flowUtils: Gating-ML Support in Flow Cytometry

Josef Spidlen

May 3, 2016

Abstract

Gating in flow cytometry is a highly important process for selecting populations of interests by defining the characteristics of particles for further data acquisition or analysis. Gating-ML represents a specification on how to form unambiguous XML-based gate definitions. Such a description of gates facilitates the interchange and validation of data and analysis among different software packages with the potential for significant increase of hardware and software interoperability.

The flowUtils package supports reading of Gating-ML version 1.5, and both, reading and writing of Gating-ML 2.0. Gating-ML 2.0 is the latest version of Gating-ML as of October 2014.

Keywords: Flow cytometry, gating, XML, data standard

1 Introduction

1.1 Background

Gating in flow cytometry is a highly important process for selecting populations of interests by defining the characteristics of particles for further data acquisition or analysis. A gate is a filter (set of boundaries) that serve to isolate a specific group of cytometric events (*e.g.*, cells) from a larger set. A standard formal way of exchanging unambiguous descriptions of gates is crucial for interoperability among analytical hardware and software applications.

Gating-ML represents a specification on how to form unambiguous XML-based gate definitions. Such a description of gates facilitates the interchange and validation of data and analysis between different software packages with the potential for significant increase of hardware and software interoperability.

1.2 Gating-ML 1.5

Gating-ML has undergone several revisions since the first public release in February 2006. In January 2008, Gating-ML version 1.5 (**?**) became an International Society for Advancement of Cytometry (ISAC) Candidate Recommendation. Gating-ML 1.5 supports rectangular gates in n dimensions (i.e., from one–dimensional range gates up to n-dimensional hyper-rectangular regions), polygon gates in two (and more) dimensions, ellipsoid gates in n dimensions, decision

tree structures, and Boolean collections of any of the types of gates. Gates are uniquely identified and may be ordered into a hierarchical structure to describe a gating strategy. Gates may be applied on parameters (*i.e.*, dimensions) as in list mode data files (*e.g.*, FCS files) or on transformed parameters as described by a data transformation. Supported transformations include logarithmic, polynomial of degree one (*i.e.*, linear combination with translation), square root, asinh (inverse hyperbolic sin), split-scale, Hyperlog, and ratio of two parameters, as well as inverse transformations wherever these exist, *i.e.*, exponential, quadratic transformation, hyperbolic sin, inverse split scale, and EH transformations, and compensation. Arbitrary compound transformations may be created. Gates are applicable on raw "channel" values of the list mode data files unless transformations are explicitly specified.

1.3 Gating-ML 2.0

Based on feedback gathered from the implementors and development in the field, Gating-ML version 2.0 (**?**) has been developed and adopted as a candidate for an ISAC Recommendation in January 2013. Gating-ML 2.0 significantly simplifies several aspects of Gating-ML by focusing on gates, data transformations and pipelines that are useful in flow cytometry, rather than asking implementors to support a very generic approach. Gating-ML 2.0 supports rectangular gates in n dimensions (*i.e.*, from one-dimensional range gates up to n-dimensional hyper-rectangular regions), quadrant gates in n dimensions, polygon gates, ellipsoid gates in n dimensions, and Boolean collections of any of the types of gates. Supported gate types have been selected based on feedback on the Gating-ML 1.5 specification in order to keep it simple while accommodating for future innovations in automated multidimensional gating and clustering in a generic way. Gates are uniquely identified and may be ordered into a hierarchical structure to describe a gating strategy. Gates may be applied on list mode data files (*e.g.*, FCS files), which may be transformed as explicitly described. Gating-ML 2.0 specification supports open transformations (i.e., published and free to use) which have been shown useful for display or analysis of cytometry data, such as Logicle and Hyperlog. In addition, transformations such as linear, logarithmic, and inverse hyperbolic sine are supported and have been extended to allow for additional parameterization and tweaking specifically for the display of flow cytometry data. In Gating-ML 2.0, these extensions are called "FLin", "FLog" and "FASinH", respectively. A parametrized ratio of two FCS dimensions (i.e., "FRatio") and fluorescence compensation complete the list of supported transformations. Compared to Gating-ML 1.5, the list of Gating-ML 2.0 supported transformations has been shortened by omitting transformations that have not been found particularly useful or are no longer necessary due to additional design changes. In addition, values from FCS files are referenced as "scale values" (used to be channel values in Gating-ML 1.5), which eliminates the necessity to encode the "channel to scale" transformation in Gating-ML (this transformation is unambiguously captured by keywords in the FCS data file standard). Finally, Gating-ML 2.0 no longer supports compound transformations in general. Instead, each gate dimension can be defined by referencing up to one "scale" transformation plus an optional fluorescence compensation description applied on a dimension, which may be a parameter from a list mode data file, or the result of an additional transformation, such as the ratio of two FCS parameters. In October 2014, bounded transformations have been added to Gating-ML 2.0. This means that a boundary may be added to any Gating-ML 2.0 scaling transformation, or to a ratio transformation. A boundary restricts a value x (i.e. the result of a

transformation that the boundary is applied to) to the [boundMin, boundMax] interval. Using a boundary allows for unambiguous encoding of gating performed by software tools that pile off-scale events on the graph axes. In these cases, if the selected visualization (scaling transformation) is not quite appropriate, certain events could fall of the graph. However, instead of loosing these events, some software tools prefer to shift them to a predefined minimum or maximum, which may effect gate membership of these events. A Gating-ML boundary may be used in order to mimic such behavior in Gating-ML and encode these gates in a reproducible manner. All these changes have been made based on received community feedback in order to simplify the Gating-ML specification, especially for Gating-ML consumers (readers).

1.4 Gating-ML support in flowUtils

The flowUtils package supports reading of Gating-ML version 1.5 (implemented by N. Gopalakrishnan in 2008), and both reading and writing of Gating-ML 2.0 (implemented by J. Spidlen in 2013–2014). Gating-ML 2.0 is currently (as of October 2014) the latest version of Gating-ML.

2 Reading Gating-ML files

2.1 The `read.gatingML` function

Any Gating-ML 1.5 or Gating-ML 2.0 compliant XML file can be read by the `read.gatingML` function. This function requires an input file name and an environment to save objects parsed from the Gating-ML file.

```
> gateFile <- system.file("extdata", "GatingML2.0_Example1.xml",
+     package = "flowUtils")
> flowEnv  <- new.env()
> read.gatingML(gateFile, flowEnv)
> for (x in ls(flowEnv))
+     if (is(flowEnv[[x]], "filter"))
+         cat(paste("Gate", x, "of class", class(flowEnv[[x]]), "\n"))
```

```
Gate And1 of class intersectFilter
Gate Ellipse1 of class ellipsoidGate
Gate Ellipsoid3D of class ellipsoidGate
Gate FL2N-FL4N of class rectangleGate
Gate FL2N-FL4P of class rectangleGate
Gate FL2P-FL4N of class rectangleGate
Gate FL2P-FL4P of class rectangleGate
Gate Not1 of class complementFilter
Gate Or1 of class unionFilter
Gate ParAnd of class subsetFilter
Gate Polygon1 of class polygonGate
Gate Range1 of class rectangleGate
Gate RatRange1 of class rectangleGate
Gate Rectangle1 of class rectangleGate
```

2.2 Additional examples

Additional Gating-ML file examples are included with the Gating-ML specifications as well as in the gatingMLData package under `extdata/Gating-MLFiles` (for Gating-ML 1.5) and under `extdata/Gml2/Gating-MLFiles` (for Gating-ML 2.0). You can read these files using the following code:

```
> flowEnv1.5  <- new.env()
> g1.5Example <- system.file("extdata/Gating-MLFiles", "01Rectangular.xml",
+   package="gatingMLData")
> read.gatingML(g1.5Example, flowEnv1.5)
> ls(flowEnv1.5)
> flowEnv2.0  <- new.env()
> g2.0Example <- system.file("extdata/Gml2/Gating-MLFiles",
+   "gates3.xml", package = "gatingMLData")
> read.gatingML(g2.0Example, flowEnv2.0)
> ls(flowEnv2.0)
```

2.3 Exploring objects read into the environment

We are not showing the output the `ls` commands above since these contain over a 100 of different objects saved in the environments, including various gates (data filters) and transformations (scale transformations, compensation, etc.). Users are encouraged to explore these objects further. For example, if we type

```
> flowEnv2.0[['myRectangleGate4LogicleArcSinHFCSCompensated']]
```

```
Rectangular gate 'myRectangleGate4LogicleArcSinHFCSCompensated' with dimensions:
  myLogicle.FCS.PE-A: (0.2,0.8)
  Tr_Arcsinh.FCS.APC-Cy7-A: (6.2,9.9)
```

then we can see that `myRectangleGate4LogicleArcSinHFCSCompensated` is a rectangular gate. Let us use the `str` command to explore the details of this gate.

```
> str(flowEnv2.0[['myRectangleGate4LogicleArcSinHFCSCompensated']])
```

```
Formal class 'rectangleGate' [package "flowCore"] with 4 slots
  ..@ min       : num [1:2] 0.2 6.2
  ..@ max       : num [1:2] 0.8 9.9
  ..@ parameters:Formal class 'parameters' [package "flowCore"] with 1 slot
  .. .. ..@ .Data:List of 2
  .. .. .. ..$ :Formal class 'transformReference' [package "flowCore"] with 3 slots
  .. .. .. .. .. ..@ .Data           :function ()
  .. .. .. .. .. ..@ searchEnv       :<environment: 0x7ff8bf8d1cd0>
  .. .. .. .. .. ..@ transformationId: chr "myLogicle.FCS.PE-A"
  .. .. .. ..$ :Formal class 'transformReference' [package "flowCore"] with 3 slots
  .. .. .. .. .. ..@ .Data           :function ()
  .. .. .. .. .. ..@ searchEnv       :<environment: 0x7ff8bf8d1cd0>
```

```
  .. .. .. .. .. ..@ transformationId: chr "Tr_Arcsinh.FCS.APC-Cy7-A"
  ..@ filterId  : chr "myRectangleGate4LogicleArcSinHFCSCompensated"
```

