

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

Arguments

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

Value

fasinh/fsinh transform function

Examples

```
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")
```

Arguments

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

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)
```

cf_append_cols *Append data columns to a flowFrame*

Description

Append data columns to a flowFrame

Usage

```
cf_append_cols(cf, cols, ctx = .cytoctx_global)
```

Arguments

cf	A cytoframe.
cols	A numeric matrix containing the new data columns to be added. Must has column names to be used as new channel names.

Details

It is used to add extra data columns to the existing flowFrame. It handles keywords and parameters properly to ensure the new flowFrame can be written as a valid FCS through the function write.FCS .

Examples

```
library(flowCore)
data(GvHD)
tmp <- GvHD[[1]]
cf <- flowFrame_to_cytoframe(tmp)
kf <- kmeansFilter("FSC-H"=c("Pop1","Pop2","Pop3"), filterId="myKmFilter")
fres <- filter(cf, kf)
cols <- as.integer(fres@subSet)
cols <- matrix(cols, dimnames = list(NULL, "km"))
cf <- cf_append_cols(cf, cols)
```

cf_backend_type *return the cytoframe backend storage format*

Description

return the cytoframe backend storage format

Usage

```
cf_backend_type(cf)
```

Arguments

cf	cytoframe

Value

one of "mem","h5", "tile"

---

cf_get_uri *Return the file path of the underlying h5 file*

Description

Return the file path of the underlying h5 file

Usage

```
cf_get_uri(cf)

cf_get_h5_file_path(cf)
```

Arguments

cf	cytoframe object

Details

For the in-memory version of cytoframe, it returns an empty string. This can be used to check whether it is on-disk format.

See Also

Other cytoframe/cytoset IO functions: `cf_write_disk()`, `cf_write_h5()`, `cf_write_tile()`, `cs_get_uri()`, `load_cytoframe_from_fcs()`, `load_cytoframe()`, `load_cytoset_from_fcs()`

cf_is_subsetted *check whether a cytoframe/cytoset is a subsetted(by column or by row) view*

Description

check whether a cytoframe/cytoset is a subsetted(by column or by row) view

Usage

```
cf_is_subsetted(x)

cs_is_subsetted(x)
```

Arguments

x	a cytoset or cytoframe

cf_write_disk *Save the cytoframe to disk*

Description

Save the cytoframe to disk

Usage

```
cf_write_disk(
  cf,
  filename,
  backend = get_default_backend(),
  ctx = .cytoctx_global
)
```

Arguments

cf	cytoframe object
filename	the full path of the output file
backend	either "h5" or "tile"
ctx	cytoctx object, see [cytoctx] for details

See Also

Other cytoframe/cytoset IO functions: cf_get_uri(), cf_write_h5(), cf_write_tile(), cs_get_uri(), load_cytoframe_from_fcs(), load_cytoframe(), load_cytoset_from_fcs()

cf_write_h5 *Save the cytoframe as h5 format*

Description

Save the cytoframe as h5 format

Usage

```
cf_write_h5(cf, filename, ctx = .cytoctx_global)
```

Arguments

cf	cytoframe object
filename	the full path of the output h5 file
ctx	cytoctx object, see [cytoctx] for details

See Also

Other cytoframe/cytoset IO functions: cf_get_uri(), cf_write_disk(), cf_write_tile(), cs_get_uri(), load_cytoframe_from_fcs(), load_cytoframe(), load_cytoset_from_fcs()

cf_write_tile *Save the cytoframe as h5 format*

Description

Save the cytoframe as h5 format

Usage

```
cf_write_tile(cf, filename, ctx = .cytoctx_global)
```

Arguments

cf	cytoframe object
filename	the full path of the output file
ctx	cytoctx object, see [cytoctx] for details

See Also

Other cytoframe/cytoset IO functions: cf_get_uri(), cf_write_disk(), cf_write_h5(), cs_get_uri(), load_cytoframe_from_fcs(), load_cytoframe(), load_cytoset_from_fcs()

cleanup — *Remove on-disk files associatated with flowWorkspace data classes*

Description

These methods immediately delete the on-disk storage associated with cytoframe, cytoset, GatingHierarchy, or GatingSet objects

Usage

```
cf_cleanup(cf, ctx = .cytoctx_global)
```

Arguments

cf	a cytoframe, cytoset, GatingHierarchy, or GatingSet object
ctx	cytoctx object, see [cytoctx] for details

Details

this will override tempdir() in determining the top directory under which files can safely be removed.

cleanup_temp — *Remove temporary files associatated with flowWorkspace data classes*

Description

These methods immediately delete the on-disk h5 storage associated with cytoframe, cytoset, GatingHierarchy, or GatingSet objects, but only if it is under the directory pointed to by tempdir() or alternatively specified by the temp_dir option. The temp_dir option should be used with caution as it acts as a guard against accidental removal of non-temporary storage.

Usage

```
cf_cleanup_temp(x, temp_dir = NULL)

cs_cleanup_temp(x, temp_dir = NULL)

gh_cleanup_temp(x, temp_dir = NULL)

gs_cleanup_temp(x, temp_dir = NULL)
```

Arguments

x	a cytoframe, cytoset, GatingHierarchy, or GatingSet object
temp_dir	an optional argument designating another path as temporary storage. If specified this will override tempdir() in determining the top directory under which files can safely be removed.

Details

Use of these functions will generally be unnecessary for most users, but they are provided for workflows that involve repeated creation of such data structures within the same R session to avoid overwhelming temporary storage.

`compensate` *compensate the flow data asssociated with the GatingSet*

Description

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

Usage

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

Arguments

`x`	`GatingSet`, `GatingSetList`, `cytoframe`, or `cytoset`
`spillover`	`compensation` object or spillover matrix or a list of `compensation` objects

Value

a `GatingSet`, `GatingSetList`, `cytoframe`, or `cytoset` object with the underling 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)
```

convert *Methods for conversion between flowCore and flowWorkspace data classes*

Description

These methods perform conversions between flowWorkspace classes (cytoframe/cytoset) and flowCore classes (flowFrame/flowSet) as well as between single-sample and aggregated classes (e.g. between `cytoset` and a list of `cytoframes`)

Usage

```
cytoframe_to_flowFrame(cf)

flowFrame_to_cytoframe(fr, ...)

cytoset_to_flowSet(cs)

flowSet_to_cytoset(
  fs,
  path = tempfile(),
  backend = get_default_backend(),
  tmp = tempfile()
)

cytoset_to_list(cs)
```

Arguments

`cf`	cytoframe object
`fr`	flowframe
`...`	arguments passed to 'load_cytoframe_from_fcs' call
`cs`	cytoset
`fs`	flowSet or ncdfFlowSet
`path`	the h5 path for cytoset
`tmp`	the temp folder when the temporary files are written to during conversion by default, it is system temp path. And it can be changed to the customized location when there is not enough space at system path.

Details

The first set of methods consist of a pair of methods to coerce a `cytoframe` to or from a `flowFrame` and another pair to coerce a `cytoset` to or from a `flowSet`.

The conversion between the two sets of data container classes mostly entails a conversion of the back-end representation of the data. `cytoframe` and `cytoset` objects contain `flowFrame` and

flowSet objects respectively, so coercion of a cytoframe to flowFrame entails moving the data from the 'C'-level data structure to the corresponding exprs, description, and parameters slots. Coercion of a flowFrame to a cytoframe entails creation of the 'C'-level data structure from the flowFrame slots. The names of each of the methods are pretty self-explanatory.

The second set of methods perform disaggregation of data objects that represent multiple samples in to lists of data objects that represent a single sample. The opposite direction is handled by the constructors for the aggregate data classes.

Methods

cytoframe_to_flowFrame(object = "cytoframe") Returns a flowFrame object coerced from a cytoframe object.

flowFrame_to_cytoframe(object = "flowFrame") Returns a cytoframe object coerced from a flowFrame object.

cytoset_to_flowSet(object = "cytoset") Returns a flowSet object coerced from a cytoset object.

flowSet_to_cytoset(object = "flowSet") Returns a cytoset object coerced from a flowSet object.

flowSet_to_list(object = "flowSet") Returns a list of cytoframe objects with names provided by the sampleNames of the original cytoset

flowSet(object = "list) Constructs a cytoset object from a list of cytoframe objects. See documentation for cytoset

cytoset_to_list(object = "cytoset") Returns a list of cytoframe objects with names provided by the sampleNames of the original cytoset

cytoset(object = "list) Constructs a cytoset object from a list of cytoframe objects. See documentation for flowSet

See Also

merge_list_to_gs

Examples

```
library(flowCore)
data("GvHD")
fs <- GvHD[1]
cs <- flowSet_to_cytoset(fs)
cf <- cs[[1, returnType="cytoframe"]]
ff <- cytoframe_to_flowFrame(cf)
```

convert_backend *convert h5 based gs archive to tiledb*

Description

convert h5 based gs archive to tiledb

Usage

```
convert_backend(gs_dir, output_dir)
```

Arguments

gs_dir	existing gs archive path
output_dir	the new gs path

convert_legacy_gs *convert the legacy GatingSet archive (mixed with R and C++ files) to the new format (C++ only)*

Description

Older versions of flowWorkspace represented GatingSet-class objects using a combination of R and C++ files, while newer versions have moved the representation entirely to the C++ level for the sake of efficiency. In order to use GatingSet or GatingSetList archives created in older versions, they will need to be converted to the new format.

Usage

```
convert_legacy_gs(from, to, ...)

convert_legacy_gslist(from, to, ...)
```

Arguments

from	the old archive path
to	the new archive path
...	tmp the path where the temporary files will be written to during the conversion. By default it is system temp folder and sometime it is helpful to be able to customize it to other location when system temp folder is fulll or not succicient when converting big data sets.

Details

Note that it is likely some of the keyword values (mainly offsets e.g. BEGINDATA) may change slightly after the conversion due to the process of rewriting data to FCS files through write.FCS.

