Generic QA Process Functions: a Use Case

Radina Droumeva

March 27, 2012

This is a sample use case for generating simple `qaProcess` objects that can be incorporated into an HTML report about a single flow set. Let us first load the necessary libraries, and use the dataset qData – a list of 8 flow sets. We will use only the first flow set:

```
> library(flowCore)
> library(flowQ)
> data(qData)
> fs <- qData[[1]]
> fs[[2]] <- fs[[2]][1:1000]
> fs[[3]] <- fs[[3]][1:1500]
```

```
> fs

A flowSet with 4 experiments.

An object of class "AnnotatedDataFrame"
  rowNames: pid02050, pid04030, pid01027, pid02057
  varLabels and varMetadata description:
    Studynum:
    GroupID:  Group Identifier
    ...: ...
    FCS_File:  Name of FCS file
    (8 total)

  column names:
  FSC-A SSC-A FITC-A PE-A FL3-A PE-Cy7-A APC-A Time
```

Suppose we want to count the number of cells in each frame and also generate plots of FSC vs. SSC for each frame, and put these in an HTML report. The files `qaProcess.GenericNumber.R` and `qaProcess.GenericImages.R` contain the necessary functions.

1 Generic Number QA Procedure

The function `qaProcess.GenericNumber` takes as input the following:

- `numbers`: a vector of numbers which will be displayed in a column of the report
- `frameIDs`: a vector of strings which correspond to each of the frames (should match `sampleNames(flowset)`)
- `outdir`: the QA report directory
- `name`: your choice of string describing the QA process
- `cutoff` (default is `-Inf`): a numeric value, such that the numbers below this value will be displayed in red, indicating a "fail" result of the respective `qaProcessFrame` object.

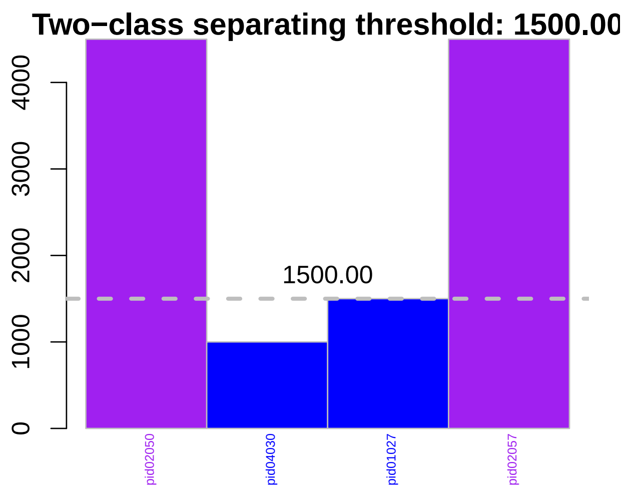
- `twoclass` (default is `FALSE`): if set to TRUE, kmeans is used to group the numbers into two groups, and the summary image will colour-code them, as well as provide a threshold cutoff value for the separation. Note that the threshold value favours the smaller numbers – it is the maximum value amongst the group with the smaller mean.

It returns a `qaProcess` object ready to be used to generate an HTML report. Let us define each of these parameters and call the function:

```
> numbers <- as(fsApply(fs, nrow), "vector")
> frameIDs <- sampleNames(fs)
> my.dir <- getwd()
> outdir <- paste(my.dir, "QAreport", sep = .Platform$file.sep)
> name <- "Total Cell Number"
> source("qaProcess.GenericNumber.R")
> qa.total.number <- qaProcess.GenericNumber(numbers = numbers,
+     frameIDs = frameIDs, outdir = outdir, name = name, twoclass = TRUE)
> qa.total.number
```

```
Quality process 'Total Cell Number' of type 'number'
```

This object can now be passed to `writeQAReport` directly. Notice that the `numbers` can be anything – total raw cell counts, total viable cell counts, percentages, etc. Also note that `frameIDs` should match the annotation of the flowset, in order for the HTML report to be built properly. As an additional feature, this function uses k-means to separate the numbers into two groups if `twoclass = TRUE`, and generates a summary image composed of a barplot of the numbers. Different colours indicate different k-means clusters – this is simply to visualize the spread of values. For example, if the numbers represent percent positive cells, and the samples tend to be either very low positive percent or very high positive percent, it may be useful to visualize the separation and the threshold value dividing those two classes of cells. The graphic below illustrates the raw cell counts.



