Sample App - Mean Q-score per cycle

Adrian Alexa

March 12, 2013
aalexa@illumina.com

Contents

1 Introduction

This document describes a simple working session using `BaseSpaceR`. For more details on the functions used throughout this document please see the package main vignette - BaseSpaceR.

A typical BaseSpace session can be divided into the following steps:

- Client authentication
- Data retrieval and access
- Data processing
- Uploading results back to BaseSpace

In this sample session we'll show how to explore the project and sample data, select the FASTQ file(s) associated with the chosen sample, compute and plot Q-score statistics at each cycle, create an `AppResults` instance and link the generated figure to it.

```
> library(BaseSpaceR)
```

2 Authentication

The first step is the authentication of the client application with the BaseSpace REST server. The authentication and communication between the client and the server is handled by an `AppAuth` instance. There are two ways to create an `AppAuth` instance. The easiest is to provide a pre-generated access token, see help for `AppAuth` and the main vignette for more details on this. Assuming the user has access to such a token, stored as a character string in `app_access_token` variable, the handler can be created as follows:

```
> aAuth <- AppAuth(access_token = app_access_token)
> aAuth

Object of class "AppAuth" with:

Client Id:
Client Secret:

Server
URL:      https://api.basespace.illumina.com
Version:  v1pre3

Authorized:      TRUE
```

If the access token is valid then the connection with the server is established and we can see it by printing the `AppAuth` object (Authorized: TRUE).

One can also use the OAuth2 authentication process, but in this case user interaction is required. Authentication through the OAuth2 process is out of the scope of this document and we'll not show it here.

Please note that for the purpose of this document one needs an access token with the following scope: `create projects, write global`

3 Accessing the data

Now that our client App is authenticated with the server we can start exploring the data. One of the first steps is to inspect the resources available to the user under the current scope (associated with the access token).

3.1 Browsing available projects

Let's assume we need to browse all the projects accessible to the user. We can do this using the `listProjects()` method.

```
> myProj <- listProjects(aAuth)
> myProj
```

```
ProjectsSummary object:
Collection with 3 projectItem objects (out of a total of 3 objects).
{
        "Id" : "2",
        "Name" : "BaseSpaceDemo",
        "Href" : "v1pre3/projects/2",
        "DateCreated" : "2012-08-19T21:14:57.0000000",
        "UserOwnedBy" : {
                "Id" : "1001",
                "Href" : "v1pre3/users/1001",
                "Name" : "Illumina Inc",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/0646200388a465f27694d67b7aaea7cc
        }
}
{
        "Id" : "12",
        "Name" : "ResequencingPhixRun",
        "Href" : "v1pre3/projects/12",
        "DateCreated" : "2012-08-19T21:14:57.0000000",
        "UserOwnedBy" : {
                "Id" : "1001",
                "Href" : "v1pre3/users/1001",
                "Name" : "Illumina Inc",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/0646200388a465f27694d67b7aaea7cc
        }
}
{
        "Id" : "148150",
        "Name" : "My Project X",
        "Href" : "v1pre3/projects/148150",
        "DateCreated" : "2012-11-08T13:20:30.0000000",
        "UserOwnedBy" : {
                "Id" : "660666",
                "Href" : "v1pre3/users/660666",
                "Name" : "Adrian Alexa",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/2ca61f7edbd417b45780c8f849622559
        }
}
```

For this instance there are 3 projects available to the user. From the returned object we can get the names and ids of the available projects.

```
> data.frame(Name = Name(myProj), Id = Id(myProj))
```

```
                 Name     Id
1       BaseSpaceDemo      2
2 ResequencingPhixRun     12
4        My Project X 148150
```