We can see that the `parameters` slot references two transformations. We can explore these further by entering

```
> str(flowEnv2.0[['myLogicle.FCS.PE-A']])

Formal class 'logicletGml2' [package "flowCore"] with 7 slots
  ..@ .Data           :function ()
  ..@ T               : num 262144
  ..@ M               : num 5
  ..@ W               : num 1
  ..@ A               : num 0.5
  ..@ parameters      :Formal class 'compensatedParameter' [package "flowCore"]
                       with 5 slots
  .. .. ..@ .Data           :function ()
  .. .. ..@ parameters      : chr "PE-A"
  .. .. ..@ spillRefId      : chr "SpillFromFCS"
  .. .. ..@ searchEnv       :<environment: 0x66ac3e0>
  .. .. ..@ transformationId: chr "PE-A_compensated_according_to_FCS"
  ..@ transformationId: chr "myLogicle.FCS.PE-A"

> str(flowEnv2.0[['Tr_Arcsinh.FCS.APC-Cy7-A']])

Formal class 'asinhtGml2' [package "flowCore"] with 6 slots
  ..@ .Data           :function ()
  ..@ T               : num 1.18
  ..@ M               : num 0.434
  ..@ A               : num 0
  ..@ parameters      :Formal class 'compensatedParameter' [package "flowCore"]
                       with 5 slots
  .. .. ..@ .Data           :function ()
  .. .. ..@ parameters      : chr "APC-Cy7-A"
  .. .. ..@ spillRefId      : chr "SpillFromFCS"
  .. .. ..@ searchEnv       :<environment: 0x66ac3e0>
  .. .. ..@ transformationId: chr "APC-Cy7-A_compensated_according_to_FCS"
  ..@ transformationId: chr "Tr_Arcsinh.FCS.APC-Cy7-A"
```

This reveals that `myLogicle.FCS.PE-A` is a Logicle transformation applied to the "PE-A" parameter, which has been compensated according to the description in the data file (*e.g.*, the `$SPILLOVER`, `SPILL` or other keywords). Analogically, we can see that `Tr_Arcsinh.FCS.APC-Cy7-A` is an inverse hyperbolic sine (ArcSinH) transformation applied to the "APC-Cy7-A" parameter, which has also been compensated according to the description in the data file.

2.4 Scaling transformations shared in Gating-ML 2.0, not in R

In the previous example, we can also see how scaling transformations are mapped between R and Gating-ML. In Gating-ML 1.5 and in R, each scaling transformation is bound to its argument (*i.e.*, to the FCS parameter) that it is supposed to be applied to. From the snippet of Gating-ML 1.5 code below, we can see that transformations "T1" and "T2" are "the same" except one of them is applied to the "FL1-H" parameter while the other one to the "FL2-H" parameter. In Gating-ML 1.5 and in R, these transformations have to be defined separately, each of them having a unique identifier assigned.

```
<!-- Snippet of Gating-ML 1.5 code -->
<transforms:transformation transforms:id="T1">
  <transforms:ln transforms:a="1" transforms:b="111.1793874">
    <data-type:parameter data-type:name="FL1-H" />
  </transforms:ln>
</transforms:transformation>
<transforms:transformation transforms:id="T2">
  <transforms:ln transforms:a="1" transforms:b="111.1793874">
    <data-type:parameter data-type:name="FL2-H" />
  </transforms:ln>
</transforms:transformation>
<gating:RectangleGate gating:id="R1">
  <gating:dimension gating:min="1" gating:max="3">
    <transforms:transformationReference transforms:ref="T1" />
  </gating:dimension>
  <gating:dimension gating:min="2" gating:max="4">
    <transforms:transformationReference transforms:ref="T2" />
  </gating:dimension>
</gating:RectangleGate>
```

This design has been primarily driven by the fact that Gating-ML 1.5 and R allow for an arbitrary combination of data transformations, although most of these compound transformations are not very meaningful for the analysis of flow cytometry data. Gating-ML 2.0 supports only pipelines that are considered meaningful for static gate based analysis of flow cytometry data (see section **??**). Consequently, only a single scaling transformation may be included when an FCS parameter is being transformed. Therefore, the Gating-ML 2.0 description is simpler and allows for the same transformation to be applicable to multiple FCS parameters. We can see the design difference in the snippet of Gating-ML 2.0 code below.

```
<!-- Snippet of Gating-ML 2.0 code -->
<transforms:transformation transforms:id="T1">
  <transforms:flog transforms:T="1024" transforms:M="4" />
</transforms:transformation>
<gating:RectangleGate gating:id="R1">
  <gating:dimension gating:min="1" gating:max="3"
   gating:transformation-ref="T1" gating:compensation-ref="uncompensated">
```

```
    <data-type:fcs-dimension data-type:name="FL1-H" />
  </gating:dimension>
  <gating:dimension gating:min="2" gating:max="4"
   gating:transformation-ref="T1" gating:compensation-ref="uncompensated">
    <data-type:fcs-dimension data-type:name="FL2-H" />
  </gating:dimension>
</gating:RectangleGate>
```

Consequently, when scaling transformations are read from a Gating-ML 2.0 file, multiple instances may be created in R if the same transformation is being applied to different FCS parameters (in one or more gates). New identifiers for these transformations will be created based on what the scaling transformation is applicable to. Based on the above example, we may see two transformations saved in the environment once this Gating-ML 2.0 snippet is read. One of them will be identified as "T1.uncompensated.FL1-H" and the other one as "T1.uncompensated.FL2-H".

2.5 Representation of spillover and spectrum matrices

Compensation objects are also parsed from the Gating-ML files and saved in the environment. For example, the "myPolygonGateWithCustomSpillover" polygon gate is drawn in the "PE-A" and "PerCP-Cy5-5-A" parameters, which are compensated according to the "MySpill" spillover matrix. This matrix has been extracted from the Gating-ML 2.0 file.

```
> str(flowEnv2.0[['myPolygonGateWithCustomSpillover']])

Formal class 'polygonGate' [package "flowCore"] with 3 slots
  ..@ boundaries: num [1:3, 1:2] 5 500 500 5 5 500
  ..@ parameters:Formal class 'parameters' [package "flowCore"] with 1 slot
  .. .. ..@ .Data:List of 2
  .. .. .. ..$ :Formal class 'compensatedParameter' [package "flowCore"] with 5 slots
  .. .. .. .. .. ..@ .Data           :function ()
  .. .. .. .. .. ..@ parameters      : chr "PE-A"
  .. .. .. .. .. ..@ spillRefId      : chr "MySpill"
  .. .. .. .. .. ..@ searchEnv       :<environment: 0x7ff8bf8d1cd0>
  .. .. .. .. .. ..@ transformationId: chr "Comp-PE"
  .. .. .. ..$ :Formal class 'compensatedParameter' [package "flowCore"] with 5 slots
  .. .. .. .. .. ..@ .Data           :function ()
  .. .. .. .. .. ..@ parameters      : chr "PerCP-Cy5-5-A"
  .. .. .. .. .. ..@ spillRefId      : chr "MySpill"
  .. .. .. .. .. ..@ searchEnv       :<environment: 0x7ff8bf8d1cd0>
  .. .. .. .. .. ..@ transformationId: chr "Comp-PerCP-Cy5-5"
  ..@ filterId  : chr "myPolygonGateWithCustomSpillover"

> flowEnv2.0[['MySpill']]
```

```
Compensation object 'MySpill':
                 PE-A PerCP-Cy5-5-A APC-A
Comp-PE          1.00          0.02  0.06
Comp-PerCP-Cy5-5 0.11          1.00  0.07
Comp-APC         0.09          0.01  1.00
```

Non-square spectrum matrices are extracted and saved the same way. These matrices are supported in Gating-ML 2.0 only. For example, you may want to review the "myPolygonGateWithCustomNonSquareSpectrumMatrix" and "MyNonSquareSpectrum" objects in the "flowEnv2.0" environment.

2.6 Applying Gating-ML files

Once the elements from a Gating-ML file have been saved in an environment, the "filters" (*a.k.a.* the gates) can be used to gate an FCS data file (*i.e.,* a `flowFrame`), or a set of FCS files (*i.e.,* a `flowSet`). Please pay attention to the `transformation` argument when reading the FCS files. Gating-ML 1.5 specifies that data shall be used as channel values by default, and any additional transformation shall be explicitly specified in the Gating-ML 1.5 file. Therefore, you will need to prevent R from applying the default "channel-to-scale" transformation when reading your data with the intention of applying a Gating-ML 1.5 file to it. This can be done by specifying `transformation=FALSE` in the `read.FCS` and `read.flowSet` functions.

