Package 'CytoPipeline'

April 5, 2026

Title Automation and visualization of flow cytometry data analysis pipelines

Version 1.10.0

Description This package provides support for automation and visualization of flow cytometry data analysis pipelines. In the current state, the package focuses on the preprocessing and quality control part. The framework is based on two main S4 classes, i.e. CytoPipeline and CytoProcessingStep. The pipeline steps are linked to corresponding R functions - that are either provided in the CytoPipeline package itself, or exported from a third party package, or coded by the user her/himself. The processing steps need to be specified centrally and explicitly using either a json input file or through step by step creation of a CytoPipeline object with dedicated methods. After having run the pipeline, obtained results at all steps can be retrieved and visualized thanks to file caching (the running facility uses a BiocFileCache implementation). The package provides also specific visualization tools like pipeline workflow summary display, and 1D/2D comparison plots of obtained flowFrames at various steps of the pipeline.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

BugReports https://github.com/UCLouvain-CBIO/CytoPipeline/issues

URL https://uclouvain-cbio.github.io/CytoPipeline

biocViews FlowCytometry, Preprocessing, QualityControl, WorkflowStep, ImmunoOncology, Software, Visualization

Collate 'CytoPipeline-functions.R' 'CytoPipeline-package.R' 'CytoPipelineClass.R' 'CytoProcessingStep.R' 'CytoProcessingStepImplementations.R' 'data.R' 'gating.R' 'utils.R' 'ggplots.R'

Depends R (>= 4.4)

Imports methods, stats, utils, withr, rlang, ggplot2 (>= 3.4.1), ggcyto, BiocFileCache, BiocParallel, flowCore, PeacoQC, flowAI, diagram, jsonlite, scales

Suggests testthat (>= 3.0.0), vdiffr, diffviewer, knitr, rmarkdown, BiocStyle, reshape2, dplyr, CytoPipelineGUI

VignetteBuilder knitr

Config/testthat/edition 3

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

git_branch RELEASE_3_22

git_last_commit 4cb0101

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2026-04-05

Author Philippe Hauchamps [aut, cre] (ORCID: <https://orcid.org/0000-0003-2865-1852>), Laurent Gatto [aut] (ORCID: <https://orcid.org/0000-0002-1520-2268>), Dan Lin [ctb]

Maintainer Philippe Hauchamps <philippe.hauchamps@uclouvain.be>

Contents

`aggregateAndSample` *Aggregate and sample multiple flow frames of a flow set together*

Description

Aggregate multiple flow frames in order to analyze them simultaneously. A new FF, which contains about nTotalEvents cells, nTotalEvents/nFiles cells from each file. Two new columns are added: a column indicating the original file by index, and a noisy version of this, for better plotting opportunities, This function is based on PeacoQC::AggregateFlowframes() where file names inputs have been replaced by a flowSet input.

Usage

```
aggregateAndSample(
  fs,
  nTotalEvents,
  setup = c("forceNEvent", "forceBalance"),
  seed = NULL,
  channels = NULL,
  writeOutput = FALSE,
  outputFile = "aggregate.fcs",
  keepOrder = FALSE
)
```

Arguments

`fs`	a flowCore::flowset
`nTotalEvents`	Total number of cells to select from the input flow frames
`setup`	How to proceed when nTotalEvents/nFiles is too high for some of the flow frames: • forceBalance (default): compute the minimum nb of events per flow frame, and keep that amount of events from each flow frame. • forceNEvents: try to be as balanced as possible, but force a total of nTotalEvents if possible, i.e. takes all events from the flow frame with too low nb of events, and then fill in the total with events from the bigger flow frames in a balanced way. However, if nTotalEvents is greater than the sum of all events, take all events only once.
`seed`	seed to be set before sampling for reproducibility. Default NULL does not set any seed.
`channels`	Channels/markers to keep in the aggregate. Default NULL takes all channels of the first file.

writeOutput	Whether to write the resulting flowframe to a file. Default FALSE
outputFile	Full path to output file. Default "aggregate.fcs"
keepOrder	If TRUE, the random subsample will be ordered in the same way as they were originally ordered in the file. Default = FALSE.

Value

returns a new flowCore::flowFrame

Examples

```
data(OMIP021Samples)

nCells <- 1000
agg <- aggregateAndSample(
    fs = OMIP021Samples,
    nTotalEvents = nCells)
```

---

appendCellID *append 'Original_ID' column to a flowframe*

Description

: on a flowCore::flowFrame, append a 'Original_ID' column. This column can be used in plots comparing the events pre and post gating. If the 'Original_ID' column already exists, the function does nothing

Usage

```
appendCellID(ff, eventIDs = seq_len(flowCore::nrow(ff)))
```

Arguments

ff	a flowCore::flowFrame
eventIDs	an integer vector containing the values to be added in expression matrix, as Original ID's.

Value

new flowCore::flowFrame containing the added 'Original_ID' column

Examples

```
data(OMIP021Samples)

retFF <- appendCellID(OMIP021Samples[[1]])
```

applyScaleTransforms *apply scale transforms*

Description

wrapper around flowCore::transform() that discards any additional parameter passed in (...) Additionally, some checks regarding channels correspondance is done: if `transList` contains transformations for channels that are not present in `x`, then these transformations are first removed.

Usage

```
applyScaleTransforms(x, transList, verbose = FALSE, ...)
```

Arguments

`x`	a flowCore::flowSet or a flowCore::flowFrame
`transList`	a flowCore::transformList
`verbose`	if TRUE, send a message per flowFrame transformed
`...`	other arguments (not used)

Value

the transformed flowFrame

Examples

```
data(OMIP021Samples)

transListPath <- file.path(system.file("extdata",
                                       package = "CytoPipeline"),
                           "OMIP021_TransList.rds")

transList <- readRDSObject(transListPath)

ff_c <- compensateFromMatrix(OMIP021Samples[[1]],
                             matrixSource = "fcs")

ff_t <- applyScaleTransforms(ff_c, transList = transList)
```

areFluoCols *find flow frame columns that represent fluorochrome channel*

Description

: find flow frame columns that represent fluorochrome channel

Usage

```
areFluoCols(
  x,
  toRemovePatterns = c("FSC", "SSC", "Time", "Original_ID", "File", "SampleID")
)
```

Arguments

x	a flowCore::flowFrame or a flowCore::flowSet
toRemovePatterns	a vector of string patterns that are to be considered as non fluorochrome

Value

a vector of booleans of which the dimension is equal to the number of columns in ff

Examples

```
data(OMIP021Samples)

areFluoCols(OMIP021Samples)
```

---

areSignalCols *find flow frame columns that represent true signal*

Description

: find flow frame columns that represent true signal

Usage

```
areSignalCols(
  x,
  toRemovePatterns = c("Time", "Original_ID", "File", "SampleID")
)
```

Arguments

x	a flowCore::flowFrame or a flowCore::flowSet
toRemovePatterns	a vector of string patterns that are to be considered as non signal

Value

a vector of booleans of which the dimension is equal to the number of columns in ff

Examples

```
data(OMIP021Samples)

areSignalCols(OMIP021Samples)
```

compensateFromMatrix *compensation of fcs file(s) from matrix*

Description

executes the classical compensation function on a flowSet or flowFrame, given a compensation matrix. The matrix can be either retrieved in the fcs files themselves or provided as a csv file.

Usage

```
compensateFromMatrix(
  x,
  matrixSource = c("fcs", "import"),
  matrixPath = NULL,
  updateChannelNames = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

x	a flowCore::flowFrame or flowCore::flowSet
matrixSource	if "fcs", the compensation matrix will be fetched from the fcs files (different compensation matrices can then be applied by fcs file) if "import", uses matrixPath to read the matrix (should be a csv file)
matrixPath	if matrixSource == "import", will be used as the input csv file path
updateChannelNames	if TRUE, updates the fluo channel names by prefixing them with "comp-"
verbose	if TRUE, displays information messages
...	additional arguments (not used)

Value

the compensated flowSet or flowFrame

Examples

```
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
sampleFiles <-
    file.path(rawDataDir, list.files(rawDataDir, pattern = "Donor"))

truncateMaxRange <- FALSE
minLimit <- NULL

# create flowCore::flowSet with all samples of a dataset
fsRaw <- readSampleFiles(
    sampleFiles = sampleFiles,
    whichSamples = "all",
    truncate_max_range = truncateMaxRange,
    min.limit = minLimit)
```

```
suppressWarnings(ff_m <- removeMarginsPeacoQC(x = fsRaw[[2]]))

ff_c <-
    compensateFromMatrix(ff_m,
                         matrixSource = "fcs")
```

---

computeScatterChannelsLinearScale

compute linear transformation of scatter channels found in ff, based on 5% and 95% of referenceChannel, set as target. If there is a transformation defined in transList for referenceChannel, it is applied first, before computing quantiles. Then the computed linear transformations (or each scatter channel) are added into the transfo_list. -A channels are computed, and same linear transformation is then applied to corresponding -W and -H channels (if they exist in ff).

Description

based on a referenceChannel

Usage

```
computeScatterChannelsLinearScale(
  ff,
  transList = NULL,
  referenceChannel,
  silent = TRUE
)
```

Arguments

ff	a flowCore::flowFrame
transList	an initial flowCore::transformList
referenceChannel	the reference channel to take target quantile values from. Can be defined as marker or channel name.
silent	if FALSE, will output some information on the computed linear transformations

Value

the transList with added linear scale transformations

Examples

```
data(OMIP021Samples)

ff <- OMIP021Samples[[1]]
refMarker <- "APCCy7 - CD4"
refChannel <- "780/60Red-A"
transList <- flowCore::estimateLogicle(ff,
                                       channels = refChannel)
```