For each available project we can get more detailed information. Let's assume we are interested in project *BaseSpaceDemo* (with `Id = 2`) for which we want to analyze the Q-score distribution of the reads. We can select it using the `Projects()` method and the ID of the project(s) of interest.

```
> selProj <- Projects(aAuth, id = 2, simplify = TRUE)
> selProj

Projects object:
{
        "HrefSamples" : "v1pre3/projects/2/samples",
        "HrefAppResults" : "v1pre3/projects/2/appresults",
        "Id" : "2",
        "Name" : "BaseSpaceDemo",
        "Href" : "v1pre3/projects/2",
        "DateCreated" : "2012-08-19T21:14:57.0000000",
        "UserOwnedBy" : {
                "Id" : "1001",
                "Href" : "v1pre3/users/1001",
                "Name" : "Illumina Inc",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/0646200388a465f27694d67b7aaea7cc
        },
        "HrefBaseSpaceUI" : "https://basespace.illumina.com/project/2/BaseSpaceDemo"
}
```

3.2 Selecting samples

Once we have decided the project we want to work on, we can browse the samples associated with it. We can list all the samples or we can limit the search to a specific number of samples. This can be achieved using the `Limit` query parameter.

```
> sampl <- listSamples(selProj, Limit = 1)
> sampl

SamplesSummary object:
Collection with 1 sampleItem objects (out of a total of 12 objects).
{
        "SampleId" : "BC_1",
        "Id" : "16018",
        "Name" : "BC_1",
        "Href" : "v1pre3/samples/16018",
        "DateCreated" : "2012-01-14T03:04:36.0000000",
        "UserOwnedBy" : {
                "Id" : "1001",
                "Href" : "v1pre3/users/1001",
                "Name" : "Illumina Inc",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/0646200388a465f27694d67b7aaea7cc
        },
        "Status" : "Complete"
}
```

The same result can be achieved by querying directly the `aAuth` handler and specifying the project ID.

```
> listSamples(aAuth, projectId = Id(selProj), Limit = 1)

SamplesSummary object:
Collection with 1 sampleItem objects (out of a total of 12 objects).
{
        "SampleId" : "BC_1",
        "Id" : "16018",
        "Name" : "BC_1",
        "Href" : "v1pre3/samples/16018",
        "DateCreated" : "2012-01-14T03:04:36.0000000",
        "UserOwnedBy" : {
                "Id" : "1001",
                "Href" : "v1pre3/users/1001",
                "Name" : "Illumina Inc",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/0646200388a465f27694d67b7aaea7cc
        },
        "Status" : "Complete"
}
```

There are a total of 12 samples associated with this project, and the listed sample (given that we restricted the result to one sample) has the ID 16018. We can get more information on this sample by calling the `Samples()` method with the `aAuth` handler and the sample ID as argument or just by calling it using the `sampl` object.

```
> inSample <- Samples(aAuth, id = Id(sampl), simplify = TRUE)
> identical(inSample, Samples(sampl, simplify = TRUE))

[1] TRUE

> inSample

Samples object:
{
        "SampleNumber" : 1,
        "ExperimentName" : "BacillusCereus",
        "HrefFiles" : "v1pre3/samples/16018/files",
        "AppSession" : {
                "Id" : "f5c152df3fb41dbf9dac94cc7d7b0ffa",
                "Href" : "v1pre3/appsessions/f5c152df3fb41dbf9dac94cc7d7b0ffa",
                "UserCreatedBy" : {
                        "Id" : "1001",
                        "Href" : "v1pre3/users/1001",
                        "Name" : "Illumina Inc",
                        "GravatarUrl" : "https://secure.gravatar.com/avatar/0646200388a465f27694d67b7
                },
                "Status" : "Complete",
                "StatusSummary" : "",
                "DateCreated" : "2012-01-14T01:02:07.0000000"
        },
        "IsPairedEnd" : true,
        "Read1" : 151,
        "Read2" : 140,
        "NumReadsRaw" : 1143322,
        "NumReadsPF" : 1117752,
        "SampleId" : "BC_1",
        "Id" : "16018",
        "Name" : "BC_1",
        "Href" : "v1pre3/samples/16018",
        "DateCreated" : "2012-01-14T03:04:36.0000000",
        "UserOwnedBy" : {
                "Id" : "1001",
                "Href" : "v1pre3/users/1001",
                "Name" : "Illumina Inc",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/0646200388a465f27694d67b7aaea7cc
        },
        "Status" : "Complete"
}
```

