

# Package ‘CytoML’

April 14, 2017

**Type** Package

**Title** GatingML interface for openCyto

**Version** 1.0.1

**Date** 2016-04-15

**Author** Mike Jiang

**Maintainer** Mike Jiang <wjjiang2@fhcrc.org>

**Description** This package is designed to use GatingML2.0 as the standard format to exchange the gated data with other software platform.

**License** Artistic-2.0

**LazyData** TRUE

**Imports** flowCore, flowWorkspace (>= 3.19.24), openCyto (>= 1.11.3), XML, data.table, flowUtils (>= 1.35.7), jsonlite, RBGL, ncdfFlow, Rgraphviz, Biobase, methods, graph, graphics, utils, base64enc

**biocViews** FlowCytometry, DataImport, DataRepresentation

**Suggests** testthat, flowWorkspaceData, knitr, ggcyto

**VignetteBuilder** knitr

**RoxygenNote** 5.0.1

**Collate** 'GatingSet2cytobank.R' 'GatingSet2flowJo.R'  
'cytobank2GatingSet.R' 'flowUtils\_functions.R'  
'read.gatingML.cytobank.R' 'graphGML\_methods.R' 'utils.R'

**NeedsCompilation** no

## R topics documented:

addCustomInfo . . . . .	2
compare.counts . . . . .	3
compensate,GatingSet,graphGML-method . . . . .	3
constructTree . . . . .	4
cytobank2GatingSet . . . . .	4
extend . . . . .	5
gating,graphGML,GatingSet-method . . . . .	6
GatingSet2cytobank . . . . .	7
GatingSet2flowJo . . . . .	8
getChildren,graphGML,character-method . . . . .	8

getCompensationMatrices,graphGML-method . . . . .	9
getGate,graphGML,character-method . . . . .	9
getNodes,graphGML-method . . . . .	10
getParent,graphGML,character-method . . . . .	11
getTransformations,graphGML-method . . . . .	11
graphGML-class . . . . .	12
matchPath . . . . .	12
parse.gateInfo . . . . .	13
plot,graphGML,missing-method . . . . .	13
read.gatingML.cytobank . . . . .	14
show,graphGML-method . . . . .	14

<b>Index</b>	<b>16</b>
--------------	-----------

---

addCustomInfo	<i>add customInfo nodes to each gate node and add BooleanAndGates</i>
---------------	-----------------------------------------------------------------------

---

## Description

add customInfo nodes to each gate node and add BooleanAndGates

## Usage

```
addCustomInfo(root, gs, flowEnv, cytobank.default.scale = TRUE, showHidden)
```

## Arguments

root	the root node of the XML
gs	a GatingSet object
flowEnv	the environment that stores the information parsed by 'read.GatingML'.
cytobank.default.scale	logical flag indicating whether to use the default Cytobank asinhtGml2 settings. Currently it should be set to TRUE in order for gates to be displayed properly in Cytobank because cytobank currently does not parse the global scale settings from GatingML.
showHidden	whether to include the hidden population nodes in the output

## Value

XML root node

---

compare.counts	<i>compare the counts to cytobank's exported csv so that the parsing result can be verified.</i>
----------------	--------------------------------------------------------------------------------------------------

---

**Description**

compare the counts to cytobank's exported csv so that the parsing result can be verified.

**Usage**

```
compare.counts(gs, file, id.vars = c("FCS Filename", "population"))
```

**Arguments**

gs	parsed GatingSet
file	the stats file (contains the populatio counts) exported from cytobank.
id.vars	either "population" or "FCS filename" that tells whether the stats file format is one population per row or FCS file per row.

**Value**

a data.table (in long format) that contains the counts from openCyto and Cytobank side by side.

**Examples**

```
xmlfile <- system.file("extdata/cytotrol_tcell_cytobank.xml", package = "CytoML")
fcsFiles <- list.files(pattern = "CytoTrol", system.file("extdata", package = "flowWorkspaceData"), full = TRUE)
gs <- cytobank2GatingSet(xmlfile, fcsFiles)
## verify the stats are correct
statsfile <- system.file("extdata/cytotrol_tcell_cytobank_counts.csv", package = "CytoML")
dt_merged <- compare.counts(gs, statsfile, id.vars = "population")
all.equal(dt_merged[, count.x], dt_merged[, count.y], tol = 5e-4)
```

---

compensate, GatingSet, graphGML-method

*compensate a GatingSet based on the compensation information stored in graphGML object*

---

**Description**

compensate a GatingSet based on the compensation information stored in graphGML object

**Usage**

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

**Arguments**

x	GatingSet
spillover	graphGML
...	unused.