```
retTransList <-
    computeScatterChannelsLinearScale(ff,
                                      transList = transList,
                                      referenceChannel = refMarker,
                                      silent = TRUE
    )
```

CytoPipeline *CytoPipeline package*

Description

CytoPipeline is a package that provides support for automation and visualization of flow cytometry data analysis pipelines. In the current state, the package focuses on the preprocessing and quality control part.

The framework is based on two main S4 classes, i.e. CytoPipeline and CytoProcessingStep. The CytoProcessingStep defines the link between pipeline step names and corresponding R functions that are either provided in the CytoPipeline package itself, or exported from a third party package, or coded by the user her/himself. The processing steps need to be specified centrally and explicitly using either a json input file or through step by step creation of a CytoPipeline object with dedicated methods.

After having run the pipeline, obtained results at all steps can be retrieved and visualized thanks to file caching (the running facility uses a BiocFileCache implementation). The package provides also specific visualization tools like pipeline workflow summary display, and 1D/2D comparison plots of obtained flowFrames at various steps of the pipeline.

For a step by step example using CytoPipeline, please have a look at the vignette!

Author(s)

Maintainer: Philippe Hauchamps <philippe.hauchamps@uclouvain.be> (ORCID)

Authors:

- Laurent Gatto <laurent.gatto@uclouvain.be> (ORCID)

Other contributors:

- Dan Lin <dan.8.lin@gsk.com> [contributor]

See Also

CytoPipelineClass, CytoProcessingStep

`CytoPipeline-class` *CytoPipeline class*

Description

Class representing a flow cytometry pipeline, and composed of two processing queues, i.e. lists of CytoProcessingStep objects :

- a list of CytoProcessingStep(s) for pre-calculation of scale transformations per channel
- a list of CytoProcessingStep(s) for the pre-processing of flow frames

Usage

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

## S4 method for signature 'missing'
CytoPipeline(
  object,
  experimentName = "default_experiment",
  sampleFiles = character(),
  pData = NULL
)

## S4 method for signature 'list'
CytoPipeline(
  object,
  experimentName = "default_experiment",
  sampleFiles = character(),
  pData = NULL
)

## S4 method for signature 'character'
CytoPipeline(
  object,
  experimentName = "default_experiment",
  sampleFiles = character(),
  pData = NULL
)

## S3 method for class 'CytoPipeline'
as.list(x, ...)

experimentName(x)

experimentName(x) <- value

sampleFiles(x)

sampleFiles(x) <- value
```

```
pData(x)

pData(x) <- value

sampleDisplayNames(x, sampleFiles = NULL)

sampleNameFromDisplayName(x, displayName)
```

Arguments

object	a character() containing a JSON input
experimentName	the experiment name
sampleFiles	a character (e.g. sampleFileNames) or a numeric vector (e.g. indices of sample files). If NULL, all samples will be displayed.
pData	the pheno Data (data.frame or NULL)
x	a CytoPipeline object
...	additional arguments (not used here)
value	the new value to be assigned. the pData<- setter is a bit more liberal than it used to be: 1. It can accept new pData containing more rows than existing sample names (the corresponding subset of pData is taken). 2. It can accept pData with row names pointing to either sample file full paths or base file names 3. It can accept pData with no row names provided the number of rows correspond to the number of sample files. Row names are then set by default to sample file base names (if unique), or sample file full paths.
displayName	a character

Value

nothing

- for as.list.CytoPipeline: the obtained list
- for sampleDisplayNames: a character vector of sample display names
- for sampleNameFromDisplayName: the sample name corresponding to the specified display name. of sample display names

Slots

scaleTransformProcessingQueue A list of CytoProcessingStep objects containing the steps for obtaining the scale transformations per channel

flowFramesPreProcessingQueue A list of CytoProcessingStep objects containing the steps for pre-processing of the samples flow frames

experimentName A character containing the experiment (run) name

sampleFiles A character vector storing all fcs files to be run into the pipeline

pData An optional data.frame containing additional information for each sample file. The pData raw names should correspond to the sample files (using full paths or base paths). If the pData contains a columns with name 'displayName', this will have an impact in the sampleDisplayNames() function, i.e. sample display names will be the one mentioned in pData, instead of typically base file names (or larger paths if base file names are not unique)

Examples

```
### *** EXAMPLE 1: building CytoPipeline step by step *** ###

rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
experimentName <- "OMIP021_PeacoQC"
sampleFiles <- file.path(rawDataDir, list.files(rawDataDir,
                                                pattern = "Donor"))

outputDir <- base::tempdir()

# main parameters : sample files and output files
pipL <- CytoPipeline(experimentName = experimentName,
                     sampleFiles = sampleFiles)

### SCALE TRANSFORMATION STEPS ###

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "scale transform",
                      CytoProcessingStep(
                          name = "flowframe_read",
                          FUN = "readSampleFiles",
                          ARGS = list(
                              whichSamples = "all",
                              truncate_max_range = FALSE,
                              min.limit = NULL
                          )
                      )
    )

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "scale transform",
                      CytoProcessingStep(
                          name = "remove_margins",
                          FUN = "removeMarginsPeacoQC",
                          ARGS = list()
                      )
    )

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "scale transform",
                      CytoProcessingStep(
                          name = "compensate",
                          FUN = "compensateFromMatrix",
                          ARGS = list(matrixSource = "fcs")
                      )
    )

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "scale transform",
                      CytoProcessingStep(
                          name = "flowframe_aggregate",
```

```
                            FUN = "aggregateAndSample",
                            ARGS = list(
                                nTotalEvents = 10000,
                                seed = 0
                            )
                        )
    )

pipL <-
    addProcessingStep(pipL,
                        whichQueue = "scale transform",
                        CytoProcessingStep(
                            name = "scale_transform_estimate",
                            FUN = "estimateScaleTransforms",
                            ARGS = list(
                                fluoMethod = "estimateLogicle",
                                scatterMethod = "linear",
                                scatterRefMarker = "BV785 - CD3"
                            )
                        )
    )

### PRE-PROCESSING STEPS ###

pipL <-
    addProcessingStep(pipL,
                        whichQueue = "pre-processing",
                        CytoProcessingStep(
                            name = "flowframe_read",
                            FUN = "readSampleFiles",
                            ARGS = list(
                                truncate_max_range = FALSE,
                                min.limit = NULL
                            )
                        )
    )

pipL <-
    addProcessingStep(pipL,
                        whichQueue = "pre-processing",
                        CytoProcessingStep(
                            name = "remove_margins",
                            FUN = "removeMarginsPeacoQC",
                            ARGS = list()
                        )
    )

pipL <-
    addProcessingStep(pipL,
                        whichQueue = "pre-processing",
                        CytoProcessingStep(
                            name = "compensate",
                            FUN = "compensateFromMatrix",
                            ARGS = list(matrixSource = "fcs")
                        )
    )
```

```
pipL <-
    addProcessingStep(
        pipL,
        whichQueue = "pre-processing",
        CytoProcessingStep(
            name = "remove_debris",
            FUN = "removeDebrisManualGate",
            ARGS = list(
                FSCChannel = "FSC-A",
                SSCChannel = "SSC-A",
                gateData =  c(73615, 110174, 213000, 201000, 126000,
                              47679, 260500, 260500, 113000, 35000)))
    )

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "pre-processing",
                      CytoProcessingStep(
                          name = "remove_dead_cells",
                          FUN = "removeDeadCellsManualGate",
                          ARGS = list(
                              FSCChannel = "FSC-A",
                              LDMarker = "L/D Aqua - Viability",
                              gateData = c(0, 0, 250000, 250000,
                                           0, 650, 650, 0)
                          )
                      )
    )

pipL <-
    addProcessingStep(
        pipL,
        whichQueue = "pre-processing",
        CytoProcessingStep(
            name = "perform_QC",
            FUN = "qualityControlPeacoQC",
            ARGS = list(
                preTransform = TRUE,
                min_cells = 150, # default
                max_bins = 500, # default
                step = 500, # default,
                MAD = 6, # default
                IT_limit = 0.55, # default
                force_IT = 150, # default
                peak_removal = 0.3333, # default
                min_nr_bins_peakdetection = 10 # default
            )
        )
    )

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "pre-processing",
                      CytoProcessingStep(
                          name = "transform",
                          FUN = "applyScaleTransforms",
                          ARGS = list()
```

```
                        )
    )

### *** EXAMPLE 2: building CytoPipeline from JSON file *** ###

jsonDir <- system.file("extdata", package = "CytoPipeline")
jsonPath <- file.path(jsonDir, "pipelineParams.json")

pipL2 <- CytoPipeline(jsonPath,
                      experimentName = experimentName,
                      sampleFiles = sampleFiles)
```

`CytoProcessingStep` *Cyto Processing step*

Description

Class containing the function and arguments to be applied in a lazy-execution framework.

Objects of this class are created using the `CytoProcessingStep()` function. The processing step is executed with the `executeProcessingStep()` function.

Usage

```
CytoProcessingStep(name = character(), FUN = character(), ARGS = list())

## S4 method for signature 'CytoProcessingStep'
show(object)

executeProcessingStep(x, ...)

getCPSName(x)

getCPSFUN(x)

getCPSARGS(x)

## S3 method for class 'CytoProcessingStep'
as.list(x, ...)

as.json.CytoProcessingStep(x, pretty = FALSE)

from.json.CytoProcessingStep(jsonString)
```

Arguments

`name`	`character` denoting a name to the step, which can be different from the function name
`FUN`	`function` or `character` representing a function name.
`ARGS`	`list` of arguments to be passed along to `FUN`.
`object`	a `CytoProcessingStep` object.

x	a CytoProcessingStep object.
...	other arguments (not used)
pretty	formatting set-up (see jsonlite::toJSON doc)
jsonString	a character() containing a JSON string.

Details

This object contains all relevant information of a data analysis processing step, i.e. the function and all of its arguments to be applied to the data.

Value

The CytoProcessingStep function returns and object of type CytoProcessingStep.

Examples

```
## Create a simple processing step object
ps1 <- CytoProcessingStep("summing step", sum)

getCPSName(ps1)

getCPSFUN(ps1)

getCPSARGS(ps1)

executeProcessingStep(ps1, 1:10)

as.list(ps1)

js_str <- as.json.CytoProcessingStep(ps1)

ps2 <- from.json.CytoProcessingStep(js_str)

identical(ps1, ps2)
```

---

estimateScaleTransforms

estimates scale tranformations

Description

this function estimates the scale transformations to be applied on a flowFrame to obtain 'good behaving' distributions, i.e. the best possible separation between + population and - population. It distinguishes between scatter channels, where either linear, or no transform is applied, and fluo channels, where either logicle transform

- using flowCore::estimateLogicle - is estimated, or no transform is applied.

The idea of linear transform of scatter channels is as follows: a reference channel (not a scatter one) is selected and a linear transform (Y = AX + B) is applied to all scatter channel, as to align their 5 and 95 percentiles to those of the reference channel For the estimateLogicle function, see flowCore documentation.