As we can see, the `inSample` object contains detailed information on the selected sample. For example, the `NumReadsRaw` entry contains the number of read pairs available in the chosen sample. We can access each entry in the Samples resource using the '`$`' operator. Please note that the access operator '`$`', works for every `Response` object.

```
> inSample$NumReadsRaw
```

```
[1] 1143322
```

3.3 Selecting and accessing files

We can now access the files linked to the selected sample. To browse the files we use the `listFiles()` function. We are interested in the FASTQ files.

```
> f <- listFiles(inSample, Extensions = ".gz")
> length(f)
```

```
[1] 6
```

```
> f
```

```
FilesSummary object:
Collection with 6 fileItem objects (out of a total of 6 objects).
{
        "Size" : 7493990,
        "Path" : "data/intensities/basecalls/s_G1_L001_I1_001.fastq.1.gz",
        "ContentType" : "application/octet-stream",
        "Id" : "535642",
        "Name" : "s_G1_L001_I1_001.fastq.1.gz",
        "Href" : "v1pre3/files/535642",
        "DateCreated" : "2012-01-14T03:04:36.0000000"
}
{
        "Size" : 7525743,
        "Path" : "data/intensities/basecalls/s_G1_L001_I1_002.fastq.1.gz",
        "ContentType" : "application/octet-stream",
        "Id" : "535643",
        "Name" : "s_G1_L001_I1_002.fastq.1.gz",
        "Href" : "v1pre3/files/535643",
        "DateCreated" : "2012-01-14T03:04:36.0000000"
}
{
        "Size" : 43480033,
        "Path" : "data/intensities/basecalls/s_G1_L001_R1_001.fastq.1.gz",
        "ContentType" : "application/octet-stream",
        "Id" : "535644",
        "Name" : "s_G1_L001_R1_001.fastq.1.gz",
        "Href" : "v1pre3/files/535644",
        "DateCreated" : "2012-01-14T03:04:36.0000000"
}
{
        "Size" : 43895096,
        "Path" : "data/intensities/basecalls/s_G1_L001_R1_002.fastq.1.gz",
        "ContentType" : "application/octet-stream",
        "Id" : "535645",
        "Name" : "s_G1_L001_R1_002.fastq.1.gz",
        "Href" : "v1pre3/files/535645",
        "DateCreated" : "2012-01-14T03:04:36.0000000"
}
{
        "Size" : 42679020,
        "Path" : "data/intensities/basecalls/s_G1_L001_R2_001.fastq.1.gz",
        "ContentType" : "application/octet-stream",
        "Id" : "535646",
        "Name" : "s_G1_L001_R2_001.fastq.1.gz",
        "Href" : "v1pre3/files/535646",
        "DateCreated" : "2012-01-14T03:04:36.0000000"
}
{
        "Size" : 43104726,
        "Path" : "data/intensities/basecalls/s_G1_L001_R2_002.fastq.1.gz",
        "ContentType" : "application/octet-stream",
        "Id" : "535647",
        "Name" : "s_G1_L001_R2_002.fastq.1.gz",
        "Href" : "v1pre3/files/535647",
        "DateCreated" : "2012-01-14T03:04:36.0000000"
}
```

The R API is design such that the user can use any query parameters implemented by the REST API. Here we selected just the *.gz files, which in the context of the `Samples` resource these are only FASTQ files.