2 Generic Images QA Procedure

Next, we will use `qaProcess.GenericImages` function to generate a similar QA process, but using images instead of numbers. The function takes the following inputs:

- `filenames`: a vector containing full file paths to the images to be used
- `frameIDs`: a vector of strings which correspond to each of the frames (should match `sampleNames(flowset)`)

- `outdir`: the QA report directory
- `name`: your choice of string describing the QA process
- `flags` (default is `NULL`): a vector of frameIDs which should "fail" the QA process, and will show up with a red bubble beside the image in the HTML report
- `width` (default is 200): width for the HTML images. The html image extension defaults to the extension of the first image in `filenames`.
- `pdf` (default is `FALSE`): indicates whether a .pdf version of the images should be generated. If `TRUE`, then also set `pdf = TRUE` in the call to `writeQAReport` to make the images in the report clickable, with their .pdf versions as destination.
- summary.graph (default is `NULL`): an optional file path for this QA process's summary image. If `NULL`, the first image in filenames is used.

It returns a `qaProcess` object. Again note that `frameIDs` should match the flowset, as well as the other `qaProcess` objects. It is also important to give your QA process a unique `name`. Each QA process gets a summary image, which is saved in a folder with the name "name." If this is not unique for different QA procedures, then it will cause confusion in the final HTML report. First, some plots need to be generated and saved:

```
> image.names <- paste(sampleNames(fs), "Plot.jpeg", sep = "")
> dir.create(paste(my.dir, "images", sep = .Platform$file.sep))
> filenames <- paste(my.dir, "images", image.names, sep = .Platform$file.sep)
> for (i in 1:length(filenames)) {
+     jpeg(filename = filenames[i])
+     par(cex = 1.5, mar = c(3, 3, 2, 1), mgp = c(2, 0.75, 0))
+     plot(exprs(fs[[sampleNames(fs)[i]]])[, c("FSC-A", "SSC-A")],
+         pch = ".", main = sampleNames(fs)[i])
+     dev.off()
+ }
```

Now the images are created, and `filenames` has been defined. The parameters `frameIDs` and `outdir` remain as defined, so we only need to come up with a name for this QA procedure. For demonstration purposes, suppose we wish to flag the third image and then call the function:

```
> flags <- sampleNames(fs)[3]
> source("qaProcess.GenericImages.R")
> qa.images <- qaProcess.GenericImages(filenames = filenames, frameIDs = frameIDs,
+     outdir = outdir, name = "FSC-SSC Plots", flags = flags)
> qa.images
```

```
Quality process 'FSC-SSC Plots' of type 'images'
```

Now we have our second `qaProcess` object. The images, again, can be of anything – perhaps illustrating a step of the automated gating procedure. A flag could, for example, signal some deviation of a sample's density of FSC as compared to other samples, or a high percentage of positive cells of certain type, etc.