Usage

```
estimateScaleTransforms(
  ff,
  fluoMethod = c("estimateLogicle", "none"),
  scatterMethod = c("none", "linearQuantile"),
  scatterRefMarker = NULL,
  specificScatterChannels = NULL,
  verbose = FALSE
)
```

Arguments

ff	a flowCore::flowFrame
fluoMethod	method to be applied to all fluo channels
scatterMethod	method to be applied to all scatter channels
scatterRefMarker	the reference channel that is used to align the
specificScatterChannels	vector of scatter channels for which we still want to apply the fluo method (and not the scatter Method)
verbose	if TRUE, send messages to the user at each step

Value

a flowCore::flowFrame with removed low quality events from the input

Examples

```
data(OMIP021Samples)

compMatrix <- flowCore::spillover(OMIP021Samples[[1]])$SPILL
ff_c <- runCompensation(OMIP021Samples[[1]], spillover = compMatrix)

transList <-
    estimateScaleTransforms(
        ff = ff_c,
        fluoMethod = "estimateLogicle",
        scatterMethod = "linear",
        scatterRefMarker = "BV785 - CD3")
```

---

execute *executing CytoPipeline object*

Description

this function triggers the execution of the processing queues of a CytoPipeline object. First, the scale tranform processing queue is run, taking the set of sample names as an implicit first input. At the end of the queue, a scale transform List is assumed to be created. Second, the flowFrame pre-processing queue, reapeatedly for each sample file. The scale transform list generated in the previous step is taken as implicit input, together with the initial sample file. At the end of the queue run, a pre-processed flowFrame is assumed to be generated. No change is made on the input CytoPipeline object, all results are stored in the cache.

Usage

```
execute(
  x,
  path = ".",
  rmCache = FALSE,
  useBiocParallel = FALSE,
  BPPARAM = BiocParallel::bpparam(),
  BPOPTIONS = BiocParallel::bpoptions(packages = c("flowCore")),
  saveLastStepFF = TRUE,
  saveFFSuffix = "_preprocessed",
  saveFFFormat = c("fcs", "csv"),
  saveFFCsvUseChannelMarker = TRUE,
  saveScaleTransforms = FALSE
)
```

Arguments

x	CytoPipeline object
path	base path, a subdirectory with name equal to the experiment will be created to store the output data, in particular the experiment cache
rmCache	if TRUE, starts by removing the already existing cache directory corresponding to the experiment
useBiocParallel	if TRUE, use BiocParallel for computation of the sample file pre-processing in parallel (one file per worker at a time). Note the BiocParallel function used is `bplapply()`
BPPARAM	if `useBiocParallel` is TRUE, sets the BPPARAM back-end to be used for the computation. If not provided, will use the top back-end on the `BiocParallel::registered()` stack.
BPOPTIONS	if `useBiocParallel` is TRUE, sets the BPOPTIONS to be passed to `bplapply()` function. Note that if you use a `SnowParams` back-end, you need to specify all the packages that need to be loaded for the different CytoProcessingStep to work properly (visibility of functions). As a minimum, the `flowCore` package needs to be loaded. (hence the default `BPOPTIONS = bpoptions(packages = c("flowCore"))`)
saveLastStepFF	if TRUE, save the final result of the pre-processing, for each file. By convention, these output files are stored in `path/x@experimentName/output/`, the file names used are the same as the initial fcs file basenames, concatenated with `saveFFSuffix`, and with file extension corresponding to `saveFFFormat`.
saveFFSuffix	FF file name suffix
saveFFFormat	either `fcs` or `csv`
saveFFCsvUseChannelMarker	if TRUE (default), converts the channels to the corresponding marker names (where the Marker is not NA). This setting is only applicable to export in csv format.
saveScaleTransforms	if TRUE (default FALSE), save on disk (in RDS format) the `flowCore::transformList` object obtained after running the scaleTransform processing queue. The file name is hardcoded to `path/experimentName/RDS/scaleTransformList.rds`

Value

nothing

Examples

```
### *** EXAMPLE 1: building CytoPipeline step by step *** ###

rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
experimentName <- "OMIP021_PeacoQC"
sampleFiles <- file.path(rawDataDir, list.files(rawDataDir,
                                                pattern = "Donor"))

outputDir <- base::tempdir()

# main parameters : sample files and output files
pipelineParams <- list()
pipelineParams$experimentName <- experimentName
pipelineParams$sampleFiles <- sampleFiles
pipL <- CytoPipeline(pipelineParams)

### SCALE TRANSFORMATION STEPS ###

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "scale transform",
                      CytoProcessingStep(
                          name = "flowframe_read",
                          FUN = "readSampleFiles",
                          ARGS = list(
                              whichSamples = "all",
                              truncate_max_range = FALSE,
                              min.limit = NULL
                          )
                      )
    )

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "scale transform",
                      CytoProcessingStep(
                          name = "remove_margins",
                          FUN = "removeMarginsPeacoQC",
                          ARGS = list()
                      )
    )

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "scale transform",
                      CytoProcessingStep(
                          name = "compensate",
                          FUN = "compensateFromMatrix",
                          ARGS = list(matrixSource = "fcs")
                      )
    )
```

```
pipL <-
    addProcessingStep(pipL,
                      whichQueue = "scale transform",
                      CytoProcessingStep(
                          name = "flowframe_aggregate",
                          FUN = "aggregateAndSample",
                          ARGS = list(
                              nTotalEvents = 10000,
                              seed = 0
                          )
                      )
    )

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "scale transform",
                      CytoProcessingStep(
                          name = "scale_transform_estimate",
                          FUN = "estimateScaleTransforms",
                          ARGS = list(
                              fluoMethod = "estimateLogicle",
                              scatterMethod = "linear",
                              scatterRefMarker = "BV785 - CD3"
                          )
                      )
    )

### PRE-PROCESSING STEPS ###

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "pre-processing",
                      CytoProcessingStep(
                          name = "flowframe_read",
                          FUN = "readSampleFiles",
                          ARGS = list(
                              truncate_max_range = FALSE,
                              min.limit = NULL
                          )
                      )
    )

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "pre-processing",
                      CytoProcessingStep(
                          name = "remove_margins",
                          FUN = "removeMarginsPeacoQC",
                          ARGS = list()
                      )
    )

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "pre-processing",
                      CytoProcessingStep(
```

```
                        name = "compensate",
                        FUN = "compensateFromMatrix",
                        ARGS = list(matrixSource = "fcs")
                    )
    )

pipL <-
addProcessingStep(
    pipL,
    whichQueue = "pre-processing",
    CytoProcessingStep(
        name = "remove_debris",
        FUN = "removeDebrisManualGate",
        ARGS = list(
            FSCChannel = "FSC-A",
            SSCChannel = "SSC-A",
            gateData =  c(73615, 110174, 213000, 201000, 126000,
                        47679, 260500, 260500, 113000, 35000)
                )
   )
)

pipL <-
    addProcessingStep(pipL,
                      whichQueue = "pre-processing",
                      CytoProcessingStep(
                          name = "remove_dead_cells",
                          FUN = "removeDeadCellsManualGate",
                          ARGS = list(
                              FSCChannel = "FSC-A",
                              LDMarker = "L/D Aqua - Viability",
                              gateData = c(0, 0, 250000, 250000,
                                           0, 650, 650, 0)
                          )
                      )
    )

pipL <-
    addProcessingStep(
        pipL,
        whichQueue = "pre-processing",
        CytoProcessingStep(
            name = "perform_QC",
            FUN = "qualityControlPeacoQC",
            ARGS = list(
                preTransform = TRUE,
                min_cells = 150, # default
                max_bins = 500, # default
                step = 500, # default,
                MAD = 6, # default
                IT_limit = 0.55, # default
                force_IT = 150, # default
                peak_removal = 0.3333, # default
                min_nr_bins_peakdetection = 10 # default
            )
        )
    )
```

```
pipL <-
    addProcessingStep(pipL,
                      whichQueue = "pre-processing",
                      CytoProcessingStep(
                          name = "transform",
                          FUN = "applyScaleTransforms",
                          ARGS = list()
                      )
    )

# execute pipeline, remove cache if existing with the same experiment name
suppressWarnings(execute(pipL, rmCache = TRUE, path = outputDir))

# re-execute as is without removing cache => all results found in cache!
suppressWarnings(execute(pipL, rmCache = FALSE, path = outputDir))

### *** EXAMPLE 2: building CytoPipeline from JSON file *** ###

jsonDir <- system.file("extdata", package = "CytoPipeline")
jsonPath <- file.path(jsonDir, "pipelineParams.json")

pipL2 <- CytoPipeline(jsonPath,
                      experimentName = experimentName,
                      sampleFiles = sampleFiles)

# note we temporarily set working directory into package root directory
# needed as json path mentions "./" path for sample files
suppressWarnings(execute(pipL2, rmCache = TRUE, path = outputDir))

### *** EXAMPLE 3: building CytoPipeline from cache (previously run) *** ###

experimentName <- "OMIP021_PeacoQC"
pipL3 <- buildCytoPipelineFromCache(
    experimentName = experimentName,
    path = outputDir)

suppressWarnings(execute(pipL3,
        rmCache = FALSE,
        path = outputDir))
```

`exportCytoPipeline` *exporting CytoPipeline objects*

Description

functions to export CytoPipeline objects in various formats

Usage

```
export2JSONFile(x, path)
```

Arguments

x	a CytoPipeline object
path	the full path to the name of the file to be created

Value

- for `export2JSONFile`: nothing

Functions

- `export2JSONFile()`: exports a CytoPipeline object to a JSON file (writing the file = side effect)

Examples

```
outputDir <- base::tempdir()

rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
experimentName <- "OMIP021_PeacoQC"
sampleFiles <- file.path(rawDataDir, list.files(rawDataDir,
                                                pattern = "Donor"))

# build CytoPipeline object using json input
jsonPath <- file.path(system.file("extdata", package = "CytoPipeline"),
                      "pipelineParams.json")

pipL <- CytoPipeline(jsonPath,
                     experimentName = experimentName,
                     sampleFiles = sampleFiles)

# remove the last pre-processing step
nPreProcessing <- getNbProcessingSteps(pipL, whichQueue = "pre-processing")
pipL <- removeProcessingStep(pipL, whichQueue = "pre-processing",
                                   index = nPreProcessing)

# export back to json file
export2JSONFile(pipL, path = file.path(outputDir, "newFile.json"))
```

`findTimeChannel` *find time channel in flowSet/flowFrame*

Description

tries to find a channel in a flowSet/flowFrame that could be the time channel. First tries to identify a channel name containing the 'time' string, then tries to identify a single monotonically increasing channel.

Usage

```
findTimeChannel(obj, excludeChannels = c())
```

Arguments

obj	a flowCore::flowFrame or flowCore::flowSet
excludeChannels	vector of column names to exclude in the search

Value

a character, name of the found channel that should be representing time. If not found, returns NULL.