Gating-ML 2.0 specifies that event "scale values" shall be used by default. This means that the "channel-to-scale" transformation (as defined by the keywords within the FCS data file) shell be applied prior applying any additional transformations described in the Gating-ML 2.0 file. Therefore, you should specify `transformation=linearize-with-PnG-scaling` in the `read.FCS` or `read.flowSet` functions if you will be working with Gating-ML 2.0 files. An example of applying a Gating-ML 2.0 file to an FCS data file is shown bellow:

```
> fcsFile <- system.file("extdata/Gml2/FCSFiles", "data1.fcs",
+     package = "gatingMLData")
> myFrame <- read.FCS(fcsFile, transformation="linearize-with-PnG-scaling")
> for (x in ls(flowEnv)) if (is(flowEnv[[x]], "filter")) {
+     result <- filter(myFrame, flowEnv[[x]])
+     print(summary(result))
+ }

And1+: 132 of 13367 events (0.99%)
Ellipse1+: 3083 of 13367 events (23.06%)
Ellipsoid3D+: 4191 of 13367 events (31.35%)
FL2N-FL4N+: 5148 of 13367 events (38.51%)
FL2N-FL4P+: 238 of 13367 events (1.78%)
FL2P-FL4N+: 7361 of 13367 events (55.07%)
FL2P-FL4P+: 620 of 13367 events (4.64%)
Not1+: 10284 of 13367 events (76.94%)
Or1+: 4507 of 13367 events (33.72%)
```

```
ParAnd+: 9 of 13367 events (0.07%)
Polygon1+: 1582 of 13367 events (11.84%)
Range1+: 440 of 13367 events (3.29%)
RatRange1+: 7679 of 13367 events (57.45%)
Rectangle1+: 252 of 13367 events (1.89%)
```

3 Writing Gating-ML files

3.1 The `write.gatingML` function

Gating-ML 2.0 compatible objects stored in an environment may be written to a Gating-ML 2.0 file using the `write.gatingML` function. Please see table **??** for details about Gating-ML compatible objects. These objects may have been created by the `read.gatingML` function, or in any other way. Below, we demonstrate how to programmatically create a simple rectangular gate and save the result in a Gating-ML 2.0 file. Please note that for readability reasons, pieces that are not significant for the understanding of examples have been omitted from XML listings in this vignette. These include lengthy XML namespace declarations and custom information produced by the `write.gatingML` function, such as details about the origin of the produced Gating-ML file. The skipped output is noted by "`...`" in the XML listings. Readers are encouraged to run the examples themselves to review the full output.

```
> flowEnv <- new.env()
> flowEnv[['myGate']] <- rectangleGate(filterId="myGate",
+     list("FSC-H"=c(150, 300), "SSC-H"=c(200, 600)))
> outputFile <- tempfile(fileext=".gating-ml2.xml")
> write.gatingML(flowEnv, outputFile)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gating:Gating-ML ... >
  ...
  <gating:RectangleGate gating:id="myGate">
    <gating:dimension gating:min="150" gating:max="300"
                      gating:compensation-ref="uncompensated">
      <data-type:fcs-dimension data-type:name="FSC-H"/>
    </gating:dimension>
    <gating:dimension gating:min="200" gating:max="600"
                      gating:compensation-ref="uncompensated">
      <data-type:fcs-dimension data-type:name="SSC-H"/>
    </gating:dimension>
  </gating:RectangleGate>
</gating:Gating-ML>
```

The second argument of the `write.gatingML` function (*i.e.*, the file name) is optional. The output is written to the "standard output" (*e.g.*, the console) if no filename is provided.

Table 1: Summary of Gating-ML concepts and related R classes

Gating-ML concept	**R Class**	**Gating-ML version**
RectangleGate	rectangleGate	1.5, 2.0
Quadrant	rectangleGate (read), quadGate (write)*	2.0
PolygonGate	polygonGate	1.5, 2.0
EllipsoidGate	ellipsoidGate	1.5, 2.0
Boolean "or" gate	unionFilter	1.5, 2.0
Boolean "and" gate	intersectFilter	1.5, 2.0
Boolean "not" gate	complementFilter	1.5, 2.0
Gate with a parent	subsetFilter	1.5, 2.0
PolytopeGate	polytopeGate	1.5
DecisionTreeGate	expressionFilter	1.5
Referenced gate	filterReference	1.5, 2.0
flin	lintGml2	2.0
flog	logtGml2	2.0
fasinh	asinhtGml2	2.0
logicle	logicletGml2	2.0
hyperlog	hyperlog (v 1.5), hyperlogtGml2 (v 2.0)	1.5, 2.0
fratio	ratiotGml2	2.0
dg1polynomial	dg1polynomial	1.5
ratio	ratio	1.5 (2.0**)
quadratic	quadratic	1.5
sqrt	squareroot	1.5
ln	logarithm	1.5
exponential	exponential	1.5
asinh	asinht	1.5 (2.0**)
sinh	sinht	1.5
EH	EHtrans	1.5
split-scale	splitscale	1.5
inverse-split-scale	invsplitscale	1.5
spilloverMatrix	compensation, compensatedParameter	1.5
spectrumMatrix	compensation, compensatedParameter	2.0

* The Quadrant gate in Gating-ML 2.0 allows for arbitrary splits of n-dimensional space, including more than one "cut" per dimension, and with the option of merging several of these "cuts" into a resulting "quadrant". The *quadGate* filter in R is a less flexible structure implementing the traditional two-dimensional quadrant gate concept (*i.e.,* with each dimension split exactly once, and always resulting in 4 quadrants). Therefore, a *quadGate* filter is saved as a Quadrant gate in Gating-ML; however, if a Quadrant gate is read from Gating-ML, then a set of appropriate *rectangleGate* filters is created.

** For Gating-ML 2.0 output, the "ratio" and "asinht" transformations from Gating-ML 1.5 will be converted to "fratio" and "fasinh", respectively.

3.2 Gating-ML compatible objects

Table **??** summarizes what R classes are used to capture various Gating-ML concepts (*i.e.,* gates, transformations, and compensations). Corresponding flowCore filters and transformations are created when Gating-ML 1.5 or 2.0 is read, and the same types of filters and transformations can be saved in Gating-ML 2.0 as long as they are Gating-ML 2.0 compatible and the analysis "pipeline" is expressible in Gating-ML 2.0 (see section **??**). Data driven filters (*e.g.*, *norm2Filter*, *kmeansFilter*, *curv1Filter*, *curv2Filter*, *boundaryFilter*, etc.) are not supported by Gating-ML.

3.3 Gating-ML 2.0 compatible pipelines

R is a powerful language allowing you to create and combine various data transformations and use these as dimensions (parameters) for your FCS data filters. However, Gating-ML 2.0 supports only the pipelines that are considered meaningful for static gate based analysis of flow cytometry data. This design decision has been made in order to make Gating-ML 2.0 implementation feasible for common flow cytometry data analysis tools. In practice, Gating-ML 2.0 compatible pipelines (*a.k.a.* "workflows") consist of the following steps:

1. Read an FCS data file and apply the "channel to scale" transformations to FCS parameters as specified by the \$PnE and \$PnG keywords. (These transformations are not explicitly described in Gating-ML.)

2. Optionally: apply compensation based on either a compensation description in the FCS data file (*e.g.*, the `$SPILLOVER`, `SPILL` or other keywords), or based on a "spectrum" matrix described in the Gating-ML 2.0 file. A spectrum matrix covers both, the traditional compensation based on square spillover matrices, as well as spectral unmixing, see (**?**).

3. For further steps, use either FCS parameters, or a "fratio" of two FCS parameters. A "fratio" in Gating-ML 2.0 is an extended ratio of two FCS parameters defined as $A\frac{x-B}{y-C}$, where x and y are FCS parameters, and $A \in \mathbb{R}$, $B \in \mathbb{R}$, and $C \in \mathbb{R}$ are constants.

4. Optionally: apply one of the Gating-ML 2.0 compatible scale transformation, *i.e.,* parameterized linear, logarithmical, inverse hyperbolic sine, Logicle or Hyperlog transformation. A transformation boundary may be used; see the boundMin and boundMax parameters of the Gating-ML 2.0 transformation functions.

5. Apply Gating-ML 2.0 compatible gates in the data space created by previous steps. Gating-ML 2.0 supported gate types include polygon gates, ellipsoid gates, range gates, rectangular and hyper-rectangular gates, quadrant gates and Boolean collections (*i.e.*, union, intersect or complement) of any of the gate types.