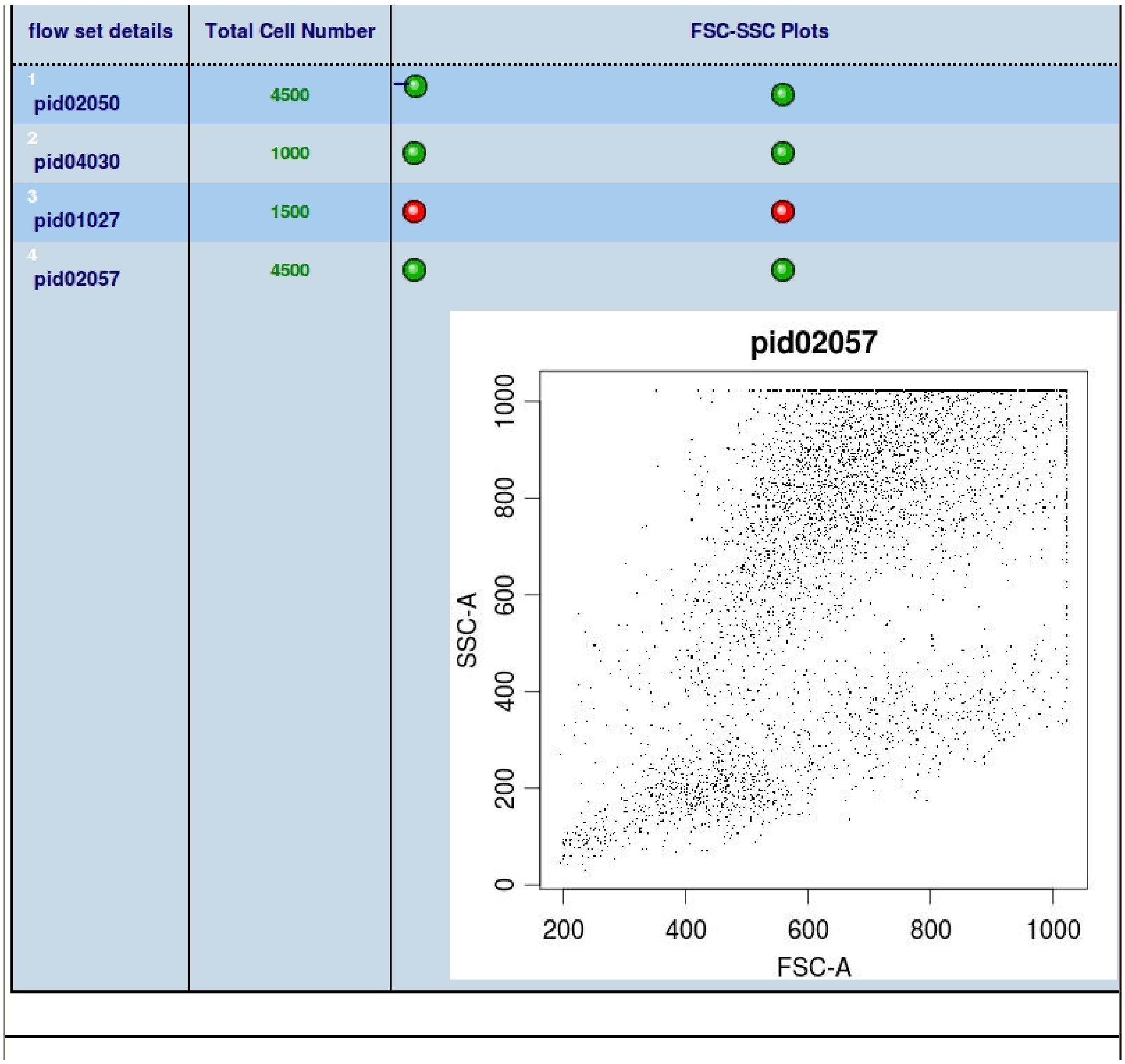
3 Simple HTML report

Finally, the HTML report can be created. `flowQ` has the documentation on the relevant function, `writeQAReport`.

```
> url <- writeQAReport(set = fs, processes = list(qa.total.number,
+     qa.images), outdir = outdir)
> url
```

```
[1] "/home/rdroumeva/Documents/Sweave/flowQsupport/QAreport/index.html"
```

Figure 1 shows a snapshot of the html report generated.



flow set details	Total Cell Number	FSC-SSC Plots	
1 pid02050	4500	🟢	🟢
2 pid04030	1000	🟢	🟢
3 pid01027	1500	🔴	🔴
4 pid02057	4500	🟢	🟢

Figure 1: Simple HTML report.