Examples

```
data(OMIP021Samples)

ret <- findTimeChannel(OMIP021Samples[[1]])
ret # "Time"
```

---

getAcquiredCompensationMatrix

extract compensation matrix from a flowCore::flowFrame

Description

helper function retrieving the compensation matrix stored in fcs file (if any). It scans the following keywords: $SPILL, $spillover and $SPILLOVER

Usage

```
getAcquiredCompensationMatrix(ff)
```

Arguments

ff	a flowCore::flowFrame

Value

the found compensation matrix

Examples

```
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
sampleFiles <-
    file.path(rawDataDir, list.files(rawDataDir, pattern = "Donor"))

truncateMaxRange <- FALSE
minLimit <- NULL

# create flowCore::flowSet with all samples of a dataset
fsRaw <- readSampleFiles(
    sampleFiles = sampleFiles,
    whichSamples = "all",
```

```
    truncate_max_range = truncateMaxRange,
    min.limit = minLimit)
compensationMatrix <- getAcquiredCompensationMatrix(fsRaw[[2]])
```

`getChannelNamesFromMarkers` *get channel names from markers*

Description

finds name of channels corresponding to user provided markers

Usage

```
getChannelNamesFromMarkers(ff, markers)
```

Arguments

`ff`	a flowCore::flowFrame
`markers`	a vector of markers, either provided as : • an array of booleans (referring to flowFrame columns) • an array of integers (indices in flowFrame columns) • an array of characters (exact markers or channel patterns)

Value

a character vector, containing the names of the corresponding channels

Examples

```
data(OMIP021Samples)

# with existing markers
ret <- getChannelNamesFromMarkers(
    OMIP021Samples[[1]],
    c(
        "FSC-A",
        "L/D Aqua - Viability",
        "FITC - gdTCR",
        "PECy5 - CD28"
    ))

ret # c("FSC-A", "525/50Violet-A", "530/30Blue-A", "670/30Yellow-A")

# with boolean vector
indices <- c(1, 6, 14, 18)
boolInput <- rep(FALSE, 21)
boolInput[indices] <- TRUE
ret2 <- getChannelNamesFromMarkers(
    OMIP021Samples[[1]],
    boolInput)

ret2 # c("FSC-A", "525/50Violet-A", "530/30Blue-A", "670/30Yellow-A")
```

```
# with indices vector
ret3 <- getChannelNamesFromMarkers(
    OMIP021Samples[[1]],
    indices
)
ret3 # c("FSC-A", "525/50Violet-A", "530/30Blue-A", "670/30Yellow-A")
```

`getFCSFileName` *get fcs file name*

Description

get basename of $FILENAME keyword if exists

Usage

```
getFCSFileName(ff)
```

Arguments

`ff`	a flowCore::flowFrame

Value

the basename of $FILENAME keyword

Examples

```
data(OMIP021Samples)

fName <- getFCSFileName(OMIP021Samples[[1]])
```

`getTransfoParams` *get tranformation parameters for a specific channel*

Description

investigates a flowCore::tranformList object to get the type and parameters of the transformation applying to a specific channel

Usage

```
getTransfoParams(transList, channel)
```

Arguments

`transList`	a flowCore::transformList
`channel`	channel name

Value

If the transformation exists for the specified channel, and is either recognized as a logicle transfo or a linear transfo, a list with two slots:

- $type a character containing the transfo type ('logicle' or 'linear')
- $params_list a list of named numeric, according to transfo type

Otherwise, NULL is returned.

Examples

```
data(OMIP021Samples)

# set-up a hybrid transformation list :
# - two channels are logicle-ly transformed with automatic param estimates
# - one channel has explicit logicle transfo with default parameters
# - one channel has linear transformation
# - other channels have no transformation
translist <- flowCore::estimateLogicle(
    OMIP021Samples[[1]],
    c("450/50Violet-A", "525/50Violet-A")
)
translist <- c(
    translist,
    flowCore::transformList(
        "FSC-A",
        flowCore::linearTransform(
            a = 0.1,
            b = 0
       )
    ),
    flowCore::transformList(
        "540/30Violet-A",
        flowCore::logicleTransform()
    )
)

ret1 <- getTransfoParams(translist, channel = "FSC-A")
ret1$type # "linear"
ret1$paramsList # a = 0.1, b = 0.

ret2 <- getTransfoParams(translist, channel = "525/50Violet-A")
ret2$type # "logicle"
ret2$paramsList # a = 0., w = 0.2834, m = 4.5, t = 262143

ret3 <- getTransfoParams(translist, channel = "540/30Violet-A")
ret3$type # "logicle
ret3$paramsList # a = 0., w = 0.5, m = 4.5, t = 262144
```

---

ggplotEvents *plot events in 1D or 2D, using ggplot2*

Description

plot events of specific channels of either : flowCore::flowFrame, or flowCore::flowSet in 2D or 1D, mimicking FlowJo type of graph.
if 1D : geom_density will be used
if 2D : geom_hex will be used

Usage

```
ggplotEvents(
  obj,
  xChannel,
  yChannel = NULL,
  nDisplayCells = Inf,
  seed = NULL,
  bins = 216,
  fill = "lightblue",
  alpha = 0.2,
  xScale = c("linear", "logicle"),
  yScale = c("linear", "logicle"),
  xLogicleParams = NULL,
  yLogicleParams = NULL,
  xLinearRange = NULL,
  yLinearRange = NULL,
  transList = NULL,
  runTransforms = FALSE
)
```

Arguments

obj	a flowCore::flowFrame or flowCore::flowSet
xChannel	channel (name or index) or marker name to be displayed on x axis
yChannel	channel (name or index) or marker name to be displayed on y axis
nDisplayCells	maximum number of events that will be plotted. If the number of events exceed this number, a sub-sampling will be performed
seed	seed used for sub-sampling (if any)
bins	used in geom_hex
fill	used in geom_density
alpha	used in geom_density
xScale	scale to be used for the x axis (note "linear" corresponds to no transformation)
yScale	scale to be used for the y axis (note "linear" corresponds to no transformation)
xLogicleParams	if (xScale == "logicle"), the parameters of the logicle transformation to be used, as a list(w = ..., m = ..., a = ..., t = ...). If NULL, these parameters will be estimated by flowCore::estimateLogicle()
yLogicleParams	if (yScale == "logicle"), the parameters of the logicle transformation to be used, as a list(w = ..., m = ..., a = ..., t = ...). If NULL, these parameters will be estimated by flowCore::estimateLogicle()
xLinearRange	if (xScale == "linear"), the x axis range to be used

yLinearRange	if (yScale == "linear"), the y axis range to be used
transList	optional list of scale transformations to be applied to each channel. If it is non null, 'x/yScale', 'x/yLogicleParams' and 'x/yLinear_range' will be discarded.
runTransforms	(TRUE/FALSE) Will the application of non linear scale result in data being effectively transformed ? • If TRUE, than the data will undergo transformations prior to visualization. • If FALSE, the axis will be scaled but the data themselves will not be transformed.

Value

a list of ggplot objects

Examples

```
data(OMIP021Samples)

### 1D Examples ###

# simple linear scale example
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "FSC-A",
             xScale = "linear")

# with explicit linear range
ggplotEvents(OMIP021Samples[[1]],
                  xChannel = "FSC-A",
                  xScale = "linear",
                  xLinearRange = c(0, 250000))

# with linear scale, several flow frames
ggplotEvents(OMIP021Samples, xChannel = "FSC-A", xScale = "linear")

# simple logicle scale example
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "450/50Violet-A",
             xScale = "logicle")

# logicle scale, explicit parameters
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "450/50Violet-A",
             xScale = "logicle", xLogicleParams = list(
                 a = 1,
                 w = 2,
                 m = 7,
                 t = 270000))

# with sub-sampling
ggplotEvents(OMIP021Samples[[2]],
             xChannel = "450/50Violet-A",
             xScale = "logicle", nDisplayCells = 5000)

# tuning some plot parameters
ggplotEvents(OMIP021Samples[[2]],
             xChannel = "450/50Violet-A",
```

```
            xScale = "logicle", alpha = 0.5, fill = "red")

# examples that use a transformation list, estimated after compensation
compensationMatrix <- flowCore::spillover(OMIP021Samples[[1]])$SPILL

ffC <- runCompensation(OMIP021Samples[[1]],
                       spillover = compensationMatrix,
                       updateChannelNames = FALSE)

transList <- flowCore::estimateLogicle(
    ffC,
    colnames(compensationMatrix))

transList <-
    c(transList,
      flowCore::transformList(
          "FSC-A",
          flowCore::linearTransform(a = 0.00001)))

# linear example, without running the transformations on data
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "450/50Violet-A",
             xScale = "linear",
             transList = transList,
             runTransforms = FALSE)

# linear example, now running the transformations on data
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "450/50Violet-A",
             xScale = "linear",
             transList = transList,
             runTransforms = TRUE)

# logicle example, without running the transformations on data
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "FSC-A",
             xScale = "logicle",
             transList = transList,
             runTransforms = FALSE)

# logicle example, now running the transformations on data
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "FSC-A",
             xScale = "logicle",
             transList = transList,
             runTransforms = TRUE)

### 2D examples ###


# simple linear example
ggplotEvents(OMIP021Samples[[1]],
                  xChannel = "FSC-A",
                  xScale = "linear",
                  yChannel = "610/20Violet-A",
                  yScale = "logicle")
```

```
# simple linear example, 2 flow frames
ggplotEvents(OMIP021Samples,
             xChannel = "FSC-A",
             xScale = "linear",
             yChannel = "SSC-A",
             yScale = "linear")

# logicle vs linear example
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "450/50Violet-A",
             xScale = "logicle",
             yChannel = "SSC-A",
             yScale = "linear")

# 2X logicle example
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "TETaGC",
             xScale = "logicle",
             yChannel = "CD27",
             yScale = "logicle")

# tuning nb of bins
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "TETaGC",
             xScale = "logicle",
             yChannel = "CD27",
             yScale = "logicle",
             bins = 128)

# using transformation list, not run on data
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "TETaGC",
             xScale = "logicle",
             yChannel = "CD27",
             yScale = "logicle",
             transList = transList,
             runTransforms = FALSE)

# using transformation list, run on data
ggplotEvents(OMIP021Samples[[1]],
             xChannel = "TETaGC",
             xScale = "logicle",
             yChannel = "CD27",
             yScale = "logicle",
             transList = transList,
             runTransforms = TRUE)
```

`ggplotFilterEvents` *plot filtered events in 2D, using ggplot*

Description

plot events of specific channels of either : flowCore::flowFrame, or flowCore::flowSet in 2D, showing the impact of applying a filter between :