Examples

```
## Not run:
convert_legacy_gs(old_gs_path, new_gs_path)

## End(Not run)
```

`cs_add_cytoframe` *Add a cytoframe to a cytoset*

Description

Add a cytoframe to a cytoset

Usage

```
cs_add_cytoframe(cs, sn, cf)
```

Arguments

`cs`	cytoset
`sn`	sample name to be added
`cf`	cytoframe to be added

`cs_get_uri` *Return the path of the underlying data files*

Description

Return the path of the underlying data files

Usage

```
cs_get_uri(x)

cs_get_h5_file_path(x)

gs_get_uri(x)
```

See Also

Other cytoframe/cytoset IO functions: `cf_get_uri()`, `cf_write_disk()`, `cf_write_h5()`, `cf_write_tile()`, `load_cytoframe_from_fcs()`, `load_cytoframe()`, `load_cytoset_from_fcs()`

cs_set_cytoframe *update a cytoframe in a cytoset*

Description

update a cytoframe in a cytoset

Usage

```
cs_set_cytoframe(cs, sn, cf)
```

Arguments

cs	cytoset
sn	sample name
cf	cytoframe

cytoctx *Create cyto context that carries tiledb context parameters*

Description

Create cyto context that carries tiledb context parameters

Convert cytoctx to a list

Usage

```
cytoctx(cred = NULL, num_threads = 1L)

print.cytoctx(x, ...)

ctx_to_list(x)
```

Arguments

cred	credentials for s3 access. It is a list containing elements of "AWS_ACCESS_KEY_ID", "AWS_SECRET_ACCESS_KEY", "AWS_REGION" when NULL, read the default credential file from disk (e.g., ~/.aws/credentials)
num_threads	tiledb multithread parameters
x	cytoctx

Value

a cytoctx object that is a wrapper around the external pointer,thus it is copy-by-reference object

cytoframe — cytoframe*: A reference class for efficiently managing the data representation of a* flowFrame

Description

This class serves the same purpose as the flowFrame class from the flowCore package: to store quantitative data on cell populations from a single FCS run. The primary difference is in the underlying representation of the data. While flowFrame objects store the underlying data matrix in the exprs slot as an R object, cytoframe objects store the matrix (as well as the data from the other slots) in a C data structure that is accessed through an external pointer. This allows for greater optimization of data operations including I/O, parsing, transformation, and gating.

Details

From the user's standpoint, interacting with a cytoframe is very similar to interacting with a flowframe, with one important difference. While operations such as subsetting or copying a flowFrame using the standard R assignment operator (<-) will perform a deep copy of the data in its slots, the same operations on a cytoframe will produce a view to the same underlying data as the original object. This means that changes made to the cytoframe resulting from subsetting or copying will affect the original cytoframe. If a deep copy of the underyling data is desired, the realize_view method will accomplish this.

Because the cytoframe class inherits from flowFrame, the flowFrame slots are present but not utilized. Thus, attempting to access them directly will yield empty data structures. However, the exprs, parameters, or description methods work in a manner similar to a flowFrame by accessing the same information from the underlying data structure.

Methods

Many of the methods here have their own documentation pages or are more extensively explained in the documentation for flowFrame, so those documentation pages may be consulted as well for more details.

[Subsetting. Returns an object of class cytoframe. The syntax for subsetting is similar to that of data.frames. In addition to the usual index vectors (integer and logical by position, character by parameter names), cytoframes can be subset via filterResult and filter objects.

Usage:

```
cytoframe[i,j]

cytoframe[filter,]

cytoframe[filterResult,]
```

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

$ Subsetting by channel name. This is similar to subsetting of columns of data.frames, i.e., `frame$FSC.H` is equivalent to `frame[,"FSC.H"]`. Note that column names may have to be quoted if they are not valid R symbols (e.g. `frame$"FSC-H"` or `frame$`FSC-H``).

exprs, exprs<- `exprs` returns an object of class `matrix` containing the measured intensities. Rows correspond to cells, columns to the different measurement channels. The `colnames` attribute of the matrix should hold the names or identifiers for the channels. The `rownames` attribute would usually not be set.

`exprs<-` replaces the raw data intensities. The replacement value must be a numeric matrix with `colnames` matching the parameter definitions. Implicit subsetting is allowed (i.e. less columns in the replacement value compared to the original `cytoframe`), but all columns must be defined in the original `cytoframe`.

Usage:

```
exprs(cytoframe)

exprs(cytoframe) <-value
```

head, tail Show first/last elements of the raw data matrix

Usage:

```
head(cytoframe)

tail(cytoframe)
```

keyword, keyword<- Extract all entries or a single entry from the annotations by keyword or replace the entire list of key/value pairs with a new named list. See keyword for details.

Usage:

```
keyword(cytoframe)

keyword(cytoframe,character)

keyword(cytoframe) <-list(value)
```

parameters, parameters<- Extract parameters and return an object of class AnnotatedDataFrame containing information about each column of the `cytoframe`, or replace such an object.

This information will generally be filled in by `load_cytoframe_from_fcs` or similar functions using data from the FCS keywords describing the parameters. To access the actual pa-

rameter annotation, use pData(parameters(cytoframe)).

Replacement is only valid with AnnotatedDataFrames containing all varLabels name, desc, range, minRange and maxRange, and matching entries in the name column to the colnames of the exprs matrix. See parameters for more details.

Usage:

```
parameters(cytoframe)

parameters(cytoframe) <-value
```

show Display details about the cytoframe object.

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

Usage:

```
summary(cytoframe)
```

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

Usage:

```
plot(cytoframe,...)

plot(cytoframe,character,...)

plot(cytoframe,smooth=FALSE,...)
```

ncol, nrow, dim Extract the dimensions of the data matrix.

Usage:

```
ncol(cytoframe)

nrow(cytoframe)
```

```
dim(cytoframe)
```

featureNames, colnames, colnames<- `colnames` and `featureNames` are synonyms. They extract parameter names (i.e., the colnames of the data matrix). For `colnames` there is also a replacement method. This will update the `name` column in the `parameters` slot as well.

Usage:

```
featureNames(cytoframe)

colnames(cytoframe)

colnames(cytoframe) <-value
```

markernames, markernames<- Access or replace the marker names associated with the channels of the `cytoframe`. For replacement, `value` should be a named list or character vector where the names correspond to the channel names and the values correpond to the marker names.

Usage:

```
markernames(object)

markernames(object) <-value
```

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

Usage:

```
names(cytoframe)
```

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

Usage:

```
identifier(cytoframe)
```

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

Parameters:

x: cytoframe object.

type: Range type. either "instrument" or "data". Default is "instrument"

Usage:

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

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

Usage:

```
each_row(cytoframe,function,...)

each_col(cytoframe,function,...)
```

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

Usage:

```
transform(cytoframe,translist,...)
```

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

Usage:

```
filter(cytoframe,filter)
```

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

Usage:

```
split(cytoframe,filter,flowSet=FALSE,...)
```

```
split(cytoframe,filterResult,flowSet=FALSE,...)

split(cytoframe,factor,flowSet=FALSE,...)
```

Subset Subset a `cytoframe` according to a `filter` or a logical vector. The same can be done using the standard subsetting operator with a `filter`, `filterResult`, or a logical vector as first argument.

Usage:

```
Subset(cytoframe,filter)

Subset(cytoframe,logical)
```

cbind2 Not yet implemented.
Expand a `cytoframe` by the data in a `numeric` `matrix` of the same length. The `matrix` must have column names different from those of the `cytoframe`. The additional method for `numerics` only raises a useful error message.

Usage:

```
cbind2(cytoframe,matrix)

cbind2(cytoframe,numeric)
```

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

Usage:

```
compensate(cytoframe,matrix)

compensate(cytoframe,data.frame)

compensate(cytoframe,compensation)
```

decompensate Not yet implemented.
Reverse the application of a compensation matrix (or a `compensation` object) on a `cytoframe` object. This returns a decompensated `cytoframe`.

Usage:

```
decompensate(cytoframe,matrix)
```

```
decompensate(cytoframe,data.frame)
```

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

Usage:

```
spillover(cytoframe)
```

realize_view Returns a new `cytoframe` with its own copy of the underlying data (a deep copy). The optional `filepath` argument accepts a string to specify a full filename for storing the new copy of the data in h5 format.

Usage:

```
realize_view(cytoframe,filepath)
```

See Also

`flowSet`, `read.FCS`

---

cytoframe-labels *Methods to change channel and marker names for* `cytoframe` *and* `cytoset` *objects*

Description

The methods allow direct alteration of channel names or marker names of `cytoframe` and `cytoset` objects. These objects are accessed by reference and changed in place, so there is no need to assign the return value of these methods.

Usage

```
cf_swap_colnames(x, col1, col2)

cf_rename_channel(x, old, new)

cf_rename_marker(x, old, new)

cs_swap_colnames(x, col1, col2)
```

Arguments

x	a cytoframe
col1	first channel name to swap
col2	second channel name to swap
old	old channel or marker name to be changed
new	new channel or marker name after change

---

cytoset — cytoset: *a reference class for efficiently managing the data representation of a* flowSet

---

Description

This class is a container for a set of cytoframe objects, analagous to a flowSet.

Details

Similar to the distinction between the cytoframe and flowFrame classes, the primary difference between the cytoset and flowSet classes is in the underlying representation of the data. Because cytoset is a reference class, copying or subsetting a cytoset object will return a cytoset pointing to the same underlying data. A deep copy of the data can be obtained via the realize_view method.

There is one notable exception to the typical behavior of most methods returning a cytoframe. The standard extraction operator ([[]]) will by default perform a deep copy of the subset being extracted and return a flowFrame. This is for the sake of compatibility with existing user scripts.

Creating Objects

Objects can be created using cytoset() and then adding samples by providing a cytoframe and sample name to cs_add_cytoframe:

```
cs <- cytoset()
cs_add_cytoframe(cs, "Sample Name", cytoframe)
```

The safest and easiest way to create cytosets directly from FCS files is via the load_cytoset_from_fcs function, and there are alternative ways to specify the files to read. See the separate documentation for details.

Methods

[, [[Subsetting. `x[i]` where `i` is a scalar, returns a `cytoset` object, and `x[[i]]` a `flowFrame` object. In this respect the semantics are similar to the behavior of the subsetting operators for lists. `x[i,j]` returns a `cytoset` for which the parameters of each `cytoframe` have been subset according to `j`, `x[[i,j]]` returns the subset of a single `flowFrame` for all parameters in `j`.