4 Helper Functions

4.1 Generic Number Quality Assurance Function

```
qaProcess.GenericNumber <- function(numbers, frameIDs, outdir, name="generic",
                                    cutoff=-Inf, twoclass=FALSE)
{
# This function generates a qaProcess object using numbers, which can be used to generate
# an HTML report about a flowset
# Args:
#   numbers: a vector of numbers which will be displayed in a column of the report
#   frameIDs: a vector of strings which correspond to each of the frames
#             (should match sampleNames(flowset))
#   outdir: the QA report directory
#   name: your choice of string describing the QA process
#   cutoff (default -Inf): a numeric value, such that the numbers below this value will be
#       displayed in red, indicating a "fail"
#   twoclass (defaultFALSE): if TRUE, kmeans will be applied to the numbers to separate them
#       into two arbitrary groups (e.g. positive/negative classes) and different
#       colours will be used in the summary plot. The threshold value, taken as the
#       maximum value in the lower class, will also be plotted in the summary plot.
# Value:
#   returns a qaProcess object
#

  if(!is.null(dim(numbers)) || !is.null(dim(frameIDs)))
    stop("numbers and frameIDs must be vectors")

  if(length(numbers) != length(frameIDs))
    stop("Number of numeric values passed and number of frame IDs do not match")

  if(!file.exists(outdir))
    dir.create(outdir, recursive=TRUE)
  names(numbers) <- frameIDs

  # Create directory for images to be used in the HTML report, as well as a
  summary image for this QA process:
  unique.id <- format.hexmode(as.integer(Sys.time())/runif(1)*proc.time()["elapsed"])
  image.dir <- paste(outdir, gsub(" ", "_", name), "/", sep="")
  dir.create(image.dir, recursive=TRUE, showWarnings=FALSE)
  summary.file <- file.path(image.dir, "summary.pdf")

  # Use k-means to define two arbitrary classes of numbers
  # (e.g. if you have really low positive counts vs. really high positive counts,
  # you may be interested in the cut off value or the visualization only)
  # A bar plot with blue and purple bars will display the two classes.
  colours <- rep("blue", length(numbers))
  if(twoclass && length(unique(numbers)) > 1)
  {
    km <- kmeans(x = numbers, centers = 2, nstart = 50)
    indices <- km$cluster
    threshold <- min(max(numbers[which(indices == 1)]), max(numbers[which(indices == 2)]))
    smaller.cluster <- which.min(km$centers[,1])
    bigger.cluster <- which.max(km$centers[,1])
    colours[which(indices == smaller.cluster)] <- "blue"
    colours[which(indices == bigger.cluster)] <- "purple"
  }
  colours[which(numbers < cutoff)] <- "red"

  # Generate and save the summary image:
  pdf(file = summary.file, width = max(5,length(numbers)/10), height=4)
  par(cex = 1.2, mar = c(3, 3, 1, 1))
  b <- barplot(numbers, border = "gray", col = colours, names.arg = frameIDs,
          cex.names = 0.5, ylab=name, space = 0, axisnames=FALSE, xaxt="n")
  text(b, 0, labels = frameIDs, srt=90,  xpd=TRUE, cex=.5, pos=1, offset = 1, col=colours)
  if(twoclass && length(unique(numbers)) > 1)
  {
    abline(h = threshold, lwd = 3, lty = "dashed", col = "gray")
    text(2, threshold, sprintf("%.2f", threshold), pos=3, offset=0.5)
    title(main = paste("Two-class separating threshold: ",
          sprintf("%.2f", threshold), sep = ""))
  } else {
    title(main = paste("Average: ", sprintf("%.2f", mean(numbers)), sep = ""))
    abline(h = mean(numbers), lwd = 3, lty = "dashed", col = "gray")
  }
  dev.off()
  summary.graph <- qaGraph(fileName = summary.file, imageDir = image.dir)

  # Formally define individual qaProcessFrame objects. Note the use of a numeric aggregator
  # with a "passed" slot defined by comparing a value to the cutoff value. If using the default
  # cutoff = -Inf, all frames will "pass."
  frame.processes <- lapply(frameIDs,
    function(i)
    {
      qaProcessFrame(i, summaryAggregator = numericAggregator(numbers[i] > cutoff, x = numbers[i]))
    })
  names(frame.processes) <- frameIDs

  # Compile all the qaFrameProcess objects into one qaProcess object and return:
  qa.process <- qaProcess(id = unique.id, name = name, type = "number",
                          summaryGraph = summary.graph, frameProcesses=frame.processes)
  return(qa.process)
}
```