**Value**

compensated GatingSet

---

constructTree	<i>Reconstruct the population tree from the GateSets</i>
---------------	----------------------------------------------------------

---

**Description**

Reconstruct the population tree from the GateSets

**Usage**

```
constructTree(flowEnv, gateInfo)
```

**Arguments**

flowEnv	the environment contains the elements parsed by read.gatingML function
gateInfo	the data.frame contains the gate name, fcs filename parsed by parse.gateInfo function

**Value**

a graphNEL represent the population tree. The gate and population name are stored as nodeData in each node.

---

cytobank2GatingSet	<i>A wrapper that parse the gatingML and FCS files into GatingSet</i>
--------------------	-----------------------------------------------------------------------

---

**Description**

A wrapper that parse the gatingML and FCS files into GatingSet

**Usage**

```
cytobank2GatingSet(xml, FCS)
```

**Arguments**

xml	the full path of gatingML file
FCS	FCS files to be loaded

**Value**

a GatingSet

**Examples**

```
xmlfile <- system.file("extdata/cytotrol_tcell_cytobank.xml", package = "CytoML")
fcsFiles <- list.files(pattern = "CytoTrol", system.file("extdata", package = "flowWorkspaceData"), full = TRUE)
gs <- cytobank2GatingSet(xmlfile, fcsFiles)
#plotGate(gs[[1]])
```

---

extend	<i>extend the gate to the minimum and maximum limit of both dimensions based on the bounding information.</i>
--------	---------------------------------------------------------------------------------------------------------------

---

**Description**

It is equivalent to the behavior of shifting the off-scale boundary events into the gate boundary that is described in bounding transformation section of gatingML standard.

**Usage**

```
extend(gate, bound, data.range = NULL, plot = FALSE,
       limits = c("original", "extended"))
```

```
## S3 method for class 'polygonGate'
extend(gate, bound, data.range = NULL, plot = FALSE,
       limits = c("original", "extended"))
```

```
## S3 method for class 'rectangleGate'
extend(gate, ...)
```

```
## S3 method for class 'ellipsoidGate'
extend(gate, ...)
```

**Arguments**

gate	a flowCore filter/gate
bound	numeric matrix representing the bounding information parsed from gatingML. Each row corresponds to a channel. rownames should be the channel names. colnames should be c("min", "max")
data.range	numeric matrix specifying the data limits of each channel. It is used to set the extended value of vertices and must have the same structure as 'bound'. when it is not supplied, c(-.Machine\$integer.max, -.Machine\$integer.max) is used.
plot	whether to plot the extended polygon.
limits	character whether to plot in "extended" or "original" gate limits. Default is "original".
...	other arguments

**Details**

The advantage of extending gates instead of shifting data are two folds: 1. Avoid the extra computation each time applying or plotting the gates 2. Avoid changing the data distribution caused by adding the gates

Normally this function is not used directly by user but invoked when parsing GatingML file exported from Cytobank.

**Value**

a flowCore filter/gate

**Examples**

```
library(flowCore)
sqrcut <- matrix(c(300,300,600,600,50,300,300,50),ncol=2,nrow=4)
colnames(sqrcut) <- c("FSC-H","SSC-H")
pg <- polygonGate(filterId="nonDebris", sqrcut)
pg
bound <- matrix(c(100,3e3,100,3e3), byrow = TRUE, nrow = 2, dimnames = list(c("FSC-H", "SSC-H"), c("min", "max")))
pg.extened <- extend(pg, bound, plot = TRUE)
```

---

gating,graphGML,GatingSet-method

*Apply the gatingML graph to a GatingSet*

---

**Description**

It applies the gates to the GatingSet based on the population tree described in graphGML.

**Usage**

```
## S4 method for signature 'graphGML,GatingSet'
gating(x, y, ...)
```

**Arguments**

x	graphGML
y	GatingSet
...	other arguments

**Value**

Nothing. As the side effect, gates generated by gating methods are saved in GatingSet.

---

GatingSet2cytobank      *Convert a GatingSet to a Cytobank-compatible gatingML*

---

### Description

this function retrieves the gates from GatingSet and writes a customized GatingML-2.0 file that can be imported into cytobank.