The reason for the default behavior of the extraction operator `[[]]` returning a `flowFrame` rather than `cytoframe` is for backwards compatibility with existing user scripts. This behavior can be overridden to instead return a `cytoframe` with the additional `returnType` argument.

Usage:

```
cytoset[i]

cytoset[i,j]

cytoset[[i]]

cytoset[[i,returnType = "cytoframe"]]
```

get_cytoframe_from_cs Extract a `cytoframe` from a `cytoset` by supplying either a sample name or index and optionally supplying a subset of columns.

The `cytoframe` to be extracted (`i` argument) can be specified using its sample name (character) or index in the cytoset (int/numeric). Columns (`j` argument) can be specified using channel name (character), index (int/numeric), or logical vector. If this argument is missing, all columns will be selected.

Usage:

(Assuming `cs` is a `cytoset` and `cf` is the extracted `cytoframe`) `cf <-get_cytoframe_from_cs(cs,i,j)` `cf <-get_cytoframe_from_cs(cs,i)`

$ Subsetting by frame name. This will return a single `cytoframe` object. Note that names may have to be quoted if they are not valid R symbols (e.g. `cytoset$"sample 1"`).

colnames, colnames<- Extract or replace the `character` object with the (common) column names of all the data matrices in the `cytoframe`s.

Usage:

```
colnames(cytoset)

colnames(cytoset) <-value
```

identifier, identifier<- Extract or replace the `name` item from the environment.

Usage:

```
identifier(cytoset)

identifier(cytoset) <-value
```

phenoData, phenoData<- Extract or replace the AnnotatedDataFrame containing the phenotypic data for the whole data set. Each row corresponds to one of the cytoframes. The sampleNames of phenoData (see below) must match the names of the cytoframes in the frames environment.

Usage:

```
phenoData(cytoset)

phenoData(cytoset) <-value
```

pData, pData<- Extract or replace the data frame (or columns thereof) containing actual phenotypic information from the phenoData of the underlying data.

Usage:

```
pData(cytoset)

pData(cytoset)$someColumn <-value
```

varLabels, varLabels<- Not yet implemented.
Extract and set varLabels in the AnnotatedDataFrame of the phenoData of the underyling data.

Usage:

```
varLabels(cytoset)

varLabels(cytoset) <-value
```

sampleNames Extract and replace sample names from the phenoData. Sample names correspond to frame identifiers, and replacing them will also replace the GUID for each cytoframe. Note that each sample name needs to be unique.

Usage:

```
sampleNames(cytoset)
```

```
sampleNames(cytoset) <-value
```

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

Usage:

```
keyword(cytoset,list(keywords))

keyword(cytoset,keywords)

keyword(cytoset) <-list(foo="bar")
```

length The number of `cytoframe` objects in the set.

Usage:

```
length(cytoset)
```

show display object summary.

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

Usage:

```
summary(cytoset)
```

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

Usage:

```
fsApply(cytoset,function,...)

fsApply(cytoset,function,use.exprs=TRUE,...)
```

compensate Apply a compensation matrix on all frames in a `cytoset` object. See `compensate` for details.

Usage:

```
compensate(cytoset,matrix)
```

transform Apply a transformation function on all frames of a `cytoset` object. See `transform` for details.

Usage:

```
transform(cytoset,...)
```

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

Usage:

```
filter(cytoset,filter)

filter(cytoset,list(filters))
```

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

Usage:

```
split(cytoset,filter)

split(cytoset,filterResult)

split(cytoset,list(filters))

split(cytoset,factor)
```

Subset Returns a `cytoset` of `cytoframe`s that have been subset according to a `filter` or `filterResult`, or according to a list of such items of equal length as the `cytoset`. See `Subset` for details.

Usage:

```
Subset(cytoset,filter)
```

```
Subset(cytoset,filterResult)

Subset(cytoset,list(filters))
```

rbind2 Not yet implemented.
Combine two `cytoset` objects, or one `cytoset` and one `cytoframe` object.

Usage:

```
rbind2(cytoset,cytoset)

rbind2(cytoset,cytoframe)
```

spillover Compute spillover matrix from a compensation set. See `spillover` for details.

realize_view Returns a new `cytoset` with its own copy of the underlying data (a deep copy). The optional `filepath` argument accepts a string to specify a full directory name for storing the new copies of the data from the FCS files in h5 format.

Usage:

```
realize_view(cytoset,filepath)
```

cs_add_cytoframe Adds a `cytoframe` to the `cytoset` with sample name given by a string.

Usage:

```
cs_add_cytoframe(cytoset,"SampleName",cytoframe)
```

`delete_gs` *delete the archive of GatingSet*

Description

delete the archive of GatingSet

Usage

```
delete_gs(path, ctx = .cytoctx_global)
```

Arguments

`path`	either a local path or s3 path (e.g. "s3://bucketname/gs_path)

`estimateLogicle` *Compute logicle transformation from the flowData associated with a GatingHierarchy*

Description

See details in 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 GatinigSet
 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"
```

---

filter_to_list *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

```
filter_to_list(x)
```

Arguments

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

Value

a list

---

flowjo_biexp *construct the flowJo-type biexponentioal 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

```
flowjo_biexp(
  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 unkown.
inverse	logical whether to return the inverse transformation function.

Examples

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

---

flowjo_biexp_trans *flowJo biexponential transformation.*

Description

Used for constructing biexponential transformation object.

Usage

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

flowJo_biexp_trans(...)
```

Arguments

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

Value

biexponential transformation object

Examples

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

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

print(trans.func(brks))
```

---

flowjo_fasinh *inverse hyperbolic sine transform function*

Description

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

Usage

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

flowjo_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_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)

flowJo_fasinh_trans(...)
```

Arguments

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

Value

fasinh transformation object

Examples

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

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

flowjo_log_trans *flog transform function*

Description

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

Usage

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

Arguments

decade	total number of decades (i.e. log(max)-log(min)
offset	offset to the orignal input(i.e. min value)
scale	the linear scale factor
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

flog(or its inverse) transform function

Examples

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

inverse.trans <- trans[["inverse"]]
inverse.trans(data.trans)

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

```
#different
trans <- flowjo_log_trans(decade = 3, offset = 30)
data.trans <- trans[["transform"]](data.raw)
data.trans
inverse.trans <- trans[["inverse"]]
inverse.trans(data.trans)
```

---

flowWorkspace-deprecated

Deprecated functions in package **flowWorkspace**.

Description

getStats –> gs(/gh)_pop_get_stats

getProp –> gh_pop_get_proportion

getTotal –> gh_pop_get_count

getPopStats –> gs(/gh)_pop_get_stats

getNodes –> gs_get_pop_paths

getParent –> gs_pop_get_parent

getChildren –> gs_pop_get_children

getGate –> gs(/gh)_get_gate

getIndices –> gh_pop_get_indices

isGated –> gh_pop_is_gated

isNegated –> gh_pop_is_negated

isHidden –> gh_pop_is_hidden

getData –> gs(/gh)_get_data

getTransformations –> gh_get_transformations

getCompensationMatrices –> gh_get_compensations

plotGate –> autoplot

setNode –> gs(/gh)_set_node_name/gs(/gh)_set_node_visible

isNcdf –> gs_is_h5

flowData –> gs_cyto_data

flowData<- –> gs_cyto_data<-

getLoglevel –> get_log_level

setLoglevel –> set_log_level

rbind2 –> gslist_to_gs

filterObject –> filter_to_list

add –> gs_pop_add

Rm –> gs_pop_remove

copyNode –> gh_copy_gate

openWorkspace –> open_flowjo_xml

flowJo.flog –> flowjo_log_trans

flowJoTrans –> flowjo_biexp

flowJo_biexp_trans –> flowjo_biexp_trans

flowJo.fasinh –> flowjo_fasinh

flowJo.fsinh –> flowjo_fsinh

flowJo_fasinh_trans –> flowjo_fasinh_trans

getDescendants –> gh_pop_get_descendants

getSingleCellExpression –> gs_get_singlecell_expression

groupByTree –> gs_split_by_tree

groupByChannels –> gs_split_by_channels

checkRedundantNodes –> gs_check_redundant_nodes

dropRedundantNodes –> gs_remove_redundant_nodes

dropRedundantChannels –> gs_drop_redundant_channels

updateChannels –> gs_update_channels

moveNode –> gh_pop_move

setGate –> gs(/gh)_pop_set_gate

updateIndices –> gh_pop_set_indices

getMergedStats –> gs_pop_get_count_with_meta

set.count.xml –> gh_pop_set_xml_count

---

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

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

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

---

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

Description

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

Usage

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

Arguments

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

GatingHierarchy-class *Class GatingHierarchy*

Description

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

Details

There is a one-to-one correspondence between GatingHierarchy objects and FCS files in the flowJo workspace. Each sample (FCS file) is associated with it's 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 <- open_flowjo_xml(wsfile);
 G<-try(flowjo_to_gatingset(ws,path=d,name=1));
 gh <- G[[1]]
 gh_pop_compare_stats(gh);
 gh_plot_pop_count_cv(gh)
 nodes <- gs_get_pop_paths(gh)
 thisNode <- nodes[4]
 require(ggcyto)
 autoplot(gh,thisNode);
 gh_pop_get_gate(gh,thisNode);
 gh_pop_get_data(gh,thisNode)

## End(Not run)
```

GatingSet-class *Class* "GatingSet"

Description

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

Details

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

Slots

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

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

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 <- open_flowjo_xml(wsfile);
 G<-try(flowjo_to_gatingset(ws,execute=TRUE,path=d,name=1));
 gs_plot_pop_count_cv(G);

## End(Not run)
```

GatingSet-methods *constructors for GatingSet*

Description

construct a gatingset with empty trees (just root node)

Usage

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

Arguments

x	a flowSet, ncdfFlowSet, or cytoset
...	arguments passed to flowSet_to_cytoset() when x is a flowSet

Examples

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

## End(Not run)
```

GatingSetList-class *Class* "GatingSetList"

Description

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

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

Usage

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

Arguments

x	a list of GatingSet
samples	character vector specifying the order of samples. if not specified, the samples are ordered as the underlying stored order.