The `Path` element gives us the relative location of the file(s). We can use this to uniquely identify the files.

```
> ## full location of the files
> f$Path
```

```
[1] "data/intensities/basecalls/s_G1_L001_I1_001.fastq.1.gz"
[2] "data/intensities/basecalls/s_G1_L001_I1_002.fastq.1.gz"
[3] "data/intensities/basecalls/s_G1_L001_R1_001.fastq.1.gz"
[4] "data/intensities/basecalls/s_G1_L001_R1_002.fastq.1.gz"
[5] "data/intensities/basecalls/s_G1_L001_R2_001.fastq.1.gz"
[6] "data/intensities/basecalls/s_G1_L001_R2_002.fastq.1.gz"
```

We further select the fastq files that contain only R1 or R2 in their name.

```
> idx <- grep("_R(1|2)_", Name(f))
```

We can therefore download the selected file(s) to a local directory preserving the path information.

```
> outDir <- paste("Sample", Id(inSample), sep = "_")
> floc <- file.path(outDir, f$Path[idx])
> names(floc) <- basename(floc)
> floc
```

```
                                            s_G1_L001_R1_001.fastq.1.gz
"Sample_16018/data/intensities/basecalls/s_G1_L001_R1_001.fastq.1.gz"
                                            s_G1_L001_R1_002.fastq.1.gz
"Sample_16018/data/intensities/basecalls/s_G1_L001_R1_002.fastq.1.gz"
                                            s_G1_L001_R2_001.fastq.1.gz
"Sample_16018/data/intensities/basecalls/s_G1_L001_R2_001.fastq.1.gz"
                                            s_G1_L001_R2_002.fastq.1.gz
"Sample_16018/data/intensities/basecalls/s_G1_L001_R2_002.fastq.1.gz"
```

Next we download the selected file(s) to the `outDir` (this might take a while, depending on the number of files selected and the speed of the connection). Method `getFiles()` is used for downloading the data stream.

```
> getFiles(aAuth, id = Id(f)[idx], destDir = outDir, verbose = FALSE)
```

We can quickly check if the files were downloaded:

```
> file.exists(floc)
```

```
[1] TRUE TRUE TRUE TRUE
```

4 Data crunching

To compute the average Q-scores at each cycle we need to read the FASTQ file in R and extract the base qualities for each read. The Bioconductor `ShortRead` library offers tools for this. The functions used in this computation, `getQscoreCounts()` and `getQscoreStats()` are shown in Section 6.

```
> library(ShortRead)
> qtab <- lapply(floc, getQscoreCounts)
```

One can use the `parallel` package and the `mclapply` function to compute `qtab`. This is a perfect scenario for batch processing since we perform the computation on each FASTQ file separately.

```
> library(parallel)
> qtab <- mclapply(floc, getQscoreCounts, mc.cores = length(floc))
```

If the read pairs are spread across multiple FASTQ files, we have to aggregate the counts for both R1 and R2.

```
> idxR1 <- grep("_R1_", names(floc), fixed = TRUE)
> idxR2 <- grep("_R2_", names(floc), fixed = TRUE)
> if(length(idxR1) != length(idxR2))
+   stop("Missing files for R1 or R2")
> x <- getQscoreStats(cbind(Reduce("+", qtab[idxR1]), Reduce("+", qtab[idxR2])))
```

We can quickly inspect the computed statistics:

```
> head(x)
```

```
     5% median 95%     mean
[1,] 30     34  34 32.80311
[2,] 30     34  34 32.97502
[3,] 30     34  34 33.03821
[4,] 35     37  37 36.46075
[5,] 35     37  37 36.39961
[6,] 35     37  37 36.39945
```