- a 'pre' flowframe

Usage

```
ggplotFilterEvents(
  ffPre,
  ffPost,
  xChannel,
  yChannel,
  nDisplayCells = 10000,
  seed = NULL,
  size = 0.5,
  xScale = c("linear", "logicle"),
  yScale = c("linear", "logicle"),
  xLogicleParams = NULL,
  yLogicleParams = NULL,
  xLinearRange = NULL,
  yLinearRange = NULL,
  transList = NULL,
  runTransforms = FALSE,
  interactive = FALSE
)
```

Arguments

ffPre	a flowCore::flowFrame, before applying filter
ffPost	a flowCore::flowFrame, after applying filter
xChannel	channel (name or index) or marker name to be displayed on x axis
yChannel	channel (name or index) or marker name to be displayed on y axis
nDisplayCells	maximum number of events that will be plotted. If the number of events exceed this number, a subsampling will be performed
seed	seed used for sub-sampling (if any)
size	used by geom_point()
xScale	scale to be used for the x axis (note "linear" corresponds to no transformation)
yScale	scale to be used for the y axis (note "linear" corresponds to no transformation)
xLogicleParams	if (xScale == "logicle"), the parameters of the logicle transformation to be used, as a list(w = ..., m = ..., a = ..., t = ...) If NULL, these parameters will be estimated by flowCore::estimateLogicle()
yLogicleParams	if (yScale == "logicle"), the parameters of the logicle transformation to be used, as a list(w = ..., m = ..., a = ..., t = ...) If NULL, these parameters will be estimated by flowCore::estimateLogicle()
xLinearRange	if (xScale == "linear"), linear range to be used
yLinearRange	if (yScale == "linear"), linear range to be used
transList	optional list of scale transformations to be applied to each channel. If it is non null, 'x/yScale', 'x/yLogicleParams' and 'x/yLinear_range' will be discarded.
runTransforms	(TRUE/FALSE) Will the application of non linear scale result in data being effectively transformed ? • If TRUE, than the data will undergo transformations prior to visualization.

- If FALSE, the axis will be scaled but the data themselves are not transformed.

`interactive` if TRUE, transform the scaling formats such that the ggcyto::x_scale_logicle() and ggcyto::y_scale_logicle() do work with plotly::ggplotly()

Value

a ggplot object

Examples

```
data(OMIP021Samples)

ffPre <- OMIP021Samples[[1]]

# creating a manual polygon gate filter based on channels L/D and FSC-A

LDMarker <- "L/D Aqua - Viability"

LDChannel <- getChannelNamesFromMarkers(ffPre, markers = LDMarker)
liveGateMatrix <- matrix(
    data = c(
        50000, 50000, 100000, 200000, 200000,
        100, 1000, 2000, 2000, 1
    ),
    ncol = 2,
    dimnames = list(
        c(),
        c("FSC-A", LDChannel)
    )
)

liveGate <- flowCore::polygonGate(
    filterId = "Live",
    .gate = liveGateMatrix
)

selectedLive <- flowCore::filter(ffPre, liveGate)
ffL <- flowCore::Subset(ffPre, selectedLive)


# show the results

# subsample 5000 points
ggplotFilterEvents(
    ffPre = ffPre,
    ffPost = ffL,
    nDisplayCells = 5000,
    xChannel = "FSC-A", xScale = "linear",
    yChannel = LDMarker, yScale = "logicle") +
    ggplot2::ggtitle("Live gate filter - 5000 points")

# with all points
ggplotFilterEvents(
    ffPre = ffPre,
    ffPost = ffL,
```

```
    nDisplayCells = Inf,
    xChannel = "FSC-A", xScale = "linear",
    yChannel = LDMarker, yScale = "logicle") +
    ggplot2::ggtitle("Live gate filter - all points")
```

`ggplotFlowRate` *plot flow rate as a function of time, using ggplot2*

Description

plot flow rate as a function of time, using ggplot2

Usage

```
ggplotFlowRate(obj, title = "Flow Rate", timeUnit = 100)
```

Arguments

`obj`	a flowCore::flowFrame or flowCore::flowSet
`title`	a title for the graph
`timeUnit`	which time interval is used to calculate "instant" flow rate (default = 100 ms)

Value

a ggplot graph

Examples

```
data(OMIP021Samples)

# single flowFrame plot
ggplotFlowRate(OMIP021Samples[[1]])

# two flowFrames plot
ggplotFlowRate(OMIP021Samples)

# single plot with title
ggplotFlowRate(OMIP021Samples[[1]], title = "Test Flow Rate plot")

# explicit time unit
ggplotFlowRate(OMIP021Samples[[1]], timeUnit = 50)
```

handlingProcessingSteps
handling processing steps in CytoPipeline objects

Description

functions to manipulate processing steps in processing queues of CytoPipeline objects

Usage

```
addProcessingStep(
  x,
  whichQueue = c("scale transform", "pre-processing"),
  newPS
)

removeProcessingStep(
  x,
  whichQueue = c("scale transform", "pre-processing"),
  index
)

getNbProcessingSteps(x, whichQueue = c("scale transform", "pre-processing"))

getProcessingStep(
  x,
  whichQueue = c("scale transform", "pre-processing"),
  index
)

getProcessingStepNames(x, whichQueue = c("scale transform", "pre-processing"))

cleanProcessingSteps(
  x,
  whichQueue = c("both", "scale transform", "pre-processing")
)

showProcessingSteps(x, whichQueue = c("scale transform", "pre-processing"))
```

Arguments

x	a CytoPipeline object
whichQueue	selects the processing queue for which we manage the processing steps
newPS	the new processing step to be added (CytoProcessingStep object)
index	index of the processing step to remove

Value

- for addProcessingStep: the updated CytoPipeline object
- for removeProcessingStep: the updated CytoPipeline object

- for `getNbProcessingSteps`: the number of processing steps present in the target queue
- for `getProcessingStep`: the obtained CytoProcessingStep object
- for `getProcessingStepNames`: the vector of step names
- for `cleanProcessingSteps`: the updated CytoPipeline object
- for `showProcessingSteps`: nothing (only console display side effect is required)

Functions

- `addProcessingStep()`: adds a processing step in one of the processing queues (at the end), returns the modified CytoPipeline object
- `removeProcessingStep()`: removes a processing step from one of the processing queues, returns the modified CytoPipeline object
- `getNbProcessingSteps()`: gets the number of processing steps in a processing queue
- `getProcessingStep()`: gets a processing step at a specific index of a processing queue
- `getProcessingStepNames()`: gets a character vector of all processing step names of a specific processing queue
- `cleanProcessingSteps()`: deletes all processing steps in one or both processing queues, returns the modified CytoPipeline object
- `showProcessingSteps()`: shows all processing steps in a processing queue

Examples

```
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
experimentName <- "OMIP021_PeacoQC"
sampleFiles <- file.path(rawDataDir, list.files(rawDataDir,
                                                pattern = "Donor"))
transListPath <-
    file.path(system.file("extdata", package = "CytoPipeline"),
              "OMIP021_TransList.rds")

# main parameters : sample files and experiment name
pipelineParams <- list()
pipelineParams$experimentName <- experimentName
pipelineParams$sampleFiles <- sampleFiles

# create CytoPipeline object (no step defined yet)
pipL <- CytoPipeline(pipelineParams)

# add a processing step in scale tranformation queue
pipL <- addProcessingStep(pipL,
                          whichQueue = "scale transform",
                          CytoProcessingStep(
                              name = "scale_transform_read",
                              FUN = "readRDS",
                              ARGS = list(file = transListPath)
                          ))

getNbProcessingSteps(pipL, "scale transform") # returns 1
```

```
# add another processing step in scale transformation queue
pipL <- addProcessingStep(pipL,
                          whichQueue = "scale transform",
                          CytoProcessingStep(
                              name = "scale_transform_sum",
                              FUN = "sum",
                              ARGS = list()
                          )
)

getNbProcessingSteps(pipL, "scale transform") # returns 2

getProcessingStepNames(pipL, whichQueue = "scale transform")

# removes second processing step in scale transformation queue
pipL <- removeProcessingStep(pipL,
                             whichQueue = "scale transform",
                             index = 2)

# get processing step object
pS <- getProcessingStep(pipL, whichQueue = "scale transform", index = 1)
getCPSName(pS) #"scale_transform_read"

# add a processing step in pre-processing queue
pipL <- addProcessingStep(pipL,
                          whichQueue = "pre-processing",
                          CytoProcessingStep(
                              name = "pre-processing_sum",
                              FUN = "sum",
                              ARGS = list()
                          ))
getNbProcessingSteps(pipL, "scale transform") # returns 1
getNbProcessingSteps(pipL, "pre-processing") # returns also 1

showProcessingSteps(pipL, whichQueue = "scale transform")
showProcessingSteps(pipL, whichQueue = "pre-processing")

# cleans both processing queues
pipL <- cleanProcessingSteps(pipL)
pipL
```

`inspectCytoPipelineObjects` *inspect CytoPipeline results objects*

Description

functions to obtain results objects formats

Usage

```
getCytoPipelineExperimentNames(
  path = ".",
  pattern = NULL,
```

```
  ignore.case = FALSE,
  fixed = FALSE
)

getCytoPipelineObjectFromCache(
  x,
  path = ".",
  whichQueue = c("scale transform", "pre-processing"),
  sampleFile = NULL,
  objectName
)

getCytoPipelineObjectInfos(
  x,
  path = ".",
  whichQueue = c("scale transform", "pre-processing"),
  sampleFile = NULL
)

getCytoPipelineFlowFrame(
  x,
  path = ".",
  whichQueue = c("scale transform", "pre-processing"),
  sampleFile,
  objectName
)

getCytoPipelineScaleTransform(
  x,
  path = ".",
  whichQueue = c("scale transform", "pre-processing"),
  sampleFile = NULL,
  objectName
)

plotCytoPipelineProcessingQueue(
  x,
  whichQueue = c("pre-processing", "scale transform"),
  purpose = c("run status", "description"),
  sampleFile = NULL,
  path = ".",
  title = TRUE,
  box.type = "ellipse",
  lwd = 1,
  box.prop = 0.5,
  box.cex = 0.7,
  cex.txt = 0.7,
  box.size = 0.1,
  dtext = 0.15,
  ...
)
```

```
collectNbOfRetainedEvents(experimentName, path = ".", whichSampleFiles)
```

Arguments

path	root path to locate the search for file caches
pattern	optional pattern limiting the search for experiment names
ignore.case	(TRUE/FALSE) used in pattern matching (grepl)
fixed	(TRUE/FALSE) used in pattern matching (grepl)
x	a CytoPipeline object
whichQueue	which queue to look into
sampleFile	which sampleFile is looked for: • if whichQueue == "scale transform", the sampleFile is ignored • if NULL and whichQueue == "pre-processing", the sampleFile is defaulted to the first one belonging to the experiment
objectName	(character) which object name to look for
purpose	purpose of the workflow plot • if "run status" (default), the disk cache will be inspected and the box colours will be set according to run status (green = run, orange = not run, red = definition not consistent with cache). Moreover, the object classes and names will be filled in if found in the cache. • if "description", the workflow will be obtained from the step definition in the x object, not from the disk cache. As a result, all boxes will be coloured in black, and no object class and name will be provided.
title	if TRUE, adds a title to the plot
box.type	shape of label box (rect, ellipse, diamond, round, hexa, multi)
lwd	default line width of arrow and box (one numeric value)
box.prop	length/width ratio of label box (one numeric value)
box.cex	relative size of text in boxes (one numeric value)
cex.txt	relative size of arrow text (one numeric value)
box.size	size of label box (one numeric value)
dtext	controls the position of arrow text relative to arrowhead (one numeric value)
...	other arguments passed to diagram::plotmat()
experimentName	the experimentName used to select the file cache on disk
whichSampleFiles	indicates for which sample files the number of retained events are to be collected. If missing, all sample files will be used.