4.2 Generic Image Quality Assurance Function

```
qaProcess.GenericImages <- function(
  filenames,
  frameIDs,
  outdir="QAReport",
  name="Gating Images",
  flags = NULL,
  width=200,
  pdf = FALSE,
  summary.graph = NULL,
  ...)
{
# This function generates a qaProcess object using images, which can be used to generate
# an HTML report about a flowset
# Args:
#   filenames: a vector of paths to images which will be displayed in a column of the report
#   frameIDs: a vector of strings which correspond to each of the frames
#      (should match sampleNames(flowset))
#   outdir: the QA report directory
#   name: your choice of string describing the QA process
#   flags: a vector of frameIDs corresponding to frames that should "fail" this QA procedure
#   width (default is 200): width for the HTML images. The html image extension defaults
#     to the extension of the first image in filenames, or to "jpeg" if the extension
#     is not one of png, jpeg, pdf or eps
#   summary.graph: an optional file path for this QA process's summary image. If NULL,
#     the first image in filenames is
#      used.
# Value:
#   returns a qaProcess object
#

  if(!is.null(dim(filenames)) || !is.null(dim(frameIDs)))
    stop("filenames and frameIDs must be vectors")

  if(length(filenames) != length(frameIDs))
    stop("Number of images passed and number of frame IDs do not match")

  # Assess image file extension
  file.extension <- strsplit(filenames[1], "\\.")[[1]]
  file.extension <- file.extension[length(file.extension)]
  if(!is.element(file.extension, c("png", "jpeg", "pdf", "eps")))
    file.extension <- "jpeg"

  # Generate new directory for the images to be associated with the HTML report later
  unique.id <-format.hexmode(as.integer(Sys.time())/runif(1)*proc.time()["elapsed"])

  frame.names <- frameIDs
  num.frames <- length(frame.names)

  # Create a summary graph by copying the first image in the filenames vector
  summary.graph <- ifelse(is.null(summary.graph), filenames[1], summary.graph)

  super.dir <- unlist(strsplit(summary.graph, "/"))
  super.dir <- paste(paste(super.dir[-length(super.dir)], collapse="/"), "/", sep="")
  if (file.extension == "pdf")
  {
    system(paste('rm ', super.dir, "*.jpg", sep=""))
  }
  summary.qa.graph <- qaGraph(fileName = summary.graph,
                              imageDir = super.dir,
                            width    = width, pdf=pdf)

  # Generate individual qaProcessFrame objects:
  frameProcesses <- list()
  counter <- 0
  bar <- txtProgressBar(min = 0, max = length(viable.fs), style = 3)
  super.dir <- unlist(strsplit(filenames[1], "/"))
  super.dir <- paste(paste(super.dir[-length(super.dir)], collapse="/"), "/", sep="")
  for(i in seq_len(num.frames))
  {
    file.names <- NULL
    aggregator.list <- aggregatorList()
    frame.id <- frame.names[i]

    # check if the frame id was in the "flags" vector
    pass <- ifelse(is.element(frame.id, flags), FALSE, TRUE)
    # create a binary aggregator (pass or fail) associated with the image
    aggregator.list[[1]] <- binaryAggregator(pass)
    # create a summary aggregator associated with the frame overall
    summary.aggregator <- binaryAggregator(pass)

    # Create the necessary flowQ object to store the image information:
    fGraphs <- qaGraphList(imageFiles = filenames[i], imageDir = super.dir,
                           width = width, pdf = pdf)
    # Create individual qaProcessFrame objects:
    frameProcesses[[frame.id]] <- qaProcessFrame(frameID = frame.id,
                                     summaryAggregator = summary.aggregator,
                                     frameAggregators  = aggregator.list,
                                     frameGraphs       = fGraphs)
    counter <- counter + 1
    setTxtProgressBar(bar, counter)
  }

  # Finally compile everything in a single qaProcess object and return:
  output <- qaProcess(id             = unique.id,
                      name           = name,
                      type           = "images",
                      summaryGraph   = summary.qa.graph,
                      frameProcesses = frameProcesses)
  return(output)
}
```