### Usage

```
GatingSet2cytobank(gs, outFile, showHidden = FALSE,  
  cytobank.default.scale = TRUE)
```

### Arguments

gs	a GatingSet object
outFile	a file name
showHidden	whether to include the hidden population nodes in the output
cytobank.default.scale	logical flag indicating whether to use the default Cytobank asinhtGml2 settings. Currently it should be set to TRUE in order for gates to be displayed properly in Cytobank because cytobank currently does not parse the global scale settings from GatingML.

### Details

The process can be divided into four steps: 1. Read in gate geometry, compensation and transformation from gatingSet 2. Rescale gate boundaries with flowJoTrans() so gates can be displayed properly in Cytobank 3. Save gates and hierarchy structure to R environment 4. Write environment out to gatingML using write.GatingML()

### Value

nothing

### Examples

```
library(flowWorkspace)  
  
dataDir <- system.file("extdata", package="flowWorkspaceData")  
gs <- load_gs(list.files(dataDir, pattern = "gs_manual", full = TRUE))  
  
Rm("CD8", gs)  
  
#output to cytobank  
outFile <- tempfile(fileext = ".xml")  
GatingSet2cytobank(gs, outFile) #type by default is 'cytobank'
```

---

GatingSet2flowJo      *Convert a GatingSet to flowJo workspace*

---

**Description**

Convert a GatingSet to flowJo workspace

**Usage**

```
GatingSet2flowJo(gs, outFile, ...)
```

**Arguments**

gs	a GatingSet object
outFile	the workspace file path to write
...	other arguments showHidden whether to include the hidden population nodes in the output

**Value**

nothing

**Examples**

```
library(flowWorkspace)

dataDir <- system.file("extdata",package="flowWorkspaceData")
gs <- load_gs(list.files(dataDir, pattern = "gs_manual",full = TRUE))

#output to flowJo
outFile <- tempfile(fileext = ".wsp")
GatingSet2flowJo(gs, outFile)
```

---

getChildren,graphGML,character-method  
*get children nodes*

---

**Description**

get children nodes

**Usage**

```
## S4 method for signature 'graphGML,character'
getChildren(obj, y)
```



**Arguments**

obj                    graphGML  
y                        character parent node path

**Value**

a graphNEL node

**Examples**

```
xmlfile <- system.file("extdata/cytotrol_tcell_cytobank.xml", package = "CytoML")  
g <- read.gatingML.cytobank(xmlfile)  
getChildren(g, "GateSet_722326")  
getParent(g, "GateSet_722326")
```

---

*getCompensationMatrices,graphGML-method*  
*Extract compensation from graphGML object.*

---

**Description**

Extract compensation from graphGML object.

**Usage**

```
## S4 method for signature 'graphGML'  
getCompensationMatrices(x)
```

**Arguments**

x                        graphGML

**Value**

compensation object or "FCS" when compensation comes from FCS keywords

---

*getGate,graphGML,character-method*  
*get gate from the node*

---

**Description**

get gate from the node

**Usage**

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

**Arguments**

obj	graphGML
y	character node path

**Value**

the gate information associated with the node

---

getNodes,graphGML-method

*get nodes from graphGML object*

---

**Description**

get nodes from graphGML object

**Usage**

```
## S4 method for signature 'graphGML'
getNodes(x, y, order = c("default", "bfs", "dfs",
  "tsort"), only.names = TRUE)
```

**Arguments**

x	graphGML
y	character node index. When missing, return all the nodes
order	character specifying the order of nodes. options are "default", "bfs", "dfs", "tsort"
only.names	logical specifying whether user wants to get the entire nodeData or just the name of the population node

**Value**

It returns the node names and population names by default. Or return the entire nodeData associated with each node.

**Examples**

```
xmlfile <- system.file("extdata/cytotrol_tcell_cytobank.xml", package = "CytoML")
g <- read.gatingML.cytobank(xmlfile)
getNodes(g)
getNodes(g, only.names = FALSE)
```

---

*getParent,graphGML,character-method*  
*get parent nodes*

---

### **Description**

get parent nodes

### **Usage**

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

### **Arguments**

obj	graphGML
y	character child node path

### **Value**

a graphNEL node

---

*getTransformations,graphGML-method*  
*Extract transformations from graphGML object.*

---

### **Description**

Extract transformations from graphGML object.