Details

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

See Also

GatingSet GatingHierarchy

Examples

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

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

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

```
    pData(gslist2) <- pd
    pData(gslist2[3:2])

    #plot the gate
    autoplot(gslist2[1:2],5)

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

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

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

    #convert GatingSetList into one GatingSet by merge_list_to_gs
    gs_merged2 <- merge_list_to_gs(gslist2)
    gs_merged2

## End(Not run)

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

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

## End(Not run)
```

`get_default_backend` *get/set the default backend format of cytoframe*

Description

get/set the default backend format of cytoframe

Usage

```
get_default_backend()

set_default_backend(backend = c("h5", "mem", "tile"))
```

Arguments

backend	one of c("h5", "mem", "tile")

---

get_log_level *get/set the log level*

Description

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

Usage

```
get_log_level()

set_log_level(level = "none")
```

Arguments

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

Value

a character that represents the internal log level

Examples

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

---

gh_apply_to_cs *Construct a* GatingSet *using a template*

Description

This uses a GatingHierarchy as a template to apply to other loaded samples in the form of a cytoset, resulting in a GatingSet. The transformations and gates from the template are applied to all samples. The compensation applied to each of the samples can be controlled via the compensation_source argument.

Usage

```
gh_apply_to_cs(x, cs, swap_cols = FALSE, compensation_source = "sample", ...)
```

Arguments

x	GatingHierarchy
cs	a cytoset
swap_cols	for internal usage
compensation_source	One of the following options: • "sample" – each cytoframe will be compensated with the spillover matrix included in its own FCS • "template" – all cytoframes will be compensatied with the spillover matrix of the template GatingHierarchy • "none" – no compensation will be applied
...	not currently used

Value

a GatingSet

---

gh_apply_to_new_fcs *Construct a* GatingSet *using a template and FCS files*

Description

This uses a GatingHierarchy as a template to apply to other loaded samples in the form of a list of FCS files, resulting in a GatingSet. The transformations and gates from the template are applied to all samples.

Usage

```
gh_apply_to_new_fcs(
  x,
  files,
  swap_cols = FALSE,
  backend = get_default_backend(),
  compensation_source = "sample",
  ...
)
```

Arguments

x	GatingHierarchy
swap_cols	for internal usage
backend	the backend storage mode to use for load_cytoset_from_fcs
compensation_source	One of the following options:

- "sample" – each cytoframe will be compensated with the spillover matrix included in its own FCS
- "template" – all cytoframes will be compensatied with the spillover matrix of the template GatingHierarchy
- "none" – no compensation will be applied

... other arguments passed to load_cytoset_from_fcs

Details

This method is still included to support legacy scripts but will deprecated for the more modular workflow of loading a cytoset via load_cytoset_from_fcs followed by gh_apply_to_cs.

---

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

Description

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

Usage

```
gh_copy_gate(gh, node, to)
```

Arguments

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

Examples

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

gh_get_cluster_labels *Retrieve the cluster labels from the cluster nodes*

Description

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

Usage

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

Arguments

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

gh_get_compensations *Retrieve the compensation matrices from a* GatingHierarchy *or* GatingSet

Description

Retrieve the compensation matrices from a GatingHierarchy or GatingSet.

Usage

```
gh_get_compensations(x)

gs_get_compensations(x)
```

Arguments

x	A GatingHierarchy or GatingSet object.

Details

Return all the compensation matrices in a GatingHierarchy or GatingSet

Value

A list of matrix representing the spillover matrix in GatingHierarchy or GatingSet

Examples

```
  ## Not run:
 # Assume gh is a GatingHierarchy and gs is a GatingSet
  gh_get_compensations(gh)
  gs_get_compensations(gs)

## End(Not run)
```

`gh_get_transformations` *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

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

Arguments

`x`	A `GatingHierarchy` object
`channel`	`character` channel name
`inverse`	`logical` whether to return the inverse transformation function. Valid when only.funtion 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.
`...`	other arguments equal.spaced `logical` passed to the breaks functio to determine whether to break at 10^n or equally spaced intervals

Details

Returns a list of the transformations or a transformation in the flowJo workspace. The list is of length `L`, where `L` is the number of distinct transformations applied to samples in the `flowjo_workspace`. Each element of `L` is itself a `list` of length `M`, where `M` is the number of parameters that were transformed for a sample or group of samples in a `flowjo_workspace`. For example, if a sample has 10 parameters, and 5 are transformed during analysis, using two different sets of transformations, then L will be of length 2, and each element of L will be of length 5. The elements of `L` 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
gh_get_transformations(gh); # return a list transformation functions
 gh_get_transformations(gh, inverse = TRUE); # return a list inverse transformation functions
 gh_get_transformations(gh, channel = "FL1-H") # only return the transfromation associated with given channel
 gh_get_transformations(gh, channel = "FL1-H", only.function = FALSE) # return the entire transform object

## End(Not run)
```

---

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

Description

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

Usage

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

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

Arguments

x	A GatingHierarchy from or a GatingSet.
path	character see gs_get_pop_paths
...	Additional arguments to the barplot methods.
scales	list see barchart

Details

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

Value

Nothing is returned.

See Also

gs_pop_get_count_fast

Examples

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

## End(Not run)
```

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

Description

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

Usage

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

Arguments

x	GatingHierarchy
path	see gs_get_pop_paths
...	not used

`gh_pop_get_cluster_name` *check if a node is clustering node*

Description

check if a node is clustering node

Usage

```
gh_pop_get_cluster_name(gh, node)
```

Arguments

gh	GatingHierarchy
node	the population/node name or path

Value

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

---

gh_pop_get_data *get gated flow data from a GatingHierarchy/GatingSet/GatingSetList*

Description

get gated flow data from a GatingHierarchy/GatingSet/GatingSetList

Usage

```
gh_pop_get_data(obj, y = "root", inverse.transform = FALSE, ...)
```

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.
inverse.transform	logical flag indicating whether to inverse transform the data
...	arguments passed to ncdfFlow::[[

Details

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

Value

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

See Also

gs_cyto_data gh_pop_get_indices gh_pop_compare_stats

Examples

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

    #gh is a GatingHierarchy
    gh_pop_get_data(gh)

## End(Not run)
```

---

gh_pop_get_descendants *get all the descendant nodes for the given ancester*

Description

get all the descendant nodes for the given ancester

Usage

```
gh_pop_get_descendants(gh, node, showHidden = TRUE, ...)
```

Arguments

gh	GatingHierarchy
node	the node path
showHidden	whether show hidden nodes
...	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)))
gh_pop_get_descendants(gs[[1]], "CD4")
gh_pop_get_descendants(gs[[1]], "CD8", path = "auto")
```

gh_pop_get_full_path *convert the partial gating path to the full path*

Description

convert the partial gating path to the full path

Usage

```
gh_pop_get_full_path(gh, path)
```

Arguments

gh	GatingHierarchy object
path	the partial gating path

Value

the full gating path

gh_pop_get_indices *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

```
gh_pop_get_indices(obj, y)
```

Arguments

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

Details

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

Value

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

Note

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

See Also

gh_pop_compare_stats

Examples

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

## End(Not run)
```

---

gh_pop_get_indices_mat

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

Description

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

Usage

```
gh_pop_get_indices_mat(gh, y)
```

Arguments

gh	GatingHierarchy object
y	character vector containing the node names

gh_pop_get_proportion *Get count or proportion from populations*

Description

Get count or proportion from populations

Usage

```
gh_pop_get_proportion(x, y, xml = FALSE)

gh_pop_get_count(x, y, xml = FALSE)
```

Arguments

x	GatingHierarchy
y	character node name or path
xml	whether to extract xml stats or openCyto stats

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

Description

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

Usage

```
gh_pop_move(gh, node, to, recompute = TRUE)
```

Arguments

gh	GatingHierarchy
node	the node to be moved
to	the new parent node under which the node will be moved to
recompute	whether to recompute the gates after the node is moved. Default is TRUE.

Examples

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

---

gh_pop_set_indices *directly update event indices without changing gates*

Description

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

Usage

```
gh_pop_set_indices(obj, y, z)
```

Arguments

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

Examples

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

# subsample 30% cell events at CD3+ node
total <- gh_pop_get_count(gh, "root")
gInd <- seq_len(total) #create integer index for cd3
gInd <- sample.int(total, size = total * 0.3) #randomly select 30%
#convert it to logicle index
gInd.logical <- rep(FALSE, total)
gInd.logical[gInd] <- TRUE
#replace the original index stored at GatingHierarchy
gh_pop_set_indices(gh, "CD3+", gInd.logical)
#check the updated pop counts
```

```
gh_pop_get_stats(gs[[1]], nodes = c("CD3+", "CD4", "CD8")) #note that CD4, CD8 are not updated
#update all the descendants of CD3+
nodes <- gh_pop_get_descendants(gh, "CD3+")
for (node in nodes) suppressMessages(recompute(gh, node))
gh_pop_get_stats(gs[[1]], nodes = c("CD3+", "CD4", "CD8")) #now all are update to date
```

---

gh_pop_set_xml_count *save the event counts parsed from xml into c++ tree structure*

Description

It is for internal use by the diva parser

Usage

```
gh_pop_set_xml_count(gh, node, count)
```

Arguments

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

Examples

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

## End(Not run)
```

---

gslist_to_gs *Merge a GatingSetList into a single GatingSet*

Description

Merge a GatingSetList into a single GatingSet

Usage

```
gslist_to_gs(x, ...)
```

Arguments

x	GatingSetList
...	other arguments passed to gslist_to_gs method for ncdfFlowList

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

Description

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

Usage

```
gs_check_redundant_nodes(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 gs_split_by_tree.
path	argumented passed to gs_get_pop_paths. The default value is "auto".
...	other arguments passed to gs_get_pop_paths.

Value

a list of the character vectors inicating 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 <- gs_split_by_tree(gslist)
toRm <- gs_check_redundant_nodes(gs_groups)

## End(Not run)
```

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

Description

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

Usage

```
gs_cyto_data(x, ...)