Value

- for `getCytoPipelineExperimentNames`: a vector of character containing found experiment names
- for `getCytoPipelineObjectFromCache`: the found object (or stops with an error message if the target object is not found)
- for `getCytoPipelineObjectInfos`: a dataframe with the collected information about the found objects (or stops with an error message if no target object was found)

- for `getCytoPipelineFlowFrame`: the found flowFrame (or stops with an error message if the target object is not found, or if the object is no flowFrame)
- for `getCytoPipelineScaleTransform`: the found flowFrame (or stops with an error message if the target object is not found, or if the object is no transformList)
- for `plotCytoPipelineProcessingQueue`: nothing
- for `collectNbOfRetainedEvents`: a dataframe with the collected number of events columns refer to pre-processing steps rows refer to samples

Functions

- `getCytoPipelineExperimentNames()`: This function looks into a path for stored file caches and gets the corresponding experiment names
- `getCytoPipelineObjectFromCache()`: Given a CytoPipeline object, this function retrieves a specific object in the corresponding file cache
- `getCytoPipelineObjectInfos()`: Given a CytoPipeline object, this function retrieves the information related to a specific object name, i.e. object name and object class
- `getCytoPipelineFlowFrame()`: Given a CytoPipeline object, this function retrieves a specific flowCore::flowFrame object in the corresponding file cache object name and object class
- `getCytoPipelineScaleTransform()`: Given a CytoPipeline object, this function retrieves a specific flowCore::transformList object in the corresponding file cache
- `plotCytoPipelineProcessingQueue()`: This functions displays a plot of a processing queue of a CytoPipeline object, using diagram::plotmat().
  - If a step is in run state for all sample files, the corresponding box appears in green
  - If a step is in non run state for at least one sample file, the corresponding box appears in orange
  - If at least one step is not consistent with cache, the whole set of boxes appears in red
- `collectNbOfRetainedEvents()`: Given a CytoPipeline object, this function retrieves, for all pre-processing steps, given the output is a flowFrame, the number of retained event.

Examples

```
# preliminary run:
# build CytoPipeline object using json input, run and store results in cache
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
experimentName <- "OMIP021_PeacoQC"
sampleFiles <- file.path(rawDataDir, list.files(rawDataDir,
                                                pattern = "Donor"))

jsonDir <- system.file("extdata", package = "CytoPipeline")
jsonPath <- file.path(jsonDir, "pipelineParams.json")
outputDir <- base::tempdir()
pipL <- CytoPipeline(jsonPath,
                     experimentName = experimentName,
                     sampleFiles = sampleFiles)

# note we temporarily set working directory into package root directory
# needed as json path mentions "./" path for sample files
suppressWarnings(execute(pipL, rmCache = TRUE, path = outputDir))
```

```
# get a list of all stored experiments in a specific path taken as root dir
experimentNames <- getCytoPipelineExperimentNames(path = outputDir)

# rebuilding Cytopipeline object from cache
pipL2 <- buildCytoPipelineFromCache(experimentName = experimentNames[1],
                                    path = outputDir)

# plot scale transformation queue
plotCytoPipelineProcessingQueue(pipL2, whichQueue = "pre-processing",
                                path = outputDir)

# plot pre-processing queue
plotCytoPipelineProcessingQueue(pipL2, whichQueue = "scale transform",
                                path = outputDir)

# get object infos for a specific queue
df <- getCytoPipelineObjectInfos(pipL2, whichQueue = "pre-processing",
                                 path = outputDir,
                                 sampleFile = sampleFiles(pipL2)[1])

# get transform list (output of one step)
trans <-
    getCytoPipelineScaleTransform(pipL2, whichQueue = "scale transform",
                                  objectName =
                                      "scale_transform_estimate_obj",
                                  path = outputDir)

# get flowFrame (output of one step)
ff <- getCytoPipelineFlowFrame(pipL2, whichQueue = "pre-processing",
                               objectName = "remove_doublets_obj",
                               path = outputDir,
                               sampleFile = sampleFiles(pipL2)[1])

# get any object (output of one step)
obj <-
    getCytoPipelineObjectFromCache(pipL2, whichQueue = "scale transform",
                                   objectName = "compensate_obj",
                                   path = outputDir)
class(obj) # flowCore::flowSet

# collect number of retained events at each step
nbEventsDF <- collectNbOfRetainedEvents(
        experimentName = experimentNames[1],
        path = outputDir)
```

`interactingWithCytoPipelineCache` *interaction between CytoPipeline object and disk cache*

Description

functions supporting the interaction between a CytoPipeline object and the file cache on disk

Usage

```
deleteCytoPipelineCache(x, path = ".")

buildCytoPipelineFromCache(experimentName, path = ".")

checkCytoPipelineConsistencyWithCache(
  x,
  path = ".",
  whichQueue = c("both", "scale transform", "pre-processing"),
  sampleFile = NULL
)
```

Arguments

x	a CytoPipeline object
path	the full path to the experiment storage on disk (without the /.cache)
experimentName	the experimentName used to select the file cache on disk
whichQueue	which processing queue to check the consistency of
sampleFile	if whichQueue == "pre-processing" or "both": which sample file(s) to check on the disk cache

Value

for deleteCytoPipelineCache: TRUE if successfully removed
for buildCytoPipelineFromCache: the built CytoPipeline object
for checkCytoPipelineConsistencyWithCache: a list with the following values:

- isConsistent (TRUE/FALSE)
- inconsistencyMsg: character filled in by an inconsistency message in case the cache and CytoPipeline object are not consistent with each other
- scaleTransformStepStatus: a character vector, containing, for each scale transform step, a status from c("run", "not run", "inconsistent")
- preProcessingStepStatus: a character matrix, containing, for each pre-processing step (rows), for each sample file (columns), a status from c("run", "not run", "inconsistent")

Functions

- deleteCytoPipelineCache(): delete the whole disk cache corresponding to the experiment of a CytoPipeline object
- buildCytoPipelineFromCache(): builds a new CytoPipeline object, based on the information stored in the file cache
- checkCytoPipelineConsistencyWithCache(): check the consistency between the processing steps described in a CytoPipeline object, and what is stored in the file cache

Examples

```
# preliminary run:
# build CytoPipeline object using json input, run and store results in cache
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
experimentName <- "OMIP021_PeacoQC"
```

```
sampleFiles <- file.path(rawDataDir, list.files(rawDataDir,
                                                pattern = "Donor"))

jsonDir <- system.file("extdata", package = "CytoPipeline")
jsonPath <- file.path(jsonDir, "pipelineParams.json")
outputDir <- base::tempdir()
pipL <- CytoPipeline(jsonPath,
                     experimentName = experimentName,
                     sampleFiles = sampleFiles)

# note we temporarily set working directory into package root directory
# needed as json path mentions "./" path for sample files
suppressWarnings(execute(pipL, rmCache = TRUE, path = outputDir))


# rebuild CytoPipeline from stored results in cache, for a specific
# experiment

experimentName <- "OMIP021_PeacoQC"
pipL2 <- buildCytoPipelineFromCache(
    experimentName = experimentName,
    path = outputDir)


# checking consistency between CytoPipeline object and cache
res <- checkCytoPipelineConsistencyWithCache(pipL2)
#res

suppressWarnings(execute(pipL2, rmCache = FALSE, path = outputDir))
# (everything is already stored in cache)

# deleting cache related to a specific experiment
pipL3 <- CytoPipeline(experimentName = experimentName)
deleteCytoPipelineCache(pipL3, path = outputDir)
```

`OMIP021Samples` *OMIP021Samples dataset*

Description

OMIP021Samples dataset

Format

a flowCore::flowSet with two different flowFrames each one contains one flow cytometry sample corresponding to Donor1.fcs and Donor2.fcs in following source. A subsampling of 5,000 events has been performed on each file.