### **Usage**

```
## S4 method for signature 'graphGML'  
getTransformations(x)
```

### **Arguments**

x	graphGML
---	----------

### **Value**

transformerList object

---

graphGML-class	<i>A graph object returned by 'read.gatingML.cytobank' function.</i>
----------------	----------------------------------------------------------------------

---

### Description

Each node corresponds to a population(or GateSet) defined in gatingML file. The actual gate object (both global and tailored gates) is associated with each node as nodeData. Compensation and transformations are stored in graphData slot.

### Details

The class simply extends the graphNEL class and exists for the purpose of method dispatching.

---

matchPath	<i>Given the leaf node, try to find out if a collection of nodes can be matched to a path in a graph(tree) by the bottom-up searching</i>
-----------	-------------------------------------------------------------------------------------------------------------------------------------------

---

### Description

Given the leaf node, try to find out if a collection of nodes can be matched to a path in a graph(tree) by the bottom-up searching

### Usage

```
matchPath(g, leaf, nodeSet)
```

### Arguments

g	graphNEL
leaf	the name of leaf(terminal) node
nodeSet	a set of node names

### Value

TRUE if path is found, FALSE if not path is matched.

---

parse.gateInfo	<i>Parse the cytobank custom_info for each gate</i>
----------------	-----------------------------------------------------

---

**Description**

Fcs filename and gate name stored in 'custom\_info' element are beyond the scope of the gatingML standard and thus not covered by the default 'read.gatingML'.

**Usage**

```
parse.gateInfo(file, ...)
```

**Arguments**

file	xml file path
...	additional arguments passed to the handlers of 'xmlTreeParse'

**Value**

a data.frame that contains three columns: id (gateId), name (gate name), fcs (fcs\_file\_filename).

---

plot, graphGML, missing-method	<i>plot the population tree stored in graphGML.</i>
--------------------------------	-----------------------------------------------------

---

**Description**

The node with dotted order represents the population that has tailored gates (sample-specific gates) defined.

**Usage**

```
## S4 method for signature 'graphGML,missing'
plot(x, y = "missing", label = c("popName",
  "gateName"))
```

**Arguments**

x	a graphNEL generated by constructTree function
y	not used
label	specifies what to be displayed as node label. Can be either 'popName' (population name parsed from GateSets) or 'gateName' (the name of the actual gate associated with each node)

**Value**

nothing

**Examples**

```
xmlfile <- system.file("extdata/cytotrol_tcell_cytobank.xml", package = "CytoML")
g <- read.gatingML.cytobank(xmlfile)
plot(g)
```

---

```
read.gatingML.cytobank
```

*Parser for gatingML exported by Cytobank*

---

**Description**

The Default parser (flowUtils::read.gatingML) does not parse the population tree as well as the custom information from cytobank. (e.g. gate name, fcs filename).

**Usage**

```
read.gatingML.cytobank(file, ...)
```

**Arguments**

file	Gating-ML XML file
...	additional arguments passed to the handlers of 'xmlTreeParse'

**Value**

a graphGML that represents the population tree. The gate and population name are stored in node-Data of each node. Compensation and transformations are stored in graphData.

**Examples**

```
xml <- system.file("extdata/cytotrol_tcell_cytobank.xml", package = "CytoML")
g <- read.gatingML.cytobank(xml) #parse the population tree
#plot(g) #visualize it
```

---

```
show,graphGML-method  show method for graphGML
```

---

**Description**

show method for graphGML

**Usage**

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

**Arguments**

object	graphGML
--------	----------

**Value**

nothing

# Index

`addCustomInfo`, [2](#)

`compare.counts`, [3](#)

`compensate`, `GatingSet`, `graphGML`-method, [3](#)

`constructTree`, [4](#)

`cytobank2GatingSet`, [4](#)

`extend`, [5](#)

`gating`, `graphGML`, `GatingSet`-method, [6](#)

`GatingSet2cytobank`, [7](#)

`GatingSet2flowJo`, [8](#)

`getChildren`, `graphGML`, `character`-method, [8](#)

`getCompensationMatrices`, `graphGML`-method, [9](#)

`getGate`, `graphGML`, `character`-method, [9](#)

`getNodes`, `graphGML`-method, [10](#)

`getParent`, `graphGML`, `character`-method, [11](#)

`getTransformations`, `graphGML`-method, [11](#)

`graphGML`-class, [12](#)

`matchPath`, [12](#)

`parse.gateInfo`, [13](#)

`plot`, `graphGML`, `missing`-method, [13](#)

`read.gatingML.cytobank`, [14](#)

`show`, `graphGML`-method, [14](#)