3.4 Examples with compensation

Example below demonstrates the inclusion of a compensation description in the Gating-ML output. Same as before, objects will be created programmatically and exported in a Gating-ML 2.0 output.

```
> flowEnv <- new.env()
> covM <- matrix(c(62.5, 37.5, 37.5, 62.5), nrow = 2, byrow=TRUE)
> colnames(covM) <- c("FL1-H", "FL2-H")
> compPars <- list(
+   compensatedParameter(parameters="FL1-H", spillRefId="SpillFromFCS",
+     transformationId=paste("FL1-H", "_compensated_according_to_FCS", sep=""),
+     searchEnv=flowEnv),
+   compensatedParameter(parameters="FL2-H", spillRefId="SpillFromFCS",
+     transformationId=paste("FL2-H", "_compensated_according_to_FCS", sep=""),
+     searchEnv=flowEnv)
+ )
> myEl <- ellipsoidGate(mean=c(12, 16), distance=1, .gate=covM, filterId="myEl")
> myEl@parameters <- new("parameters", compPars)
> flowEnv[['myEl']] <- myEl
> write.gatingML(flowEnv)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gating:Gating-ML ... ">
  ...
  <gating:EllipsoidGate gating:id="myEl">
    <gating:dimension gating:compensation-ref="FCS">
      <data-type:fcs-dimension data-type:name="FL1-H"/>
    </gating:dimension>
    <gating:dimension gating:compensation-ref="FCS">
      <data-type:fcs-dimension data-type:name="FL2-H"/>
    </gating:dimension>
    <gating:mean>
      <gating:coordinate data-type:value="12"/>
      <gating:coordinate data-type:value="16"/>
    </gating:mean>
    <gating:covarianceMatrix>
      <gating:row>
        <gating:entry data-type:value.FL1-H="62.5"/>
        <gating:entry data-type:value.FL2-H="37.5"/>
      </gating:row>
      <gating:row>
        <gating:entry data-type:value.FL1-H="37.5"/>
        <gating:entry data-type:value.FL2-H="62.5"/>
      </gating:row>
    </gating:covarianceMatrix>
    <gating:distanceSquare data-type:value="1"/>
  </gating:EllipsoidGate>
</gating:Gating-ML>
```

The `spillRefId="SpillFromFCS"` indicates that compensation according to the description in the FCS data file shall used. In the Gating-ML output, this is described as `<gating:dimension`

gating:compensation-ref="FCS"> for the appropriate dimensions. If we wanted to use a compensation based on a custom spillover (or spectrum) matrix instead, we could modify the code as follows:

```
> spillM <- matrix(c(1, 0.03, 0.07, 1), nrow = 2, byrow=TRUE)
> colnames(spillM) <- c("FL1-H", "FL2-H")
> rownames(spillM) <- c("Comp-FL1-H", "Comp-FL2-H")
> pars <- new("parameters", list("FL1-H", "FL2-H"))
> myComp <- compensation(spillover=spillM, compensationId='myComp', pars)
> flowEnv[['myComp']] <- myComp
> compPars <- list(
+   compensatedParameter(parameters="FL1-H", spillRefId="myComp",
+     transformationId="Comp-FL1-H", searchEnv=flowEnv),
+   compensatedParameter(parameters="FL2-H", spillRefId="myComp",
+     transformationId="Comp-FL2-H", searchEnv=flowEnv)
+ )
> myEl@parameters <- new("parameters", compPars)
> flowEnv[['myEl']] <- myEl
> write.gatingML(flowEnv)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gating:Gating-ML ... >
  ...
  <transforms:spectrumMatrix transforms:id="myComp">
    <transforms:fluorochromes>
      <data-type:fcs-dimension data-type:name="Comp-FL1-H"/>
      <data-type:fcs-dimension data-type:name="Comp-FL2-H"/>
    </transforms:fluorochromes>
    <transforms:detectors>
      <data-type:fcs-dimension data-type:name="FL1-H"/>
      <data-type:fcs-dimension data-type:name="FL2-H"/>
    </transforms:detectors>
    <transforms:spectrum>
      <transforms:coefficient transforms:value="1"/>
      <transforms:coefficient transforms:value="0.03"/>
    </transforms:spectrum>
    <transforms:spectrum>
      <transforms:coefficient transforms:value="0.07"/>
      <transforms:coefficient transforms:value="1"/>
    </transforms:spectrum>
  </transforms:spectrumMatrix>
  <gating:EllipsoidGate gating:id="myEl">
    <gating:dimension gating:compensation-ref="myComp">
      <data-type:fcs-dimension data-type:name="Comp-FL1-H"/>
    </gating:dimension>
    <gating:dimension gating:compensation-ref="myComp">
```

```
      <data-type:fcs-dimension data-type:name="Comp-FL2-H"/>
    </gating:dimension>
    ...
  </gating:EllipsoidGate>
</gating:Gating-ML>
```

Note that new names have been assigned to parameters compensated according to a custom spillover matrix (*i.e.*, `Comp-FL1-H` and `Comp-FL2-H`). This is necessary due to the generic Gating-ML 2.0 design, which also supports non-square spectrum matrices (where there is no direct one-to-one correspondence between the measured and "compensated" values). The following piece of code demonstrates how a non-square spectrum matrix can be generated and saved in Gating-ML.

```
> flowEnv <- new.env()
> specM <- matrix(c(0.78, 0.13, 0.22, 0.05, 0.57, 0.89), nrow = 2, byrow=TRUE)
> colnames(specM) <- c("FL1-H", "FL2-H", "FL3-H")
> rownames(specM) <- c("Deconvoluted-P1", "Deconvoluted-P2")
> pars <- new("parameters", list("FL1-H", "FL2-H", "FL3-H"))
> mySpecM <- compensation(spillover=specM, compensationId='specM', pars)
> flowEnv[['mySpecM']] <- mySpecM
> compPars <- list(
+   compensatedParameter(parameters="FL1-H", spillRefId="mySpecM",
+     transformationId="Deconvoluted-P1", searchEnv=flowEnv),
+   compensatedParameter(parameters="FL2-H", spillRefId="mySpecM",
+     transformationId="Deconvoluted-P2", searchEnv=flowEnv)
+ )
> myEl@parameters <- new("parameters", compPars)
> flowEnv[['myEl']] <- myEl
> write.gatingML(flowEnv)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gating:Gating-ML ... >
 ...
  <transforms:spectrumMatrix transforms:id="specM">
    <transforms:fluorochromes>
      <data-type:fcs-dimension data-type:name="Deconvoluted-P1"/>
      <data-type:fcs-dimension data-type:name="Deconvoluted-P2"/>
    </transforms:fluorochromes>
    <transforms:detectors>
      <data-type:fcs-dimension data-type:name="FL1-H"/>
      <data-type:fcs-dimension data-type:name="FL2-H"/>
      <data-type:fcs-dimension data-type:name="FL3-H"/>
    </transforms:detectors>
    <transforms:spectrum>
      <transforms:coefficient transforms:value="0.78"/>
      <transforms:coefficient transforms:value="0.13"/>
```

```
      <transforms:coefficient transforms:value="0.22"/>
    </transforms:spectrum>
    <transforms:spectrum>
      <transforms:coefficient transforms:value="0.05"/>
      <transforms:coefficient transforms:value="0.57"/>
      <transforms:coefficient transforms:value="0.89"/>
    </transforms:spectrum>
  </transforms:spectrumMatrix>
</gating:Gating-ML>
```

3.5 Example with scaling transformations

In the following example, we will use a *quadGate* to demonstrate how scaling transformations can be included in the Gating-ML output.

```
> flowEnv <- new.env()
> myTrQuad <- quadGate(filterId = "myTrQuad", "APC-A" = 0.5, "APC-Cy7-A" = 0.6)
> trArcSinH1 <- asinhtGml2(parameters = "APC-A", T = 1000, M = 4.5, A = 0,
+   transformationId="trArcSinH1")
> trLogicle1 <- logicletGml2(parameters = "APC-Cy7-A", T = 1000, W = 0.5,
+   M = 4.5, A = 0, transformationId="trLogicle1")
> flowEnv[['trArcSinH1']] <- trArcSinH1
> flowEnv[['trLogicle1']] <- trLogicle1
> trPars <- list(
+   transformReference("trArcSinH1", flowEnv),
+   transformReference("trLogicle1", flowEnv)
+ )
> myTrQuad@parameters <- new("parameters", trPars)
> flowEnv[['myTrQuad']] <- myTrQuad
> write.gatingML(flowEnv)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gating:Gating-ML ... >
  ...
  <transforms:transformation transforms:id="trArcSinH1">
    <transforms:fasinh transforms:T="1000" transforms:M="4.5" transforms:A="0"/>
  </transforms:transformation>
  <transforms:transformation transforms:id="trLogicle1">
    <transforms:logicle transforms:T="1000" transforms:M="4.5" transforms:W="0.5"
                        transforms:A="0"/>
  </transforms:transformation>
  <gating:QuadrantGate gating:id="myTrQuad">
    <gating:divider gating:transformation-ref="trArcSinH1"
                    gating:compensation-ref="uncompensated" gating:id="myTrQuad.D1">
      <data-type:fcs-dimension data-type:name="APC-A"/>
      <gating:value>0.5</gating:value>
```

```
    </gating:divider>
    <gating:divider gating:transformation-ref="trLogicle1"
                    gating:compensation-ref="uncompensated" gating:id="myTrQuad.D2">
      <data-type:fcs-dimension data-type:name="APC-Cy7-A"/>
      <gating:value>0.6</gating:value>
    </gating:divider>
    <gating:Quadrant gating:id="myTrQuad.PP">
      <gating:position gating:divider_ref="myTrQuad.D1" gating:location="1.5"/>
      <gating:position gating:divider_ref="myTrQuad.D2" gating:location="1.6"/>
    </gating:Quadrant>
    <gating:Quadrant gating:id="myTrQuad.PN">
      <gating:position gating:divider_ref="myTrQuad.D1" gating:location="1.5"/>
      <gating:position gating:divider_ref="myTrQuad.D2" gating:location="-0.4"/>
    </gating:Quadrant>
    <gating:Quadrant gating:id="myTrQuad.NP">
      <gating:position gating:divider_ref="myTrQuad.D1" gating:location="-0.5"/>
      <gating:position gating:divider_ref="myTrQuad.D2" gating:location="1.6"/>
    </gating:Quadrant>
    <gating:Quadrant gating:id="myTrQuad.NN">
      <gating:position gating:divider_ref="myTrQuad.D1" gating:location="-0.5"/>
      <gating:position gating:divider_ref="myTrQuad.D2" gating:location="-0.4"/>
    </gating:Quadrant>
  </gating:QuadrantGate>
</gating:Gating-ML>
```