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

gs_cyto_data(x) <- value
```

Arguments

x	A GatingSet
...	other arugments
inverse.transform	logical flag indicating whether to inverse transform the data
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.

---

gs_get_compensation_internal

extract compensation object from GatingSet

Description

extract compensation object from GatingSet

Usage

```
gs_get_compensation_internal(gs, sampleName)
```

Arguments

gs	GatingSet
sampleName	sample name

gs_get_leaf_nodes *get all the leaf nodes*

Description

get all the leaf nodes

Usage

```
gs_get_leaf_nodes(x, ancestor = "root", ...)

gh_get_leaf_nodes(x, ancestor = "root", ...)
```

Arguments

x	GatingHierarchy/GatingSet object
ancestor	ancestor node where the leaf nodes descend from. Default is 'root'.
...	arguments passed to 'gs_get_pop_paths" method

Value

the leaf nodes

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

Description

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

Usage

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

gh_get_pop_paths(
  x,
  y = NULL,
  order = "regular",
```

```
  path = "full",
  showHidden = FALSE,
  ...
)
```

Arguments

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

Details

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

Value

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

Examples

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

## End(Not run)
```

---

gs_get_singlecell_expression

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

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

gs_get_singlecell_expression_by_gate(...)
```

Arguments

x	A GatingSet or GatingSetList object .
nodes	character vector specifying different cell populations
other.markers	character vector specifying the extra markers/channels to be returned besides the ones derived from "nodes" and "map" argument.It is only valid when threshold is set to FALSE.
swap	logical indicates whether channels and markers of flow data are swapped.
threshold	logical indicates whether to threshold the flow data by setting intensity value to zero when it is below the gate threshold.
marginal	logical indicates whether 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.
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.
inverse.transform	logical flag indicating whether to inverse transform the data
...	other arguments map a named list providing the mapping between node names (as specified in the gating hierarchy of the gating set) and channel names (as specified in either the desc or name columns of the parameters of the associated flowFrames in the GatingSet). see examples. ignore.case whether to ignore case when match the marker names. Default is FALSE.

Value

A list of numerci matrices

Author(s)

Mike Jiang <wjiang2@fhcrc.org>

See Also

gh_pop_get_indices gs_pop_get_count_fast

Examples

```
## Not run:
  #G is a GatingSet
nodes <- c("4+/TNFa+", "4+/IL2+")
res <- gs_get_singlecell_expression(gs, nodes)
res[[1]]

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

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

## End(Not run)
```

---

gs_is_persistent *determine whether the flow data associated with a GatingSet is persistent(on-disk) or in-memory*

Description

determine whether the flow data associated with a GatingSet is persistent(on-disk) or in-memory

Usage

```
gs_is_persistent(x)

gs_is_h5(x)

isNcdf(x)
```

Arguments

x	GatingSet object

Value

logical

---

gs_plot_diff_tree *visualize the tree structure differnece among the GatingSets*

Description

visualize the tree structure differnece among the GatingSets

Usage

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

Arguments

x	list of groups(each group is a list of 'GatingSet'). it is usually the outcome from gs_split_by_tree.
path	passed to getNodes
...	passed to getNodes

Examples

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

## End(Not run)
```

---

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

Description

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

Usage

```
gs_pop_add(gs, gate, validityCheck = TRUE, ...)

gs_pop_remove(gs, node, ...)
```

Arguments

gs	A GatingSet
gate	A flowCore::filter or a list of flowCore::filters or logical vectors to be added to the GatingSet. when logical vectors, they represent the indices of events to be included in the populations. It can be global that represents the index to the original full events or local index that is relative to the parent population cell events. See examples for more details.
validityCheck	logical whether to check the consistency of tree structure across samples. default is TRUE. Can be turned off when speed is prefered to the robustness.
...	some other arguments to specify how the gates are added to the gating tree. • names a character vector of length four,which specifies the population names resulted by adding a quadGate.The order of the names is clock-wise starting from the top left quadrant population. • parent a character scalar to specify the parent node name where the new gate to be added to, by default it is NULL,which indicates the root node • name a character scalar to specify the node name of population that is generated by the gate to be added. • recompute a logical flag • negated: a logical scalar to specify whether the gate is negated,which means the the population outside of the gate will be kept as the result population. It is FALSE by default.
node	A character identifies the population node in a GatingHierrarchy or GatingSet to remove

Value

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

See Also

GatingSet-class

Examples

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

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

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

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

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

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

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

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

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

#plot the gating hierarchy
    plot(gs[[1]])
#remove one node causing the removal of all the descendants
    gs_pop_remove('rectangle', gs = gs)
    gs_get_pop_paths(gs[[1]])

    #add logical vectors as gate
    lg <- sapply(sampleNames(gs), function(sn){
                                    gh <- gs[[sn]]
                        dat <- exprs(gh_pop_get_data(gh, "cd3+"))#get events data matrix for this sample at cd3+ no
```

```
                                    vec <- dat[, "FSC-A"] > 1e4 & data[, "SSC-A"] > 1e5
                                    vec
                                    })
    gs_pop_add(gs, lg, name = "new_bool", parent = "cd3+")

## End(Not run)
```

`gs_pop_get_count_fast` *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

gs_pop_get_count_fast is more useful than getPop. Returns a table of population statistics for all populations in a `GatingHierarchy`/`GatingSet`. Includes the xml counts, openCyto counts and frequencies.

Usage

```
gs_pop_get_count_fast(
  x,
  statistic = c("count", "freq"),
  xml = FALSE,
  subpopulations = NULL,
  format = c("long", "wide"),
  path = "full",
  ...
)

gs_pop_get_count_with_meta(x, ...)
```

Arguments

`x`	a `GatingSet` or `GatingSetList`
`statistic`	`character` specifies the type of population statistics to extract.(only valid when format is "wide"). Either "freq" or "count" is currently supported.
`xml`	`logical` indicating whether the statistics come from xml (if parsed from xml workspace) or from openCyto.
`subpopulations`	`character` vector to specify a subset of populations to return. (only valid when format is "long")
`format`	`character` value of c("wide", "long") specifing whether to origanize the output in long or wide format
`path`	`character` see gs_get_pop_paths
`...`	additional arguments passed to `gs_pop_get_count_fast`

Details

gs_pop_get_count_fast 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

Value

gs_pop_get_count_fast returns a `data.frame` with columns for the population name, xml derived counts, openCyto derived counts, and the population proportions (relative to their parent poupulation).

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

See Also

gs_get_pop_paths

Examples

```
        ## Not run:
        #gh is a GatingHierarchy
        gs_pop_get_count_fast(gh);
        gh_pop_get_stats(gh,gs_get_pop_paths(gh,tsort=T)[5])

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

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

## End(Not run)
```

---

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

Description

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

Usage

```
gh_pop_get_gate(obj, y)

gs_pop_get_gate(obj, y)
```

Arguments

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

Value

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

See Also

gh_pop_get_data gs_get_pop_paths

Examples

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

## End(Not run)
```

---

gs_pop_get_gs *subset gs by population node*

Description

Basically it returns a new GatingSet with only the substree of the given population node

Usage

```
gs_pop_get_gs(gs, pop)
```

Arguments

gs	GatingSet
pop	the population node that will become the new root node

Value

a new GatingSet that share the underlying events data

`gs_pop_get_parent` *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

```
gs_pop_get_parent(obj, y, ...)

gh_pop_get_parent(obj, y, ...)

gs_pop_get_children(obj, y, showHidden = TRUE, ...)

gh_pop_get_children(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 gs_get_pop_paths methods
`showHidden`	`logical` whether to include the hidden children nodes.

Value

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

See Also

gs_get_pop_paths

Examples

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

## End(Not run)
```

`gs_pop_get_stats` *Extract stats from populations(or nodes)*

Description

Extract stats from populations(or nodes)

Usage

```
gs_pop_get_stats(x, ...)

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

Arguments

`x`	a GatingSet or GatingHierarchy
`...`	arguments passed to gs_get_pop_paths method.
`nodes`	the character vector specifies the populations of interest. default is all available nodes
`type`	the character vector specifies the type of pop stats or a function used to compute population stats. when character, it is expected to be either "count" or "percent". Default is "count" (total number of events in the populations). when a function, it takes a flowFrame object through 'fr' argument and return the stats as a named vector.
`xml`	whether to extract xml stats or openCyto stats

`inverse.transform`
: logical flag . Whether inverse transform the data before computing the stats.

`stats.fun.arg`
: a list of arguments passed to 'type' when 'type' is a function.

Value

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

Examples

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

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

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

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

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

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

## End(Not run)
```

---

gs_pop_get_stats_tfilter

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

---

Description

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

Usage

```
gs_pop_get_stats_tfilter(x, ...)

gh_pop_get_stats_tfilter(
  x,
  nodes = NULL,
  type = c("Count", "Frequency"),
  inverse.transform = FALSE,
  stats.fun.arg = list(),
  tfilter = NULL,
  path = c("full", "auto"),
  ...
)
```

Arguments

x	GatingSet or GatingHierarchy
nodes	the character vector specifies the populations of interest. default is all available nodes
type	the character vector specifies the type of pop stats or a function used to compute population stats. When it is a character, it is expected to be either "Count" or "Frequency". Default is "Count" (total number of events in the populations). When it is a function, it takes a flowFrame object through the 'fr' argument and returns the stats as a named vector.
inverse.transform	logical flag . Whether to inverse transform the data before computing the stats.
stats.fun.arg	a list of arguments passed to 'type' when 'type' is a function.
tfilter	Either a list (tmin, tmax) specifying the minimum and maximum of a the time window filter or a GatingHierarchy, whose minimum and maximum time will be used to determine the window. For both x and the reference GatingHierarchy in tfilter, the only channels that will match this filter are "Time" or "time" and the filter will be applied to each event such that only events with time value t where tmin <= t <= tmax will be evaluated.
path, ...	arguments passed to 'gh_get_pop_paths()'

---

gs_pop_set_gate *update the gate*

Description

update the population node with a flowCore-compatible gate object

Usage

```
gh_pop_set_gate(obj, y, value, negated = FALSE, ...)