Value

nothing

Source

https://flowrepository.org/experiments/305

`qualityControlFlowAI` *perform QC with flowAI*

Description

this function is a wrapper around flowAI::flow_auto_qc() function. It also pre-selects the channels to be handled (=> all signal channels)

Usage

```
qualityControlFlowAI(
  ff,
  preTransform = FALSE,
  transList = NULL,
  outputDiagnostic = FALSE,
  outputDir = NULL,
  ...
)
```

Arguments

`ff`	a flowCore::flowFrame
`preTransform`	if TRUE, apply the transList scale transform prior to running the gating algorithm
`transList`	applied in conjunction with preTransform
`outputDiagnostic`	if TRUE, stores diagnostic files generated by flowAI in outputDir directory
`outputDir`	used in conjunction with outputDiagnostic
`...`	additional parameters passed to flowAI::flow_auto_qc()

Value

a flowCore::flowFrame with removed low quality events from the input

Examples

```
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
sampleFiles <-
    file.path(rawDataDir, list.files(rawDataDir, pattern = "Donor"))

truncateMaxRange <- FALSE
minLimit <- NULL

# create flowCore::flowSet with all samples of a dataset
fsRaw <- readSampleFiles(
    sampleFiles = sampleFiles,
    whichSamples = "all",
```

```
    truncate_max_range = truncateMaxRange,
    min.limit = minLimit)

suppressWarnings(ff_QualityControl <-
    qualityControlFlowAI(fsRaw[[2]],
                         remove_from = "all", # all default
                         second_fractionFR = 0.1,
                         deviationFR = "MAD",
                         alphaFR = 0.01,
                         decompFR = TRUE,
                         outlier_binsFS = FALSE,
                         pen_valueFS = 500,
                         max_cptFS = 3,
                         sideFM = "both",
                         neg_valuesFM = 1))
```

`qualityControlPeacoQC` *perform QC with PeacoQC*

Description

this function is a wrapper around PeacoQC::PeacoQC() function. It also pre-selects the channels to be handled (=> all signal channels)

Usage

```
qualityControlPeacoQC(
  ff,
  preTransform = FALSE,
  transList = NULL,
  outputDiagnostic = FALSE,
  outputDir = NULL,
  ...
)
```

Arguments

`ff`	a flowCore::flowFrame
`preTransform`	if TRUE, apply the transList scale transform prior to running the gating algorithm
`transList`	applied in conjunction with preTransform
`outputDiagnostic`	if TRUE, stores diagnostic files generated by PeacoQC in outputDir directory
`outputDir`	used in conjunction with outputDiagnostic
`...`	additional parameters passed to PeacoQC::PeacoQC()

Value

a flowCore::flowFrame with removed low quality events from the input

Examples

```
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
sampleFiles <-
    file.path(rawDataDir, list.files(rawDataDir, pattern = "Donor"))

truncateMaxRange <- FALSE
minLimit <- NULL

# create flowCore::flowSet with all samples of a dataset
fsRaw <- readSampleFiles(
    sampleFiles = sampleFiles,
    whichSamples = "all",
    truncate_max_range = truncateMaxRange,
    min.limit = minLimit)

suppressWarnings(ff_m <- removeMarginsPeacoQC(x = fsRaw[[2]]))

ff_c <-
    compensateFromMatrix(ff_m,
                         matrixSource = "fcs")

transList <-
    estimateScaleTransforms(
        ff = ff_c,
        fluoMethod = "estimateLogicle",
        scatterMethod = "linear",
        scatterRefMarker = "BV785 - CD3")


ff_QualityControl <- suppressWarnings(
    qualityControlPeacoQC(
        ff_c,
        preTransform = TRUE,
        transList = transList,
        min_cells = 150,
        max_bins = 500,
        MAD = 6,
        IT_limit = 0.55,
        force_IT = 150,
        peak_removal = (1/3),
        min_nr_bins_peakdetection = 10))
```

`readRDSObject` *read RDS object*

Description

wrapper around readRDS, which discards any additional parameters passed in (...)

Usage

```
readRDSObject(RDSFile, ...)
```

Arguments

RDSFile	a RDS file containing a R object object
...	other arguments (not used)

Value

the read R object

Examples

```
data(OMIP021Samples)

transListPath <- file.path(system.file("extdata",
                                       package = "CytoPipeline"),
                           "OMIP021_TransList.rds")

transList <- readRDSObject(transListPath)

ff_c <- compensateFromMatrix(OMIP021Samples[[1]],
                             matrixSource = "fcs")

ff_t <- applyScaleTransforms(ff_c, transList = transList)
```

---

readSampleFiles *Read fcs sample files*

Description

Wrapper around flowCore::read.fcs() or flowCore::read.flowSet(). Also adds a "Cell_ID" additional column, used in flowFrames comparison

Usage

```
readSampleFiles(
  sampleFiles,
  whichSamples = "all",
  nSamples = NULL,
  seed = NULL,
  channelMarkerFile = NULL,
  ...
)
```

Arguments

sampleFiles	a vector of character path to sample files
whichSamples	one of: • 'all' if all sample files need to be read • 'random' if some samples need to be chosen randomly (in that case, using nSamples and seed) • a vector of indexes pointing to the sampleFiles vector

nSamples	number of samples to randomly select (if whichSamples == "random"). If nSamples is higher than nb of available samples, the output will be all samples
seed	an optional seed parameters (provided to ease reproducibility).
channelMarkerFile	an optional path to a csv file which provides the mapping between channels and markers. If provided, this csv file should contain a Channel column, and a Marker column. Optionally a 'Used' column can be provided as well (TRUE/FALSE). Channels for which the 'Used' column is set to FALSE will not be incorporated in the created flowFrame.
...	additional parameters passed to flowCore file reading functions.

Value

either a flowCore::flowSet or a flowCore::flowFrame if length(sampleFiles) == 1

Examples

```
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
sampleFiles <-
    file.path(rawDataDir, list.files(rawDataDir, pattern = "Donor"))

truncateMaxRange <- FALSE
minLimit <- NULL

# create flowCore::flowSet with all samples of a dataset
res <- readSampleFiles(
    sampleFiles = sampleFiles,
    whichSamples = "all",
    truncate_max_range = truncateMaxRange,
    min.limit = minLimit)

#res

# create a flowCore::flowFrame with one single sample
res2 <- readSampleFiles(
    sampleFiles = sampleFiles,
    whichSamples = 2,
    truncate_max_range = truncateMaxRange,
    min.limit = minLimit)

#res2
```

---

removeChannels *remove channels from a flowFrame*

Description

: in a flowCore::flowFrame, remove the channels of the given names.

Usage

```
removeChannels(ff, channels)
```

Arguments

ff	a flowCore::flowFrame
channels	the channel names to be removed

Value

a new flowCore::flowFrame with the removed channels

Examples

```
data(OMIP021Samples)

retFF <- removeChannels(OMIP021Samples[[1]],
                        channel = "FSC-A")
```

---

removeDeadCellsManualGate

remove dead cells from a flowFrame using manual gating

Description

remove dead cells from a flowFrame, using manual gating in the FSC-A, '(a)Live/Dead' 2D representation. The function uses flowCore::polygonGate()

Usage

```
removeDeadCellsManualGate(
  ff,
  preTransform = FALSE,
  transList = NULL,
  FSCChannel,
  LDMarker,
  gateData,
  ...
)
```

Arguments

ff	a flowCore::flowFrame
preTransform	boolean, if TRUE: the transList list of scale transforms will be applied first on the LD channel.
transList	applied in conjunction with preTransform == TRUE
FSCChannel	a character containing the exact name of the forward scatter channel
LDMarker	a character containing the exact name of the marker corresponding to (a)Live/Dead channel, or the Live/Dead channel name itself
gateData	a numerical vector containing the polygon gate coordinates first the FSCChannel channel coordinates of each points of the polygon gate, then the LD channel coordinates of each points (prior to scale transform)
...	additional parameters passed to flowCore::polygonGate()

Value

a flowCore::flowFrame with removed dead cells from the input

Examples

```
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
sampleFiles <-
    file.path(rawDataDir, list.files(rawDataDir, pattern = "Donor"))

truncateMaxRange <- FALSE
minLimit <- NULL

# create flowCore::flowSet with all samples of a dataset
fsRaw <- readSampleFiles(
    sampleFiles = sampleFiles,
    whichSamples = "all",
    truncate_max_range = truncateMaxRange,
    min.limit = minLimit)

suppressWarnings(ff_m <- removeMarginsPeacoQC(x = fsRaw[[2]]))

ff_c <-
    compensateFromMatrix(ff_m,
                         matrixSource = "fcs")

remDeadCellsGateData <- c(0, 0, 250000, 250000,
                          0, 650, 650, 0)

ff_lcells <-
    removeDeadCellsManualGate(ff_c,
                              FSCChannel = "FSC-A",
                              LDMarker = "L/D Aqua - Viability",
                              gateData = remDeadCellsGateData)
```

`removeDebrisManualGate` *remove debris from a flowFrame using manual gating*

Description

remove debris from a flowFrame, using manual gating in the FSC-A, SSC-A 2D representation. The function internally uses flowCore::polygonGate()

Usage

```
removeDebrisManualGate(ff, FSCChannel, SSCChannel, gateData, ...)
```

Arguments

`ff`	a flowCore::flowFrame
`FSCChannel`	a character containing the exact name of the forward scatter channel

SSCChannel	a character containing the exact name of the side scatter channel
gateData	a numerical vector containing the polygon gate coordinates first the FSCChannel channel coordinates of each points of the polygon gate, then the SSCChannel channel coordinates of each points.
...	additional parameters passed to flowCore::polygonGate()

Value

a flowCore::flowFrame with removed debris events from the input

Examples

```
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
sampleFiles <-
    file.path(rawDataDir, list.files(rawDataDir, pattern = "Donor"))

truncateMaxRange <- FALSE
minLimit <- NULL

# create flowCore::flowSet with all samples of a dataset
fsRaw <- readSampleFiles(
    sampleFiles = sampleFiles,
    whichSamples = "all",
    truncate_max_range = truncateMaxRange,
    min.limit = minLimit)

suppressWarnings(ff_m <- removeMarginsPeacoQC(x = fsRaw[[2]]))

ff_c <-
    compensateFromMatrix(ff_m,
                         matrixSource = "fcs")


remDebrisGateData <- c(73615, 110174, 213000, 201000, 126000,
                       47679, 260500, 260500, 113000, 35000)

ff_cells <-
    removeDebrisManualGate(ff_c,
                           FSCChannel = "FSC-A",
                           SSCChannel = "SSC-A",
                           gateData = remDebrisGateData)
```

---

removeDoubletsCytoPipeline

remove doublets from a flowFrame, using CytoPipeline custom algorithm

---

Description

Wrapper around CytoPipeline::singletGate(). Can apply the flowStats function subsequently on several channel pairs, e.g. (FSC-A, FSC-H) and (SSC-A, SSC-H)

Usage

```
removeDoubletsCytoPipeline(ff, areaChannels, heightChannels, nmads, ...)
```

Arguments

ff	a flowCore::flowFrame
areaChannels	a character vector containing the name of the 'area type' channels one wants to use
heightChannels	a character vector containing the name of the 'height type' channels one wants to use
nmads	a numeric vector with the bandwidth above the ratio allowed, per channels pair (cells are kept if the ratio between -A channel[i] and -H channel[i] is smaller than the median ratio + nmad[i] times the median absolute deviation of the ratios). Default is 4, for all channel pairs.
...	additional parameters passed to CytoPipeline::singletGate()

Value

a flowCore::flowFrame with removed doublets events from the input

Examples

```
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
sampleFiles <-
    file.path(rawDataDir, list.files(rawDataDir, pattern = "Donor"))

truncateMaxRange <- FALSE
minLimit <- NULL

# create flowCore::flowSet with all samples of a dataset
fsRaw <- readSampleFiles(
    sampleFiles = sampleFiles,
    whichSamples = "all",
    truncate_max_range = truncateMaxRange,
    min.limit = minLimit)

suppressWarnings(ff_m <- removeMarginsPeacoQC(x = fsRaw[[2]]))

ff_c <-
    compensateFromMatrix(ff_m,
                         matrixSource = "fcs")

ff_s <-
    removeDoubletsCytoPipeline(ff_c,
                               areaChannels = c("FSC-A", "SSC-A"),
                               heightChannels = c("FSC-H", "SSC-H"),
                               nmads = c(3, 5))
```

`removeMarginsPeacoQC` *remove margin events using PeacoQC*

Description

Wrapper around PeacoQC::RemoveMargins(). Also pre-selects the channels to be handled (=> all signal channels) If input is a flowSet, it applies removeMargins() to each flowFrame of the flowSet.