If we wanted to add a boundary to the transformation, we could do so by adding the boundMin and/or boundMax attributes to the transformation definition as follows:

```
> trArcSinH1 <- asinhtGml2(parameters = "APC-A", T = 1000, M = 4.5, A = 0,
+   transformationId="trArcSinH1", boundMin = 0.02, boundMax = 0.96)
> trLogicle1 <- logicletGml2(parameters = "APC-Cy7-A", T = 1000, W = 0.5,
+   M = 4.5, A = 0, transformationId="trLogicle1", boundMin = -0.04)
```

3.6 Example with scaling transformations and compensation

Previous code (section **??**) can be easily modified so that compensated parameters are used as arguments of the Logicle and ArcSinH transformations. In addition, we will use these transformations directly rather than creating a transformation reference. Consequently, the `write.gatingML` function will create the "trArcSinH1" and "trLogicle1" transformations even without us having to save these in the environment.

```
> rm(list=ls(flowEnv), envir=flowEnv)
> trArcSinH1@parameters <- compensatedParameter(parameters="APC-A",
+   spillRefId="SpillFromFCS", searchEnv=flowEnv,
+   transformationId= "APC-A_compensated_according_to_FCS")
> trLogicle1@parameters <- compensatedParameter(parameters="APC-Cy7-A",
```

```
+   spillRefId="SpillFromFCS", searchEnv=flowEnv,
+   transformationId="APC-Cy7-A_compensated_according_to_FCS")
> trPars <- list(trArcSinH1,trLogicle1)
> myTrQuad@parameters <- new("parameters", trPars)
> flowEnv[['myTrQuad']] <- myTrQuad
> write.gatingML(flowEnv)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gating:Gating-ML ... >
  ...
  <transforms:transformation transforms:id="trArcSinH1">
    <transforms:fasinh transforms:T="1000" transforms:M="4.5" transforms:A="0"/>
  </transforms:transformation>
  <transforms:transformation transforms:id="trLogicle1">
    <transforms:logicle transforms:T="1000" transforms:M="4.5" transforms:W="0.5"
                        transforms:A="0"/>
  </transforms:transformation>
  <gating:QuadrantGate gating:id="myTrQuad">
    <gating:divider gating:transformation-ref="trArcSinH1"
                    gating:compensation-ref="FCS" gating:id="myTrQuad.D1">
      <data-type:fcs-dimension data-type:name="APC-A"/>
      <gating:value>0.5</gating:value>
    </gating:divider>
    <gating:divider gating:transformation-ref="trLogicle1"
                    gating:compensation-ref="FCS" gating:id="myTrQuad.D2">
      <data-type:fcs-dimension data-type:name="APC-Cy7-A"/>
      <gating:value>0.6</gating:value>
    </gating:divider>
    ...
  </gating:QuadrantGate>
</gating:Gating-ML>
```

3.7 Gating-ML 1.5 objects in Gating-ML 2.0 output

In certain cases, a Gating-ML 1.5 compatible transformation can be transformed and expressed in Gating-ML 2.0 (see table **??**). For example, the Gating-ML 1.5 "ratio" transformation can be expressed as Gating-ML 2.0 "fratio". The Gating-ML 1.5 "ratio" is defined as

$$f(x, y) = \frac{x}{y}$$

The parameterized "fratio" transformation in Gating-ML 2.0 is defined as

$$f(x, y, A, B, C) = A\frac{x - B}{y - C}$$

Therefore, we can express the Gating-ML 1.5 "ratio" as Gating-ML 2.0 "fratio" by setting $A = 1$, $B = 0$, and $C = 0$. Example shown below demonstrates that this conversion is done automatically when the `write.gatingML` is called:

```
> flowEnv <- new.env()
> rat1 <- ratio("FSC-A", "SSC-A", transformationId = "rat1")
> myRectGate <- rectangleGate(filterId="myRectGate", "rat1"=c(0.8, 1.4))
> myRectGate@parameters <- new("parameters", list(rat1))
> flowEnv[['myRectGate']] <- myRectGate
> write.gatingML(flowEnv)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gating:Gating-ML ...>
  ...
  <transforms:transformation transforms:id="rat1">
    <transforms:fratio transforms:A="1" transforms:B="0" transforms:C="0">
      <data-type:fcs-dimension data-type:name="FSC-A"/>
      <data-type:fcs-dimension data-type:name="SSC-A"/>
    </transforms:fratio>
  </transforms:transformation>
  <gating:RectangleGate gating:id="myRectGate">
    <gating:dimension gating:min="0.8" gating:max="1.4"
                      gating:compensation-ref="uncompensated">
      <data-type:new-dimension data-type:transformation-ref="rat1"/>
    </gating:dimension>
  </gating:RectangleGate>
</gating:Gating-ML>
```

Similarly for the parameterized inverse hyperbolic sine transformation, which in Gating-ML 1.5 is defined as

$$\mathrm{f}(x, a, b) = \mathrm{asinh}(ax) * b$$

and in Gating-ML 2.0 as

$$\mathrm{f}(x, T, M, A) = \frac{\mathrm{asinh}(x \sinh(M \ln(10))/T) + A \ln(10)}{(M + A) \ln(10)}$$

Therefore, the `write.gatingML` function can convert the Gating-ML 1.5 parameterization to Gating-ML 2.0 by setting $A = 0$, $M = 1/(b * \ln(10))$ and $T = (\sinh(1/b))/a$ as demonstrated below:

```
> flowEnv <- new.env()
> myASinH <- asinht("FL3-W", a = 1.5828, b = 0.0965, transformationId = "myASinH")
> gate1 <- rectangleGate(filterId="gate1", "myASinH"=c(0.3, 0.7))
> gate1@parameters <- new("parameters", list(myASinH))
> flowEnv[['gate1']] <- gate1
> write.gatingML(flowEnv)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gating:Gating-ML ...>
  ...
```

```
  <transforms:transformation transforms:id="myASinH">
    <transforms:fasinh transforms:T="10000.1131651903"
                       transforms:M="4.5004609523653"
                       transforms:A="0"/>
  </transforms:transformation>
  <gating:RectangleGate gating:id="gate1">
    <gating:dimension gating:min="0.3" gating:max="0.7"
                      gating:transformation-ref="myASinH"
                      gating:compensation-ref="uncompensated">
      <data-type:fcs-dimension data-type:name="FL3-W"/>
    </gating:dimension>
  </gating:RectangleGate>
</gating:Gating-ML>
```

3.8 Example with compensation, ratio and scaling together

So far, our examples included relatively simple gates. Next, we will demonstrate the use of a custom compensation along with a scaling transformation (log) applied to the ratio of two FCS parameters, "FL1-A" and "FL1-W". This will create one dimension of a polygon gate. The second dimension will be created as a Hyperlog transformation of "FL2-A", which will be compensated using the same spillover matrix. This example also demonstrates a spillover matrix with multiple measurement types of the same signal (*i.e.*, the area and width). As noted in (**?**), the recommended approach is to set up a sparse spillover matrix that isolates the different measurement types by setting some matrix elements to zero, indicating no spillover between two measurements. By specifying a value of zero for the spillover between different measurement types, the different measurement types are isolated in the matrix. Thus, the spillover for one measurement type can be properly accounted for independent of any other type using a single matrix.

```
> flowEnv <- new.env()
> # Creation of a simplified spillover matrix
> spillM <- matrix(c(1, 0, 0.03, 0, 0, 1, 0, 0.07, 0.1, 0, 1, 0, 0, 0.05, 0, 1),
+     nrow = 4, byrow=TRUE)
> colnames(spillM) <- c("FL1-A", "FL1-W", "FL2-A", "FL2-W")
> rownames(spillM) <- c("cFL1-A", "cFL1-W", "cFL2-A", "cFL2-W")
> pars <- new("parameters", list("FL1-A", "FL1-W", "FL2-A", "FL2-W"))
> myComp <- compensation(spillover=spillM, compensationId='myComp', pars)
> flowEnv[['myComp']] <- myComp
> myComp

Compensation object 'myComp':
       FL1-A FL1-W FL2-A FL2-W
cFL1-A   1.0  0.00  0.03  0.00
cFL1-W   0.0  1.00  0.00  0.07
cFL2-A   0.1  0.00  1.00  0.00
cFL2-W   0.0  0.05  0.00  1.00
```

```
> # First dimension is a log(cFL1-A / cFL1-W)
> myRatio <- ratio("FL1-A", "FL1-W", transformationId = "myRatio")
> myRatio@numerator <- compensatedParameter(parameters="FL1-A",
+     spillRefId="myComp", transformationId="cFL1-A", searchEnv=flowEnv)
> myRatio@denominator <- compensatedParameter(parameters="FL1-W",
+     spillRefId="myComp", transformationId="cFL1-W", searchEnv=flowEnv)
> myLog <- logtGml2(myRatio, T = 1, M = 1, transformationId="myLog")
> # Second dimension is a Hyperlog(cFL2-A)
> secPar <- compensatedParameter(parameters="FL2-A", spillRefId="myComp",
+     transformationId="cFL2-A", searchEnv=flowEnv)
> myHLog <- hyperlogtGml2(secPar, T=262144, M=4.5, W=0.5, A=0, "myHLog")
> # A Polygon gate in the two defined dimensions
> vertices <- matrix(c(0.9, 0.5, 1.2, 0.6, 1.1, 0.8), nrow=3, ncol=2, byrow=TRUE)
> myGate <- polygonGate(filterId="myGate", .gate=vertices,
+     new("parameters", list(myLog, myHLog)))
> flowEnv[['myGate']] <- myGate
> # Finally, write the Gating-ML output
> write.gatingML(flowEnv)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gating:Gating-ML ...>
  ...
  <transforms:transformation transforms:id="myHLog">
    <transforms:hyperlog transforms:T="262144" transforms:M="4.5"
                         transforms:W="0.5" transforms:A="0"/>
  </transforms:transformation>

  <transforms:transformation transforms:id="myLog">
    <transforms:flog transforms:T="1" transforms:M="1"/>
  </transforms:transformation>

  <transforms:transformation transforms:id="myRatio">
    <transforms:fratio transforms:A="1" transforms:B="0" transforms:C="0">
      <data-type:fcs-dimension data-type:name="cFL1-A"/>
      <data-type:fcs-dimension data-type:name="cFL1-W"/>
    </transforms:fratio>
  </transforms:transformation>

  <transforms:spectrumMatrix transforms:id="myComp">
    <transforms:fluorochromes>
      <data-type:fcs-dimension data-type:name="cFL1-A"/>
      <data-type:fcs-dimension data-type:name="cFL1-W"/>
      <data-type:fcs-dimension data-type:name="cFL2-A"/>
      <data-type:fcs-dimension data-type:name="cFL2-W"/>
    </transforms:fluorochromes>
```

```
    <transforms:detectors>
      <data-type:fcs-dimension data-type:name="FL1-A"/>
      <data-type:fcs-dimension data-type:name="FL1-W"/>
      <data-type:fcs-dimension data-type:name="FL2-A"/>
      <data-type:fcs-dimension data-type:name="FL2-W"/>
    </transforms:detectors>
    <transforms:spectrum>
      <transforms:coefficient transforms:value="1"/>
      <transforms:coefficient transforms:value="0"/>
      <transforms:coefficient transforms:value="0.03"/>
      <transforms:coefficient transforms:value="0"/>
    </transforms:spectrum>
    <transforms:spectrum>
      ...
    </transforms:spectrum>
    ...
  </transforms:spectrumMatrix>

  <gating:PolygonGate gating:id="myGate">
    <gating:dimension gating:transformation-ref="myLog"
                      gating:compensation-ref="myComp">
      <data-type:new-dimension data-type:transformation-ref="myRatio"/>
    </gating:dimension>
    <gating:dimension gating:transformation-ref="myHLog"
                      gating:compensation-ref="myComp">
      <data-type:fcs-dimension data-type:name="cFL2-A"/>
    </gating:dimension>
    <gating:vertex>
      <gating:coordinate data-type:value="0.9"/>
      <gating:coordinate data-type:value="0.5"/>
    </gating:vertex>
    <gating:vertex>
      <gating:coordinate data-type:value="1.2"/>
      <gating:coordinate data-type:value="0.6"/>
    </gating:vertex>
    <gating:vertex>
      <gating:coordinate data-type:value="1.1"/>
      <gating:coordinate data-type:value="0.8"/>
    </gating:vertex>
  </gating:PolygonGate>
</gating:Gating-ML>
```