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

Arguments

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

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

## End(Not run)
```

---

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

Description

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

Usage

```
gh_pop_set_name(x, y, value)

gs_pop_set_name(x, y, value)
```

Arguments

x	GatingHierarchy
y	pop name/path
value	A character the name of the node

Examples

```
## Not run:
    # G is a GatingHierarchy
    gs_get_pop_paths(G[[1]])#return node names
    gh_pop_set_name(G,"L","lymph")

## End(Not run)
```

`gs_pop_set_visibility` *hide/unhide a node*

Description

hide/unhide a node

Usage

```
gh_pop_set_visibility(x, y, value)

gs_pop_set_visibility(x, y, value)
```

Arguments

`x`	`GatingHierarchy` object
`y`	`character` node name or path
`value`	TRUE/FALSE to indicate whether to hide a node

Examples

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

## End(Not run)
```

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

---

Description

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

Usage

```
gs_remove_redundant_channels(gs, ...)
```

Arguments

gs	a GatingSet
...	other arugments passed to gs_get_pop_paths 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 <- gs_remove_redundant_channels(gs)

## End(Not run)
```

---

gs_remove_redundant_nodes

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

Description

It is usually called after gs_split_by_tree and gs_check_redundant_nodes. The operation is done in place through external pointers which means all the orginal GatingSets are modified.

Usage

```
gs_remove_redundant_nodes(x, toRemove)
```

Arguments

x	GatingSet or list of groups(each group is a list of 'GatingSet'). When it is a list, it is usually the outcome from gs_split_by_tree.
toRemove	list of the node sets to be removed. its length must equals to the length of 'x'. When x is a list, toRemove is usually the outcome from gs_check_redundant_nodes.

Examples

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

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

## End(Not run)
```

`gs_split_by_channels` *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

```
gs_split_by_channels(x)
```

Arguments

`x`	a list of GatingSets

Examples

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

## End(Not run)
```

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

Description

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

Usage

```
gs_split_by_tree(x)
```

Arguments

x	a list of GatingSets or one GatingSet

Value

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

Examples

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

## End(Not run)
```

gs_update_channels *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

```
gs_update_channels(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 underling 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 <- gs_update_channels(gs, map = data.frame(old = c("Qdot 655-A")
                                               , new = c("QDot 655-A")
                                               )
                          )

## End(Not run)
```

---

identifier-methods *Retrieve/replace the GUID of a GatingSet or GatingSetList*

Description

Retrieve or replace the GUID (globally unique identifier) for a GatingSet or GatingSetList

Usage

```
identifier(object)

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

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

Arguments

object	a GatingSet or GatingSetList
value	string

`is_tiledb_support` *check whether cytolib is build with tiledb support*

Description

check whether cytolib is build with tiledb support

Usage

```
is_tiledb_support()
```

Value

TRUE or FALSE

`keyword` *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", ...)
```

Arguments

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

Details

See `keyword` in Package 'flowCore'

See Also

`keyword-methods`

Examples

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

## End(Not run)
```

---

keyword-mutators *Methods to alter keywords in* cytoframe, cytoset, GatingHierarchy, *or* GatingSet *objects*

Description

These methods allow for direct insertion, deletion, or renaming of keywords in cytoframe, cytoset, GatingHierarchy, or GatingSet objects.

Usage

```
cf_keyword_insert(cf, keys, values)

cf_keyword_delete(cf, keys)

cf_keyword_rename(cf, old_keys, new_keys)

cf_keyword_set(cf, keys, values)

cs_keyword_insert(cs, keys, values)

cs_keyword_delete(cs, keys)

cs_keyword_rename(cs, old_keys, new_keys)

cs_keyword_set(cs, keys, values)

gh_keyword_insert(gh, keys, values)

gh_keyword_delete(gh, keys)

gh_keyword_rename(gh, old_keys, new_keys)
```

```
gh_keyword_set(gh, keys, values)

gs_keyword_insert(gs, keys, values)

gs_keyword_delete(gs, keys)

gs_keyword_rename(gs, old_keys, new_keys)

gs_keyword_set(gs, keys, values)
```

Arguments

cf	a cytoframe
keys	the keyword names to insert/delete/replace – single value or vector
values	the values to associate with the supplied keywords – single value or vector of sample length as keys
old_keys	the old keyword name (for renaming)
new_keys	the new keyword name (for renaming)
gh	a GatingHierarchy
gs	a GatingSet

Details

Each of the methods taking two character vectors (keys/values or old_keys/new_keys) will also accept a single named vector for flexibility in usage.

For the functions that take a vector of keys and a vector of values (the keyword_insert and keyword_set functions), the names of this vector should be the keys to which the values of the vector will be assigned.

For the keyword_rename functions, the names of this vector should be the existing keyword names (old_keys) while the values should be the replacement keyword names (new_keys).

See examples for details

Examples

```
library(flowCore)
data(GvHD)
cs <- flowSet_to_cytoset(GvHD[1:2])

keys <- c("CYTNUM", "CREATOR")

# Values before changes
keyword(cs, keys)

# Set two keyword values using separate key and values vectors
values <- c("E3598", "CELLQuest  3.4")
cs_keyword_set(cs, keys, values)
```

```
# Values after changes
keyword(cs, keys)

# Change the values again using a single named vector
values <- c("E3599", "CELLQuest  3.5")
names(values) <- keys
cs_keyword_set(cs, values)

# Values after changes
keyword(cs, keys)
```

---

lapply-methods *apply* FUN *to each sample (i.e.* GatingHierarchy *or* cytoframe*) in a* GatingSet *or* cytoset

---

Description

sample names are used for names of the returned list

Usage

```
lapply(X, FUN, ...)
```

Arguments

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

---

length *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

---

load_cytoframe *Load the cytoframe from disk*

Description

Load the cytoframe from disk

Usage

```
load_cytoframe(uri, on_disk = TRUE, readonly = on_disk, ctx = .cytoctx_global)
```

Arguments

uri	path to the cytoframe file
on_disk	logical flag indicating whether to keep the data on disk and load it on demand. Default is TRUE.
readonly	logical flag indicating whether to open h5 data as readonly. Default is TRUE. And it is valid when on_disk is set to true.
ctx	cytoctx object, see [cytoctx] for details

See Also

Other cytoframe/cytoset IO functions: cf_get_uri(), cf_write_disk(), cf_write_h5(), cf_write_tile(), cs_get_uri(), load_cytoframe_from_fcs(), load_cytoset_from_fcs()

---

load_cytoframe_from_fcs

Read a single FCS file in to a cytoframe

Description

Similar to read.FCS, this takes a filename for a single FCS file and returns a cytoframe.

Usage

```
load_cytoframe_from_fcs(
  filename,
  transformation = "linearize",
  which.lines = NULL,
  alter.names = FALSE,
  column.pattern = NULL,
  invert.pattern = FALSE,
  decades = 0,
  is_h5 = NULL,
  backend = get_default_backend(),
  uri = NULL,
  h5_filename = NULL,
  min.limit = NULL,
  truncate_max_range = TRUE,
  dataset = NULL,
  emptyValue = TRUE,
  num_threads = 1,
  ignore.text.offset = FALSE,
  text.only = FALSE
)
```

Arguments

filename	The filename of the single FCS file to be read
transformation	A character string that defines the type of transformation. Valid values are linearize (default), linearize-with-PnG-scaling, or scale. The linearize transformation applies the appropriate power transform to the data. The linearize-with-PnG-scaling transformation applies the appropriate power transform for parameters stored on log scale, and also a linear scaling transformation based on the "gain" (FCS $PnG keywords) for parameters stored on a linear scale. The scale transformation scales all columns to $[0, 10^{decades}]$. defaulting to $decades = 0$ as in the FCS4 specification. A logical can also be used: TRUE is equal to linearize and FALSE(or NULL) corresponds to no transformation. Also, when the transformation keyword of the FCS header is set to "custom" or "applied", no transformation will be used.
which.lines	Numeric vector to specify the indices of the lines to be read. If it is NULL, all the records are read. If it is of length 1, a random sample of the size indicated by which.lines is read in.
alter.names	Logical indicating whether or not we should rename the columns to valid R names using make.names. The default is FALSE.
column.pattern	An optional regular expression defining parameters we should keep when loading the file. The default is NULL.
invert.pattern	Logical. By default, FALSE. If TRUE, inverts the regular expression specified in column.pattern. This is useful for indicating the channel names that we do not want to read. If column.pattern is set to NULL, this argument is ignored.
decades	When scaling is activated, the number of decades to use for the output.

`is_h5` Logical indicating whether the data should be stored in h5 format

`h5_filename` String specifying a name for the h5 file if `is_h5` is TRUE

`min.limit` The minimum value in the data range that is allowed. Some instruments produce extreme artifactual values. The positive data range for each parameter is completely defined by the measurement range of the instrument and all larger values are set to this threshold. The lower data boundary is not that well defined, since compensation might shift some values below the original measurement range of the instrument. This can be set to an arbitrary number or to `NULL` (the default value), in which case the original values are kept.

`truncate_max_range` Logical. Default is TRUE. can be optionally turned off to avoid truncating the extreme positive value to the instrument measurement range, i.e. '$PnR'.

`dataset` The FCS file specification allows for multiple data segments in a single file. Since the output of `load_cytoframe_from_cytoset` is a single `cytoframe` we can't automatically read in all available sets. This parameter allows the user to choose one of the subsets for import. Its value should be an integer in the range of available data sets. This argument is ignored if there is only a single data segment in the FCS file.

`emptyValue` Logical indicating whether or not to allow empty values for keywords in TEXT segment. It affects how double delimiters are treated. If TRUE, double delimiters are parsed as a pair of start and end single delimiters for an empty value. Otherwise, double delimiters are parsed as one part of the string of the keyword value. The default is TRUE.

`num_threads` Integer allowing for parallelization of the parsing operation by specifiying a number of threads

`ignore.text.offset` Logical indicating whether to ignore the keyword values in TEXT segment when they don't agree with the HEADER. Default is FALSE, which throws the error when such a discrepancy is found. Users can turn it on to ignore the TEXT segment when they are sure of the accuracy of the HEADER segment so that the file still can be read.

`text.only` whether to only parse text section of FCS (default is FALSE), it is sometime useful to skip loading data section for the faster loading meta data from FCS `read.AnnotatedDataFrame`, see details

Details

The function `load_cytoframe_from_fcs` works with the output of the FACS machine software from a number of vendors (FCS 2.0, FCS 3.0 and List Mode Data LMD). However, the FCS 3.0 standard includes some options that are not yet implemented in this function. If you need extensions, please let us know. The output of the function is an object of class `cytoframe`.

For specifications of FCS 3.0 see `http://www.isac-net.org` and the file `../doc/fcs3.html` in the doc directory of the package.

The `which.lines` arguments allow you to read a subset of the record as you might not want to read the thousands of events recorded in the FCS file. It is mainly used when there is not enough memory to read one single FCS (which probably will not happen). It will probably take more time than reading the entire FCS (due to the multiple disk IO).

Value

An object of class cytoframe that contains the data, the parameters monitored, and the keywords and values saved in the header of the FCS file.

See Also

Other cytoframe/cytoset IO functions: cf_get_uri(), cf_write_disk(), cf_write_h5(), cf_write_tile(), cs_get_uri(), load_cytoframe(), load_cytoset_from_fcs()

---

load_cytoset_from_fcs *Read one or several FCS files in to a cytoset*

Description

Similar to read.flowSet, this takes a list of FCS filenames and returns a cytoset.

Usage