Usage

```
removeMarginsPeacoQC(x, channelSpecifications = NULL, ...)
```

Arguments

`x`	a flowCore::flowSet or a flowCore::flowFrame
`channelSpecifications`	A list of lists with parameter specifications for certain channels. This parameter should only be used if the values in the internal parameters description is too strict or wrong for a number or all channels. This should be one list per channel with first a minRange and then a maxRange value. This list should have the channel name found back in colnames(flowCore::exprs(ff)), or the corresponding marker name (found in flowCore::pData(flowCore::description(ff))) . If a channel is not listed in this parameter, its default internal values will be used. The default of this parameter is NULL. If the name of one list is set to `AllFluoChannels`, then the `minRange` and `maxRange` specified there will be taken as default for all fluorescent channels (not scatter)
`...`	additional parameters passed to PeacoQC::RemoveMargins()

Value

either a flowCore::flowSet or a flowCore::flowFrame depending on the input.

Examples

```
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
sampleFiles <-
    file.path(rawDataDir, list.files(rawDataDir, pattern = "Donor"))

truncateMaxRange <- FALSE
minLimit <- NULL
fsRaw <- readSampleFiles(sampleFiles,
                         truncate_max_range = truncateMaxRange,
                         min.limit = minLimit)
suppressWarnings(ff_m <- removeMarginsPeacoQC(x = fsRaw[[2]]))
ggplotFilterEvents(ffPre = fsRaw[[2]],
                   ffPost = ff_m,
                   xChannel = "FSC-A",
                   yChannel = "SSC-A")
```

resetCellIDs *reset 'Original_ID' column in a flowframe*

Description

: on a flowCore::flowFrame, reset 'Original_ID' column. This column can be used in plots comparing the events pre and post gating. If the 'Original_ID' column already exists, the function replaces the existing IDs by the user provided ones. If not, an `appendCellID()` is called.

Usage

```
resetCellIDs(ff, eventIDs = seq_len(flowCore::nrow(ff)))
```

Arguments

`ff`	a flowCore::flowFrame
`eventIDs`	an integer vector containing the values to be set in expression matrix, as Original ID's.

Value

new flowCore::flowFrame containing the amended (or added) 'Original_ID' column

Examples

```
data(OMIP021Samples)

ff <- appendCellID(OMIP021Samples[[1]])

subsample_ff <- subsample(ff, 100, keepOriginalCellIDs = TRUE)

# re-create a sequence of IDs, ignoring the ones before subsampling
reset_ff <- resetCellIDs(subsample_ff)
```

runCompensation *compensate with additional options*

Description

: this is a simple wrapper around the flowCore::compensate() utility, allowing to trigger an update of the fluo channel names with a prefix 'comp-' (as in FlowJo)

Usage

```
runCompensation(obj, spillover, updateChannelNames = TRUE)
```

Arguments

obj	a flowCore::flowFrame or flowCore::flowSet
spillover	compensation object or spillover matrix or a list of compensation objects
updateChannelNames	if TRUE, add a 'comp-' prefix to all fluorochrome channels (hence does not impact the columns related to FSC, SSC, or other specific keyword like TIME, Original_ID, File,...) Default TRUE.

Value

a new object with compensated data, and possibly updated column names

Examples

```
data(OMIP021Samples)

ff <- OMIP021Samples[[1]]
compMatrix <- flowCore::spillover(ff)$SPILL
ff <- runCompensation(ff,
                      spillover = compMatrix,
                      updateChannelNames = TRUE)
```

---

singletsGate *Clean doublet events from flow cytometry data*

Description

will adjust a polygon gate aimed at cleaning doublet events from the flowFrame. The main idea is to use the ratio between the two indicated channel as an indicator and select only the events for which this ratio is 'not too far' from the median ratio. More specifically, the computed ratio is ch1/(1+ch2). However, instead of looking at a constant range of this ratio, as is done in PeacoQC::removeDoublets(), which leads to a semi-conic gate, we apply a parallelogram shaped gate, by keeping a constant range of channel 2 intensity, based on the target ratio range at the mid value of channel 1.

Usage

```
singletsGate(
  ff,
  filterId = "Singlets",
  channel1 = "FSC-A",
  channel2 = "FSC-H",
  nmad = 4,
  verbose = FALSE
)
```

Arguments

ff	A flowCore::flowframe that contains flow cytometry data.
filterId	the name for the filter that is returned
channel1	The first channel that will be used to determine the doublet events. Default is "FSC-A"
channel2	The second channels that will be used to determine the doublet events. Default is "FSC-H"
nmad	Bandwidth above the ratio allowed (cells are kept if their ratio is smaller than the median ratio + nmad times the median absolute deviation of the ratios). Default is 4.
verbose	If set to TRUE, the median ratio and width will be printed. Default is FALSE.

Value

This function returns a flowCore::polygonGate.

Examples

```
data(OMIP021Samples)

# simple example with one single singlets gate filter
# FSC-A and FSC-H channels are used by default

mySingletsGate <- singletsGate(OMIP021Samples[[1]], nmad = 3)

selectedSinglets <- flowCore::filter(
    OMIP021Samples[[1]],
    mySingletsGate)

ff_l <- flowCore::Subset(OMIP021Samples[[1]], selectedSinglets)

linRange <- c(0, 250000)

ggplotFilterEvents(
    ffPre = OMIP021Samples[[1]],
    ffPost = ff_l,
    xChannel = "FSC-A", xLinearRange = linRange,
    yChannel = "FSC-H", yLinearRange = linRange)

# application of two singlets gates one after the other

singletsGate1 <- singletsGate(OMIP021Samples[[1]], nmad = 3)
singletsGate2 <- singletsGate(OMIP021Samples[[1]],
                              channel1 = "SSC-A",
                              channel2 = "SSC-H",
                              filterId = "Singlets2")

singletCombinedGate <- singletsGate1 & singletsGate2

selectedSinglets <- flowCore::filter(
    OMIP021Samples[[1]],
    singletCombinedGate)

ff_l <- flowCore::Subset(OMIP021Samples[[1]], selectedSinglets)
```

```
ggplotFilterEvents(
    ffPre = OMIP021Samples[[1]],
    ffPost = ff_l,
    xChannel = "FSC-A", xLinearRange = linRange,
    yChannel = "FSC-H", yLinearRange = linRange)

ggplotFilterEvents(
    ffPre = OMIP021Samples[[1]],
    ffPost = ff_l,
    xChannel = "SSC-A", xLinearRange = linRange,
    yChannel = "SSC-H", yLinearRange = linRange)
```

`subsample` *sub-sampling of a flowFrame*

Description

: sub-samples a flowFrame with the specified number of samples, without replacement. adds also a column 'Original_ID' if not already present in flowFrame.

Usage

```
subsample(ff, nEvents, seed = NULL, keepOriginalCellIDs = TRUE, ...)
```

Arguments

`ff`	a flowCore::flowFrame
`nEvents`	number of events to be obtained using sub-sampling
`seed`	can be set for reproducibility of event sub-sampling
`keepOriginalCellIDs`	if TRUE, adds (if not already present) a 'OriginalID' column containing the initial IDs of the cell (from 1 to nrow prior to subsampling). if FALSE, does the same, but takes as IDs (1 to nrow after subsampling)
`...`	additional parameters (currently not used)

Value

new flowCore::flowFrame with the obtained subset of samples

Examples

```
data(OMIP021Samples)

# take first sample of dataset, subsample 100 events and create new flowFrame
ff <- subsample(OMIP021Samples[[1]], nEvents = 100)
```

updateMarkerName *update marker name of a given flowFrame channel*

Description

: in a flowCore::flowFrame, update the marker name (stored in 'desc' of parameters data) of a given channel. Also update the corresponding keyword in the flowFrame.

Usage

```
updateMarkerName(ff, channel, newMarkerName)
```

Arguments

ff	a flowCore::flowFrame
channel	the channel for which to update the marker name
newMarkerName	the new marker name to be given to the selected channel

Value

a new flowCore::flowFrame with the updated marker name

Examples

```
data(OMIP021Samples)

retFF <- updateMarkerName(OMIP021Samples[[1]],
                          channel = "FSC-A",
                          newMarkerName = "Fwd Scatter-A")
```

writeFlowFrame *write flowFrame to disk*

Description

wrapper around flowCore::write.FCS() or utils::write.csv that discards any additional parameter passed in (...)

Usage

```
writeFlowFrame(
  ff,
  dir = ".",
  useFCSFileName = TRUE,
  prefix = "",
  suffix = "",
  format = c("fcs", "csv"),
  csvUseChannelMarker = TRUE,
  ...
)
```

Arguments

ff	a flowCore::flowFrame
dir	an existing directory to store the flowFrame,
useFCSFileName	if TRUE filename used will be based on original fcs filename
prefix	file name prefix
suffix	file name suffix
format	either fcs or csv
csvUseChannelMarker	if TRUE (default), converts the channels to the corresponding marker names (where the Marker is not NA). This setting is only applicable to export in csv format.
...	other arguments (not used)

Value

nothing

Examples

```
rawDataDir <-
    system.file("extdata", package = "CytoPipeline")
sampleFiles <-
    file.path(rawDataDir, list.files(rawDataDir, pattern = "Donor"))

truncateMaxRange <- FALSE
minLimit <- NULL

# create flowCore::flowSet with all samples of a dataset
res <- readSampleFiles(
    sampleFiles = sampleFiles,
    whichSamples = "all",
    truncate_max_range = truncateMaxRange,
    min.limit = minLimit)

ff_c <- compensateFromMatrix(res[[2]], matrixSource = "fcs")
outputDir <- base::tempdir()
writeFlowFrame(ff_c,
               dir = outputDir,
               suffix = "_fcs_export",
               format = "csv")
```

Index