3.9 Merging transformations

As detailed in section **??**, Gating-ML 2.0 transformations can be "shared" for multiple arguments (*i.e.*, FCS parameters). This is not the case for Gating-ML 1.5 or the Gating-ML implementation in R. Therefore, if multiple "equivalent" transformations are supposed to be written to Gating-ML 2.0, then these will be merged into a single transformation and all references will be updated accordingly in the Gating-ML 2.0 output. This behavior can be demonstrated on the following example:

```
> flowEnv <- new.env()
> logicle1 <- logicletGml2(parameters="FL1-H", T=10000, M=4.5, A=0, W=.5, "logicle1")
> logicle2 <- logicletGml2(parameters="FL2-H", T=10000, M=4.5, A=0, W=.5, "logicle2")
> lin1 <- lintGml2(parameters = "FL1-H", T = 10000, A = 0, "lin1")
> lin2 <- lintGml2(parameters = "FL2-H", T = 10000, A = 0, "lin2")
> rectG <- rectangleGate(filterId="rectG", "logicle1"=c(.1, .6), "lin2"=c(.2, .6))
> rectG@parameters <- new("parameters", list(logicle1, lin2))
> rangeG1 <- rectangleGate(filterId="rangeG1", "logicle2"=c(0.1, 0.5))
> rangeG1@parameters <- new("parameters", list(logicle2))
> rangeG2 <- rectangleGate(filterId="rangeG2", "lin1"=c(0.6, 0.9))
> rangeG2@parameters <- new("parameters", list(lin1))
> flowEnv[['rectG']] <- rectG
> flowEnv[['rangeG1']] <- rangeG1
> flowEnv[['rangeG2']] <- rangeG2
> write.gatingML(flowEnv)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gating:Gating-ML ...>
  ...
  <transforms:transformation transforms:id="lin1">
    <transforms:flin transforms:T="10000" transforms:A="0"/>
  </transforms:transformation>
  <transforms:transformation transforms:id="logicle1">
    <transforms:logicle transforms:T="10000" transforms:M="4.5"
      transforms:W="0.5" transforms:A="0"/>
  </transforms:transformation>
  <gating:RectangleGate gating:id="rangeG1">
    <gating:dimension gating:min="0.1" gating:max="0.5"
      gating:transformation-ref="logicle1" gating:compensation-ref="uncompensated">
      <data-type:fcs-dimension data-type:name="FL2-H"/>
    </gating:dimension>
  </gating:RectangleGate>
  <gating:RectangleGate gating:id="rangeG2">
    <gating:dimension gating:min="0.6" gating:max="0.9"
      gating:transformation-ref="lin1" gating:compensation-ref="uncompensated">
      <data-type:fcs-dimension data-type:name="FL1-H"/>
    </gating:dimension>
```

```
  </gating:RectangleGate>
  <gating:RectangleGate gating:id="rectG">
    <gating:dimension gating:min="0.1" gating:max="0.6"
      gating:transformation-ref="logicle1" gating:compensation-ref="uncompensated">
      <data-type:fcs-dimension data-type:name="FL1-H"/>
    </gating:dimension>
    <gating:dimension gating:min="0.2" gating:max="0.6"
      gating:transformation-ref="lin1" gating:compensation-ref="uncompensated">
      <data-type:fcs-dimension data-type:name="FL2-H"/>
    </gating:dimension>
  </gating:RectangleGate>
</gating:Gating-ML>
```

If you inspect the code, you will notice that we have defined 4 transformations: "logicle1", "logicle2", "lin1" and "lin2". The "logicle1" and "logicle2" are defined the same way except that "logicle1" is applied to "FL1-H" while "logicle2" is applied to "FL2-H". Similarly for "lin1" and "lin2". Further in the code, we define a rectangular gate in the "logicle1" and "lin2" dimensions, and two range gates in the "logicle2" and "lin1" dimensions, respectively. Due to the transformation merging, only the "logicle1" and "lin1" transformations are defined in the Gating-ML 2.0 output. The second dimension of "rectG" has been updated to reference the "lin1" transformation; however, it is correctly applied to "FL2-H". Similarly, "rangeG1" has been updated to reference the "logicle1" transformation applied to "FL2-H".

3.10 Example with unsupported pipelines

As explained in section **??**, not all pipelines expressible in R are expressible in Gating-ML 2.0. Below is an example of a pipeline involving compound scaling transformations – a Logicle transformation applied to another Logicle transformation. An error message saying that "Unexpected parameter class logicletGml2, compound transformations are not supported in Gating-ML 2.0." will be displayed if we try to save this in Gating-ML 2.0.

```
> logicle1 <- logicletGml2(parameters = "FL1-H", T = 1000, M = 4.5, A = 0,
+   W = 0.5, transformationId="logicle1")
> logicle2 <- logicletGml2(parameters = "logicle1", T = 1000, M = 4.5, A = 0,
+   W = 0.5, transformationId="logicle2")
> logicle2@parameters <- logicle1
> myRect <- rectangleGate(filterId="myRect", list("logicle2"=c(0, .6)))
> myRect@parameters <- new("parameters", list(logicle2))
> flowEnv[['myRect']] <- myRect
> x <- tryCatch(write.gatingML(flowEnv), error = function(e) { e })
> x$message

[1] "Unexpected parameter class logicletGml2, compound transformations are not
    supported in Gating-ML 2.0."
```

3.11 Example with unsupported transformations

R is a powerful programming language that allows for many different data transformations. However, only some data transformations are supported by the Gating-ML 2.0 specification (see table **??**). As shown below, an error is produced if an incompatible transformation is found in the environment that is being written to the Gating-ML 2.0 output.

```
> flowEnv <- new.env()
> tSS <- splitscale(parameters = "FL1-H", r = 1024, maxValue = 10000,
+     transitionChannel = 256, transformationId = "tSS")
> myRect <- rectangleGate(filterId="myRect", list("tss"=c(100, 700)))
> myRect@parameters <- new("parameters", list(tSS))
> flowEnv[['myRect']] <- myRect
> x <- tryCatch(write.gatingML(flowEnv), error = function(e) { e })
> x$message

[1] "Class 'splitscale' is not supported in Gating-ML 2.0 output. Only
    Gating-ML 2.0 compatible transformations are supported by Gating-ML 2.0
    output. Transformation 'tSS' is not among those and cannot be included.
    Therefore, any gate referencing this transformation would be referencing
    a non-existent transformation in the Gating-ML output. Please correct the
    gates and transformations in your environment and try again."
```

If this is the case, you will have to remove the transformation and any reference to it from the environment before being able to save the environment in Gating-ML 2.0.