```
load_cytoset_from_fcs(
  files = NULL,
  path = ".",
  pattern = NULL,
  phenoData = NULL,
  descriptions,
  name.keyword,
  transformation = "linearize",
  which.lines = NULL,
  alter.names = FALSE,
  column.pattern = NULL,
  invert.pattern = FALSE,
  decades = 0,
  is_h5 = NULL,
  h5_dir = NULL,
  backend = get_default_backend(),
  backend_dir = tempdir(),
  min.limit = NULL,
  truncate_max_range = TRUE,
  dataset = NULL,
  emptyValue = TRUE,
  num_threads = 1,
  ignore.text.offset = FALSE,
  sep = "\t",
  as.is = TRUE,
  name,
  file_col_name = NULL,
  ...
)
```

Arguments

files	Optional character vector with filenames.
path	Directory where to look for the files.
pattern	This argument is passed on to dir, see details.
phenoData	An object of class AnnotatedDataFrame, character or a list of values to be extracted from the cytoframe object, see details.
descriptions	Character vector to annotate the object of class cytoset.
name.keyword	An optional character vector that specifies which FCS keyword to use as the sample names. If this is not set, the GUID of the FCS file is used for sampleNames, and if that is not present (or not unique), then the file names are used.
transformation	see load_cytoframe_from_fcs for details.
which.lines	see load_cytoframe_from_fcs for details.
alter.names	see load_cytoframe_from_fcs for details.
column.pattern	see load_cytoframe_from_fcs for details.
invert.pattern	see load_cytoframe_from_fcs for details.
decades	see load_cytoframe_from_fcs for details.
is_h5	logical indicating whether the data should be stored in h5 format
h5_dir	String specifying a name for the h5 directory for the h5 files if is_h5 is TRUE
min.limit	see load_cytoframe_from_fcs for details.
truncate_max_range	see load_cytoframe_from_fcs for details.
dataset	see load_cytoframe_from_fcs for details.
emptyValue	see load_cytoframe_from_fcs for details.
num_threads	Integer allowing for parallelization of the parsing operation by specifiying a number of threads
ignore.text.offset	see load_cytoframe_from_fcs for details.
sep	Separator character that gets passed on to read.AnnotatedDataFrame.
as.is	logical that gets passed on to read.AnnotatedDataFrame. This controls the automatic coercion of characters to factors in the phenoData.
name	An optional character scalar used as name of the object.
file_col_name	optionally specify the column name that stores the fcs filename when phenoData is supplied read.AnnotatedDataFrame, see details.
...	Further arguments that get passed on to

Details

There are four different ways to specify the file from which data is to be imported:

First, if the argument phenoData is present and is of class AnnotatedDataFrame, then the file names are obtained from its sample names (i.e. row names of the underlying data.frame). Also column name will be generated based on sample names if it is not there. This column is mainly

used by visualization methods in flowViz. Alternatively, the argument phenoData can be of class character, in which case this function tries to read a AnnotatedDataFrame object from the file with that name by calling read.AnnotatedDataFrame(file.path(path,phenoData),...{}).

In some cases the file names are not a reasonable selection criterion and the user might want to import files based on some keywords within the file. One or several keyword value pairs can be given as the phenoData argument in form of a named list.

Third, if the argument phenoData is not present and the argument files is not NULL, then files is expected to be a character vector with the file names.

Fourth, if neither the argument phenoData is present nor files is not NULL, then the file names are obtained by calling dir(path,pattern).

Value

An object of class cytoset.

See Also

Other cytoframe/cytoset IO functions: cf_get_uri(), cf_write_disk(), cf_write_h5(), cf_write_tile(), cs_get_uri(), load_cytoframe_from_fcs(), load_cytoframe()

---

load_meta *Flush/load meta data (keywords, pData, channels/markers) to/from disk (only valid for on-disk cytoset/cytoframe)*

Description

Flush/load meta data (keywords, pData, channels/markers) to/from disk (only valid for on-disk cytoset/cytoframe)

Usage

```
cf_flush_meta(cf)

cf_load_meta(cf)

cs_flush_meta(cs)

cs_load_meta(cs)
```

Arguments

cf	cytoframe object
cs	cytoset object

lock *Lock/Unlock the cytoset/cytoframe by turning on/off its read-only flag*

Description

Lock/Unlock the cytoset/cytoframe by turning on/off its read-only flag

Usage

```
cf_lock(cf)

cf_unlock(cf)

cs_lock(cs)

cs_unlock(cs)
```

Arguments

cf	cytoframe object
cs	cytoset 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 logicletGml2
n	desired number of breaks (the actual number will be different depending on the data range)
equal.space	whether breaks at equal-spaced intervals

Value

a logicleGml2 transformation object

Examples

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

---

logicle_trans *logicle transformation.*

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` *Gating-ML 2.0 Log transformation.*

Description

Used to construct GML 2.0 flog transformer object.

Usage

```
logtGml2_trans(t = 262144, m = 4.5, n = 6, equal.space = FALSE)
```

Arguments

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

Details

GML 2.0 standard log transform function constructor. The definition is as in the GML 2.0 standard section 6.2 "parametrized logarithmic transformation – flog" This deviates from standard only in the following way. Before applying the logarithmic transformation, non-positive values are assigned the smallest positive value from the input rather than having undefined values (NA) under the transformation.

Value

logtGml2 transformation object

Examples

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

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

markernames *Get/set the column(channel) or marker names*

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
```

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)
```

merge_list_to_gs *Merge a list of GatingSets into a single GatingSet*

Description

It also checks the consistency of the cyto data and gates.

Usage

```
merge_list_to_gs(x, ...)
```

Arguments

x	a list of GatingSets
...	other arguments (not used)

ncFlowSet *Fetch the flowData object associated with a GatingSet .*

Description

Deprecated by `flowData` method

Deprecated by `flowData` method

nodeflags *The flags of gate nodes*

Description

gh_pop_is_gated checks if a node is already gated. gh_pop_is_negated checks if a node is negated. gh_pop_is_hidden checks if a node is hidden.

Usage

```
gh_pop_is_gated(obj, y)

gh_pop_is_negated(obj, y)

gh_pop_is_hidden(obj, y)

gh_pop_is_bool_gate(obj, y)
```

Arguments

obj	GatingHierarchy
y	node/gating path

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

Description

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

Usage

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

Arguments

file	xml file
...	other arguments

pData-methods *read/set pData of flow data associated with* GatingHierarchy*,* GatingSet*, or* GatingSetList

Description

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

Usage

```
pData(object)

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-methods *plot a gating tree*

Description

Plot a tree/graph representing the GatingHierarchy

Usage

```
plot(x,y, ...)
```

Arguments

x	GatingHierarchy or GatingSet. If GatingSet, the first sample will be used to extract gating tree.
y	missing or character specifies.
...	other arguments:

- boolean: TRUE|FALSE logical specifying whether to plot boolean gate nodes. Defaults to FALSE.
- showHidden: TRUE|FALSE logical whether to show hidden nodes
- layout: See layoutGraph in package Rgraphviz
- width: See layoutGraph in package Rgraphviz
- height: See layoutGraph in package Rgraphviz
- fontsize: See layoutGraph in package Rgraphviz
- labelfontsize: See layoutGraph in package Rgraphviz
- fixedsize: See layoutGraph in package Rgraphviz

Examples

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


## End(Not run)
```

plotGate-methods-defunct *Plot gates and associated cell population contained in a* GatingHierarchy *or* GatingSet

Description

Important: The plotGate methods are now defunct and gates should instead be plotted using the autoplot method from the ggcyto package. The plotGate documentation has been left here to ease the transition.

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, ...)
```

Arguments

x	GatingSet or GatingHierarchyobject
y	character the node name or full(/partial) gating path or numeric representing the node index in the GatingHierarchy. or missing which will plot all gates and one gate per page. It is useful for generating plots in a multi-page pdf. Nodes can be accessed with gs_get_pop_paths.
...	• bool logical specifying whether to plot boolean gates. • arrange.main character The title of the main page of the plot. Default is the sample name. Only valid when x is GatingHierarchy • arrange logical indicating whether to arrange different populations/nodes on the same page via arrangeGrob call. • merge logical indicating whether to draw multiple gates on the same plot if these gates share the same parent population and same x,y dimensions/parameters; • projections list of character vectors used to customize x,y axis. By default, the x,y axis are determined by the respective gate parameters. The elements of the list are named by the population name or path (see y). Each element is a pair of named character specifying the channel name(or marker name) for x, y axis. Short form of channel or marker names (e.g. "APC" or "CD3") can be used as long as they can be uniquely matched to the dimentions 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.symbolA 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 paraemter for overlay symbols.
- default.y character specifiying 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.texteither "parent" (the parent population name) or "gate "(the gate name).
- raw.scale logical whether to show the axis in raw(untransformed) scale. Default is TRUE and can be stored as session-wise setting by 'flowWorkspace.par.set("plotGate", list(raw.scale = TRUE))'
- xlim, ylim character can be either "instrument" or "data" which determines the x, y axis scale either by instrument measurement range or the actual data range. or numeric which specifies customized range. They can be stored as session-wise setting by 'flowWorkspace.par.set("plotGate", list(xlim = "instrument"))'
- ...
  path A character or numeric scalar passed to gs_get_pop_paths method (used to control how the gating/node path is displayed)
  ... The other additional arguments to be passed to xyplot.