3.12 Example with unsupported gate type

All widely used static gates are Gating-ML 2.0 compatible. Gates that are not compatible with Gating-ML 2.0 include n-dimensional polytope gates (the *polytopeGate* class) and decision tree gates (the *expressionFilter* class). These types of gates are supported by Gating-ML 1.5, but the support has been removed in Gating-ML 2.0 since these gates are almost never used for the analysis of flow cytometry data. In addition, Gating-ML 2.0 cannot be used to export data driven gate, such as *norm2Filter*, *kmeansFilter*, *curv1Filter*, *curv2Filter*, *boundaryFilter*, etc. As shown below, an error is produced if an incompatible gate is found in the environment that is supposed to be written to a Gating-ML 2.0 output.

```
> flowEnv <- new.env()
> # Gating-ML 1.5 example 5.3.4.c
> a <- matrix(c(-1, 0, 0, 0, -1, 0, 0, 0, -1, 1, 0, 0, 0, 0, 1), ncol=3)
> b <- c(100, 50, 0, 250, 300)
> myPolytope = polytopeGate(filterId='myPolytope', .gate=a, b=b,
+   list("FSC-H", "SSC-H", "FL1-H"))
> flowEnv[['myPolytope']] <- myPolytope
> x <- tryCatch(write.gatingML(flowEnv), error = function(e) { e })
> x$message
```

```
[1] "Class 'polytopeGate' is not supported in Gating-ML 2.0 output. Only
    Gating-ML 2.0 compatible gates are supported by Gating-ML 2.0 output.
    Filter 'myPolytope' is not among those and cannot be included. Please
    remove this filter and any references to it from the environment and try
    again."
```

If this is the case, you will have to remove the incompatible gate (filter) and any reference to it, including references from Boolean collections (*i.e.*, *intersectFilter*, *unionFilter* and *complementFilter*) and gating hierarchies (*subsetFilter*). A similar error will be produced if you try to include data driven gates in the Gating-ML output as shown below:

```
> flowEnv <- new.env()
> myNorm2Filter <- norm2Filter("FSC-H", "SSC-H", filterId="myNorm2Filter")
> flowEnv[['myNorm2Filter']] <- myNorm2Filter
> x <- tryCatch(write.gatingML(flowEnv), error = function(e) { e })
> x$message
```

```
[1] "Class 'norm2Filter' is not supported in Gating-ML 2.0 output. Only
    Gating-ML 2.0 compatible gates are supported by Gating-ML 2.0 output.
    Filter 'myNorm2Filter' is not among those and cannot be included. Please
    remove this filter and any references to it from the environment and try
    again."
```

4 Testing Gating-ML compliance

4.1 Additional requirements

The flowUtils package includes RUnit-based compliance tests to verify its compliance with the Gating-ML 1.5 and Gating-ML 2.0 specifications. You will need the gatingMLData package (version 2.6.0 or newer) in order to run the compliance tests. If you do not have this package, you can install it as follows:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("gatingMLData")
```

4.2 Gating-ML 1.5 compliance

Once you have the required gatingMLData package installed, the `testGatingMLCompliance` function can be used to test Gating-ML compliance. This function takes two arguments: the name of the file where the compliance HTML report shall be saved, and the version of Gating-ML that the compliance with shall be tested with. In order to test compliance with Gating-ML 1.5, you can run the following code:

```
> testGatingMLCompliance("ComplianceReport_v1.5.html", version=1.5)
```

This code will run 460 test functions, which are based on the 32 sets of compliance tests included with the Gating-ML 1.5 specification. During these tests, computed event membership

is compared against the events expected in each of the tested gates, and any discrepancies are reported as failures. The Gating-ML 1.5 compliance tests usually take about 1 – 2 minutes to complete, at which point an HTML report with 0 failures and 0 errors should be produced.

4.3 Gating-ML 2.0 compliance

The following code can be executed in order to test compliance with Gating-ML 2.0:

```
> testGatingMLCompliance("ComplianceReport_v2.0.html", version=2.0)
```

This code will execute 405 test functions and should take about 4 – 6 minutes to complete. It contains 11 sets of tests. Sets 1 and 2 are based on the two sets of compliance tests included with the Gating-ML 2.0 specification. During these tests, computed event membership is compared against the events expected in each of the tested gates, and any discrepancies are reported as failures. Sets 3, 4 and 5 implement additional compliance tests that were not included with the Gating-ML 2.0 specification. These allow us to test a few additional concepts that are not checked with the official tests (e.g., non-square spectrum matrices); however, we should note that we have used R to generate the expected results for these tests. Therefore, the tests can only ensure that flowUtils parses the provided Gating-ML 2.0 files properly and that the results remain consistent over time. The rest of the test sets is focused on writing Gating-ML 2.0. The first 5 "write Gating-ML" test sets are based on the mentioned "read Gating-ML" tests; however, we always

1. Read the Gating-ML file into an empty enviroment
2. Save that enviroment into temporary Gating-ML 2.0 file
3. Empty the enviroment
4. Read the saved temporary Gating-ML file
5. Check that we retrieved all the gates with all the expected results correctly

This way, we can also make sure that the Gating-ML files have been written correctly. The last set of "write Gating-ML" tests includes concepts that cannot be created by reading a Gating-ML 2.0 file; however, they can be created manually and exported to a Gating-ML 2.0 file. For example, this includes tests that the Gating-ML 1.5 ratio transformation can be saved in Gating-ML 2.0 and then retrieved as Gating-ML 2.0 "fratio" with the correct results. Additional tests include the "asinhtGml2" transformation with a directly embedded (rather than referenced) ratio transformation, the use of filters (rather than filter references) for Boolean gates, the proper conversion of Gating-ML 1.5 "asinht" to Gating-ML 2.0 "asinhtGml2", and also tests of various Quad gates in combination with compensation, ratio and scale transformations. An HTML report with 0 failures and 0 errors should be produced once the Gating-ML 2.0 compliance tests are completed.

5 Using Gating-ML to exchange gates with other software tools

To the best of our knowledge, R, Matlab, FlowRepository (**??**) and Cytobank (**?**) are the first Gating-ML compatible software tools. FlowJo (and other tools) also implemented most of Gating-ML, but are still working on adjusting some of the data transformations to achieve Gating-ML based interoperability.

5.1 Implicit FCS transformations

As mentioned in section **??**, the `transformation=FALSE` option should be used when applying Gating-ML 1.5, and the `transformation=linearize-with-PnG-scaling` option when applying Gating-ML 2.0. This is because Gating-ML 2.0 specifies that the "channel-to-scale" transformation shall be performed after reading the "channel" values from the FCS data file. The "channel-to-scale" transformation includes:

- The "linearization" of FCS parameters stored on a log scale, *i.e.,* with \$P*n*E values different from "0,0".
- The "correction" for gain of FCS parameters stored with \$P*n*G values different from "1".

The latter one means nothing more than the division of the parameter value by the appropriate \$P*n*G value. According to our experience, in the majority of cases, there are no \$P*n*G values in the FCS data files, and if these are present and different from "1", the data file has usually been produced by one of the older instruments. But, as this is just a linear transformation, it may be ignored by some analysis tools as data is scaled based on the size of the display with little meaning of the actual absolute expression values. However, this details is significant for being able to exchange gates using Gating-ML. Therefore, should you be working with data files with \$P*n*G values different from "1", and should you observe compatibility issues with third party software tools (gates of wrong sizes or in wrong positions), you may want to try reading your data with the `transformation=linearize` option.

5.2 Notes about precision

In R, data transformations and gates are expressed and calculated using a double-precision floating-point format, which leads to very "precise" results. Arguably, such high precision is not needed for the analysis of flow cytometry data and therefore, several other tools choose to implement lower precision solutions. Commonly, these tools incorporate a binning approach, where the full scale range is binned into a fixed number of bins (*e.g.*, 256, 1024), and gates are calculated based on this binning. Such an approach allows for faster calculations and gate membership determination. If this is the case, small differences are to be expected between populations calculated by R (which is "precise") and by other tools (which may be approximate). However, these differences should not be biologically significant since they are very small and typically, events in question will be very close to the border of a gate. This can be demonstrated on the following example. We have taken a randomly chosen gate (Figure **??**) from FlowRepository, exported the gate from FlowRepository using FlowRepository's Gating-ML 2.0 export, imported this Gating-ML file in R, and applied it to the same FCS data file that FlowRepository did. The "PE-A" channel is displayed on an ArcSinH scale.

```
> fcsFile  <- system.file("extdata/examples", "166889.fcs",
+     package = "gatingMLData")
> gateFile <- system.file("extdata/examples", "GatingML2.0_Export_166889.xml",
+     package = "gatingMLData")
> myFrame  <- read.FCS(fcsFile, transformation="linearize-with-PnG-scaling")
> flowEnv  <- new.env()
```

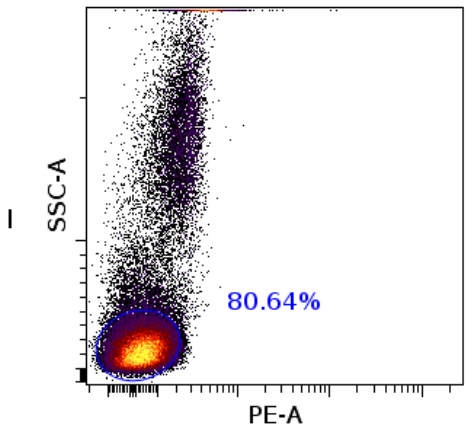

Figure 1: A screenshot from FlowRepository with an ellipse gate enclosing 80.64% of cells.