Value

a trellis object if arrange is FALSE,

References

http://www.rglab.org/

Examples

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

## End(Not run)
```

---

pop_add *Add populations to a GatingHierarchy*

Description

Add populations to a GatingHierarchy

Usage

```
pop_add(gate, gh, ...)

## S3 method for class 'filter'
pop_add(gate, gh, ...)

## S3 method for class 'filters'
pop_add(gate, gh, names = NULL, ...)

## S3 method for class 'quadGate'
pop_add(gate, gh, names = NULL, ...)

## S3 method for class 'logical'
pop_add(gate, gh, parent, name, recompute, cluster_method_name = NULL, ...)

## S3 method for class 'factor'
pop_add(gate, gh, name = NULL, ...)

## S3 method for class 'logicalFilterResult'
pop_add(gate, gh, ...)

## S3 method for class 'multipleFilterResult'
pop_add(gate, gh, name = NULL, ...)

gh_pop_remove(gh, node, ...)
```

Arguments

gate	a gate object that extends `flowCore::filter` or `flowCore::filters`
gh	GatingHierarchy
...	other arguments

names	a character vector of length four,which specifies the population names resulted by adding a quadGate.The order of the names is clock-wise starting from the top left quadrant population.
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	the population name
recompute	whether to recompute the gates
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.
node	population name/path

---

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	GatingHiearchy
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 *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

```
recompute(
  x,
  y = "root",
  alwaysLoadData = FALSE,
  verbose = FALSE,
  leaf.bool = TRUE
)

## S3 method for class 'GatingSet'
recompute(
  x,
  y = "root",
  alwaysLoadData = FALSE,
  verbose = FALSE,
  leaf.bool = TRUE
)

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

Arguments

x	GatingSet or GatingSetList
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 explictily. When TRUE, then it forces data to be loaded to guarantee the gating process to be uninterrupted at the cost of unnecessary data IO.
verbose	default is FALSE
leaf.bool	whether to compute the leaf boolean gate, default is TRUE
...	arguments

Details

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

Usage

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

Arguments

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

Details

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

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

See Also

transform_gate flowCore::rotate_gate

Examples

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

## End(Not run)
```

---

sampleNames *Get/update sample names in a GatingSet*

Description

Return a sample names contained in a GatingSet

Usage

```
sampleNames(object)

sampleNames(object) <- value
```

Arguments

object	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_cytoset	*save/load a cytoset to/from disk. load_cytoset() can load a cytoset from either the archive previously saved by save_cytoset() call or from a folder that contains a collection of inidivudal cytoframe files (either in h5 format or tiledb format)*

Description

Save/load a cytoset to/from the disk.

Usage

```
save_cytoset(cs, path, ...)

load_cytoset(path, verbose = FALSE, ...)
```

Arguments

cs	A cytoset
path	A character scalar giving the path to save/load the cytoset to/from.
...	other arguments passed to save_gs/load_gs
verbose	whether to print details. Default is FALSE.

Value

load_cytoset returns a cytoset object

Examples

```
## Not run:
#cs is a cytoset
save_cytoset(cs, outdir)
cs <-load_cytoset(outdir)

#or from cytoframe on-disk files
# e.g. h5_dir contains the cytoframes in h5 format
cs <- load_cytoset(h5_dir)


## End(Not run)
```

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

Description

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

Retrieve sample names by scanning h5 files from a GatingSet folder

Usage

```
save_gs(
  gs,
  path,
  cdf = NULL,
  backend_opt = c("copy", "move", "skip", "symlink", "link"),
  ctx = .cytoctx_global,
  ...
)

load_gs(
  path,
  h5_readonly = NULL,
  backend_readonly = TRUE,
  select = character(),
  verbose = FALSE,
  ctx = .cytoctx_global
)

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

save_gslist(gslist, path, ...)

load_gslist(path)
```

Arguments

gs	A GatingSet
path	A character scalar giving the path to save/load the GatingSet to/from.
backend_opt	a character scalar. The valid options are :"copy","move","skip","symlink" specifying what to do with the backend data file. Sometimes it is more efficient to move or create a symlink of the existing backend file to the archived folder. It is useful to "skip" archiving backend file if raw data has not been changed.
ctx	cytoctx object, see [cytoctx] for details

...	other arguments: not used.
h5_readonly	whether to open h5 data as read-only. Default is TRUE
select	an integer or character vector to select a subset of samples to load
verbose	logical flag to optionally print the versions of the libraries that were used to archive the GatingSet for troubleshooting purpose.
object	a GatingSet folder
gslist	A GatingSetList

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)
      ## Not run:
        sampleNames(gsdir)

## End(Not run)
```

---

scale_gate *Simplified geometric scaling of gates associated with nodes*

Description

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

scale_gate calls gs_pop_set_gate to modify the provided GatingHierarchy or GatingSet directly so there is no need to re-assign its output. The arguments will be essentially identical to the flowCore method, except for the specification of the target gate. Rather than being called on an object of type 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'
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 accompanyging 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)
```

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

Usage

```
## S3 method for class 'GatingHierarchy'
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 specifiying 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 divinding boundary in each dimension. For ellipsoidGate objects, this will shift the center (and therefore all points of the ellipse). For polgonGate objects, this will simply shift all of the points defining the polygon.

The method allows two different approaches to shifting a gate. Through the dx and/or dy arguments, a direct shift in each dimension can be provided. Alternatively, through the center argument, the gate can be directly moved to a new location in relation to the old center of the gate. For quadGate

objects, this center is the intersection of the two dividing boundaries (so the value of the boundary slot). For rectangleGate objects, this is the center of the rectangle defined by the intersections of the centers of each interval. For ellipsoidGate objects, it is the center of the ellipsoid, given by the mean slot. For polygonGate objects, the centroid of the old polygon will be calculated and shifted to the new location provided by center and all other points on the polygon will be shifted by relation to the centroid.

See Also

transform_gate flowCore::shift_gate

Examples

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

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

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

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

## End(Not run)
```

---

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

Description

gs_split_by_tree(x)

gs_split_by_channels(x)

gs_check_redundant_nodes(x)

gs_remove_redundant_nodes(x, toRemove)

gs_remove_redundant_channels(gs)

gs_update_channels(gs, map, all = TRUE)

gh_pop_move(gh, node, to)

gs_pop_set_visibility(x, y, FALSE)

Details

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

gs_split_by_tree splits the GatingSets into groups based on the gating tree structures.

gs_split_by_channels split GatingSets into groups based on their flow channels.

gs_check_redundant_nodes returns the terminal(or leaf) nodes that makes the gating trees to be different among GatingSets and thus can be considered to remove as redundant nodes.

gs_remove_redundant_nodes removes the terminal(or leaf) nodes that are detected as redundant by `gs_check_redundant_nodes`.

gs_remove_redundant_channels remove the redundant channels that are not used by any gate defined in the GatingSet.

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

gh_pop_move inserts a dummy gate to the GatingSet. Is is useful trick to deal with the extra non-leaf node in some GatingSets that can not be simply removed by `gs_remove_redundant_nodes`

gs_pop_set_visibility hide a node/gate in a GatingSet. It is useful to deal with the non-leaf node that causes the tree structure discrepancy.

`stats.fun` *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 `gh_pop_get_stats` method to perform the sample-wise population stats calculations.

Usage

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

Arguments

`fr`	a flowFrame represents a gated population

Value

a named numeric vector

subset *subset the GatingSet/GatingSetList based on 'pData'*

Description

subset the GatingSet/GatingSetList based on 'pData'

Usage

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

Arguments

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

Value

a codeGatingSet or GatingSetList object

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

Description

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

Usage

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

Arguments

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

Value

the new colname vector that has some colnames swapped

Examples

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

`transform` *tranform the flow data asssociated with the GatingSet*

Description

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

Usage

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

Arguments

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

Value

a `GatingSet` or `GatingSetList` object with the underling flow data transformed.

Examples

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

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

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

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


## End(Not run)
```

`transformerList` *Constructor for transformerList object*

Description

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

Usage

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

Arguments

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

Examples

```
library(flowCore)
library(scales)
#create 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 <- asinhtGml2_trans()
transformerList(c("FL1-H", "FL2-H"), list(trans.obj, trans.obj2))
```

transform_gate *Simplified geometric transformations of gates associated with nodes*

Description

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

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

Usage

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

Arguments

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

rot_center	A separate 2-dimensional center of rotation for the gate, if desired. By default, this will be the center for ellipsoidGate objects or the centroid for polygonGate objects. The rot_center argument is currently only supported for polygonGate objects. It is also usually simpler to perform a rotation and a translation individually than to manually specify the composition as a rotation around a shifted center.
dx	Either a numeric scalar or numeric vector. If it is scalar, this is just the desired shift of the gate in its first dimension. If it is a vector, it specifies both dx and dy as (dx,dy). This provides an alternate syntax for shifting gates, as well as allowing shifts of ellipsoidGate objects in more than 2 dimensions.
dy	A numeric scalar specifying the desired shift of the gate in its second dimension.
center	A numeric vector specifying where the center or centroid should be moved (rather than specifiying 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 accompanyging 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)
```

[,GatingSet,ANY,ANY,ANY-method

Bracket operators on GatingSet *and* GatingSetList *objects*

Description

[subsets a GatingSet or GatingSetList using the familiar bracket notation

[[extracts a GatingHierarchy object from a GatingSet.

Usage

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

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

Arguments

x	a GatingSet or GatingSetList
i	numeric or logical or character used as sample indices
j, ..., drop	unused

Value

The [operator returns an object of the same type as x corresponding to the subset of indices in i, while the [[operator returns a single GatingHierarchy

Index