```
> read.gatingML(gateFile, flowEnv)
> for (x in ls(flowEnv)) if (is(flowEnv[[x]], "filter")) {
+     result <- filter(myFrame, flowEnv[[x]])
+     print(summary(result))
+ }

GateSet_1_UEUtQQ.._U1NDLUE.+: 49260 of 61178 events (80.52%)
Gate_1_UEUtQSBTUOMtQSBFMQ.._UEUtQQ.._U1NDLUE.+: 49260 of 61178 events (80.52%)
```

As you can see, FlowRepository is showing 80.64% of cells in that gate, while R calculated 80.52% only. This minor difference can be explained by the fact that FlowRepository incorporates binning (with 256 bins) in the gating calculations.

Additional very minor differences can be observed due to different "channel to scale" transformations implemented in different software tools. In R, we are using the Gating-ML 2.0 compatible formula that has been provided in FCS 3.1 (**?**). Specifically, for \$DATATYPE/I/, \$PnR/r/, $r > 0$, \$PnE/f_1,f_2/, $f_1 > 0$, $f_2 > 0$: n is a logarithmic parameter with channel values going from 0 to $r - 1$, and scale values ranging from f_2 to $f_2 * 10^{f_1}$. A channel value x_c can be converted to a scale value x_s as $x_s = 10^{f_1 * x_c / r} * f_2$. If $f_2 = 0$ then f_2 shall be considered as 1 and the same formula shall be applied. However, this formula has only been standardized recently and historically, some software tools use a different calculation, namely $x_s = 10^{f_1 * x_c/(r-1)} * f_2$. Differences between using $r - 1$ *vs.* r in the formula may lead to minor differences in the resulting gating; however, these should be very minor and not significant biologically.

5.3 Notes about gate and population identifiers

As you may have noticed, two gates have been parsed from the Gating-ML file, an *ellipsoidGate* and an *intersectFilter* gate.

```
> class(flowEnv[['Gate_1_UEUtQSBTUOMtQSBFMQ.._UEUtQQ.._U1NDLUE.']])
```

```
[1] "ellipsoidGate"
attr(,"package")
[1] "flowCore"
```

```
> str(flowEnv[['Gate_1_UEUtQSBTUOMtQSBFMQ._UEUtQQ._U1NDLUE.']])
```

```
Formal class 'ellipsoidGate' [package "flowCore"] with 5 slots
  ..@ mean      : Named num [1:2] 1.31e-01 2.61e+04
  .. ..- attr(*, "names")= chr [1:2] "Tr_Arcsinh.FCS.PE-A" "SSC-A"
  ..@ cov       : num [1:2, 1:2] 1.09 4.01e+03 4.01e+03 6.20e+08
  ..@ distance  : num 1
  ..@ parameters:Formal class 'parameters' [package "flowCore"] with 1 slot
  .. .. ..@ .Data:List of 2
  .. .. .. ..$ :Formal class 'transformReference' [package "flowCore"] with 3 slots
  .. .. .. .. .. ..@ .Data           :function ()
  .. .. .. .. .. ..@ searchEnv       :<environment: 0x7ff8bc8be5f8>
  .. .. .. .. .. ..@ transformationId: chr "Tr_Arcsinh.FCS.PE-A"
  .. .. .. ..$ :Formal class 'compensatedParameter' [package "flowCore"] with 5 slots
  .. .. .. .. .. ..@ .Data           :function ()
  .. .. .. .. .. ..@ parameters      : chr "SSC-A"
  .. .. .. .. .. ..@ spillRefId      : chr "SpillFromFCS"
  .. .. .. .. .. ..@ searchEnv       :<environment: 0x7ff8bc8be5f8>
  .. .. .. .. .. ..@ transformationId: chr "SSC-A_compensated_according_to_FCS"
  ..@ filterId  : chr "Gate_1_UEUtQSBTUOMtQSBFMQ._UEUtQQ._U1NDLUE."
```

```
> class(flowEnv[['GateSet_1_UEUtQQ._U1NDLUE.']])
```

```
[1] "intersectFilter"
attr(,"package")
[1] "flowCore"
```

```
> str(flowEnv[['GateSet_1_UEUtQQ._U1NDLUE.']])
```

```
Formal class 'intersectFilter' [package "flowCore"] with 2 slots
  ..@ filters :List of 2
  .. ..$ :Formal class 'filterReference' [package "flowCore"] with 3 slots
  .. .. .. ..@ name    : chr "Gate_1_UEUtQSBTUOMtQSBFMQ._UEUtQQ._U1NDLUE."
  .. .. .. ..@ env     :<environment: 0x7ff8bc8be5f8>
  .. .. .. ..@ filterId: chr "Gate_1_UEUtQSBTUOMtQSBFMQ._UEUtQQ._U1NDLUE."
  .. ..$ :Formal class 'filterReference' [package "flowCore"] with 3 slots
  .. .. .. ..@ name    : chr "Gate_1_UEUtQSBTUOMtQSBFMQ._UEUtQQ._U1NDLUE."
  .. .. .. ..@ env     :<environment: 0x7ff8bc8be5f8>
  .. .. .. ..@ filterId: chr "Gate_1_UEUtQSBTUOMtQSBFMQ._UEUtQQ._U1NDLUE."
  ..@ filterId: chr "GateSet_1_UEUtQQ._U1NDLUE."
```

This relates to how FlowRepository exports Gating-ML. We can inspect the Gating-ML file using the following command:

```
> cat(readChar(gateFile, file.info(gateFile)$size))

<?xml version="1.0" encoding="UTF-8"?>
<gating:Gating-ML ...">
 <data-type:custom_info> ... </data-type:custom_info>
 <transforms:transformation transforms:id="Tr_Arcsinh">
  <transforms:fasinh transforms:T="176.2801790465702"
                     transforms:M="0.43429448190325176" transforms:A="0.0" />
 </transforms:transformation>
 <gating:EllipsoidGate gating:id="Gate_1_UEUtQSBTUOMtQSBFMQ.._UEUtQQ.._U1NDLUE.">
  <data-type:custom_info> ... </data-type:custom_info>
  <gating:dimension gating:compensation-ref="FCS"
                    gating:transformation-ref="Tr_Arcsinh">
   <data-type:fcs-dimension data-type:name="PE-A" />
  </gating:dimension>
  <gating:dimension gating:compensation-ref="FCS">
   <data-type:fcs-dimension data-type:name="SSC-A" />
  </gating:dimension>
  <gating:mean>
   <gating:coordinate data-type:value="0.13093575523900436" />
   <gating:coordinate data-type:value="26112.900390625" />
  </gating:mean>
  <gating:covarianceMatrix>
   <gating:row>
    <gating:entry data-type:value="1.0941582172399216" />
    <gating:entry data-type:value="4008.453938328732" />
   </gating:row>
   <gating:row>
    <gating:entry data-type:value="4008.453938328732" />
    <gating:entry data-type:value="6.198370677161704E8" />
   </gating:row>
  </gating:covarianceMatrix>
  <gating:distanceSquare data-type:value="1.0" />
 </gating:EllipsoidGate>
 <gating:BooleanGate gating:id="GateSet_1_UEUtQQ.._U1NDLUE.">
  <data-type:custom_info> ... </data-type:custom_info>
  <gating:and>
   <gating:gateReference gating:ref="Gate_1_UEUtQSBTUOMtQSBFMQ.._UEUtQQ.._U1NDLUE." />
    <!-- Boolean "and" gates are used to describe FlowRepository's populations
         (GateSets). Here, we only have one gate defining the population, but
         Gating-ML requires at least two arguments for the "and" gate. Therefore,
         we are referencing the same gate twice. -->
   <gating:gateReference gating:ref="Gate_1_UEUtQSBTUOMtQSBFMQ.._UEUtQQ.._U1NDLUE." />
  </gating:and>
 </gating:BooleanGate>
```

</gating:Gating-ML>

There are two different concepts in FlowRepository: gates and populations (also called "GateSets"). A population is defined as the intersection of one or more gates. In Gating-ML, every gate defines a population, and there is the option of combining gates into more complicated structures using the Boolean "AND", "OR" and "NOT" operators. Consequently, FlowRepository exports a Boolean "AND" gate for every population defined. If this population is defined by a single gate, then this gate will be listed twice in the operands of the Boolean "AND" gate in order to satisfy Gating-ML's requirement of 2 or more arguments for the Boolean "AND" and "OR" gates. This explains why there is the `GateSet_1_UEUtQQ.._U1NDLUE.` *intersectFilter* filter in our environment, and why it references the ellipsoid gate twice (which has no effect on the calculated result). Also, please note that if you export gates from R using Gating-ML 2.0 and import these in FlowRepository, the gates will get imported but the populations will not be defined automatically. You will need to open the population manager within FlowRepository in order to specify which combinations of gates are "meaningful" in terms of defining a useful population.

Finally, you may have noticed that the gate identifiers are long and not human readable. In Gating-ML, there is no standardized way of describing a "name" of a gate. There is a standard way of describing a gate identifier; however, the syntax of this identifier has to conform to the syntax of XML identifiers. Therefore, FlowRepository uses a modified Base64 encoding of gate names to create XML compatible gate identifiers.

We are using a different approach to ensure that the gate identifiers are XML compatible when exporting Gating-ML from R. This approach is based on replacing "illegal" characters with characters that are allowed to be part of an XML identifier. We recommend using regular ASCII characters for gate identifiers in R if you wish to keep these unchanged in your Gating-ML 2.0 export.