4.1 Plotting Q-score statistics

We can now generate a simple plot summarizing the Q-score distribution. We save the graph into a local .png file such that we can later upload this file to BaseSpace.

```
> ylim <- range(x) + c(-2L, 2L)
> gfile <- file.path(tempdir(), "Qscore_per_cycle.png")
> png(file = gfile, width = 1000, height = 500, bg = "transparent")
> plot(x = seq_len(nrow(x)), type = "n", ylim = ylim,
+      xlab = "Cycle", ylab = "Q-score",
+      main = "Q-scores statistics")
> sx <- apply(x[, c("5%", "95%")], 2, function(x) smooth.spline(x)$y)
> sx[, "95%"] <- pmax(sx[, "95%"], x[, "median"])
> polygon(c(1L:nrow(x), nrow(x):1L), c(sx[, "95%"], rev(sx[, "5%"])),
+         col = "#CCEBC580", border = NA)
> matpoints(sx, type = "l", lwd = .5, lty = 2, col = "black")
> lines(x[, "mean"], lwd = 2, col = "red")
> lines(x[, "median"], lwd = 2, col = "black")
> legend("bottomleft", col = c("black", "red", "#CCEBC580"),
+        lwd = 10, inset = c(.05, .01), cex = 1.2, bty = "n",
+        legend = c("Median", "Mean", "5% - 95%"))
> dev.off()
```

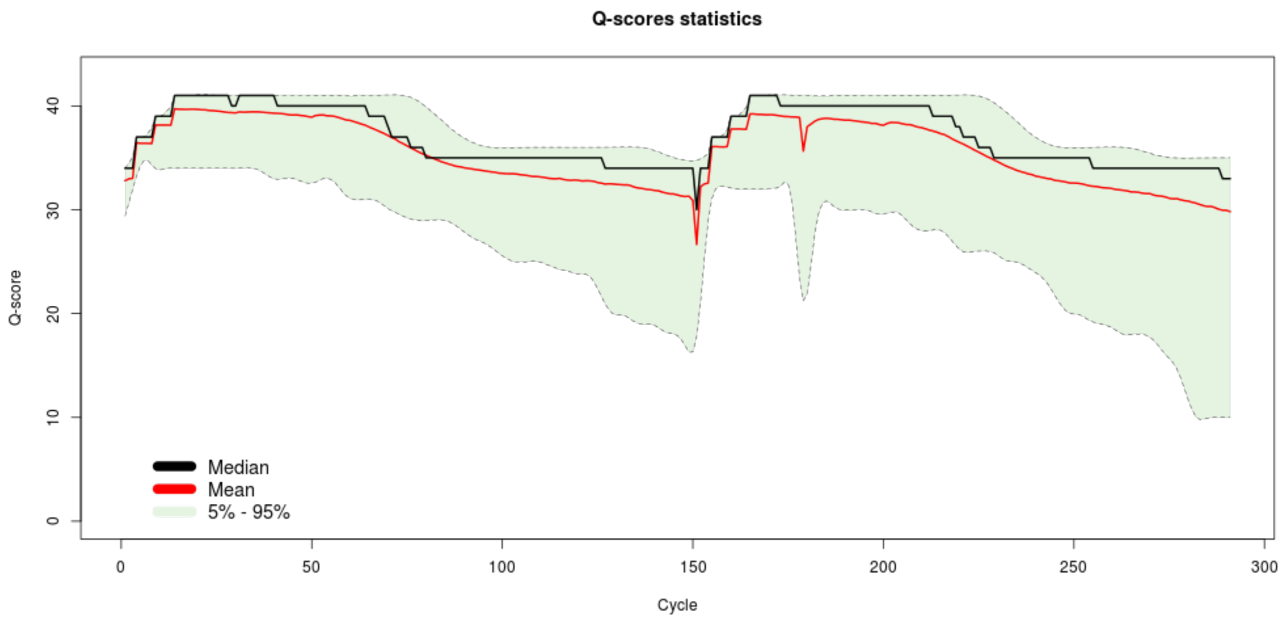

Figure 1: *Mean Q-score at each cycle. The shaded region gives the* 5% *and* 95% *bands.*

5 Storing the results

In this section we show how to store the analysis results back in BaseSpace. In this example we'll just keep the PNG figure, but we could store any type of file/data if necessary.

To do this we first need to create a new `AppResults` instance (possibly under the current project if we have permission to add results to it). Here we assume that we don't have permission to modify the project from where we read the samples. So we'll have to create a new project, associate the samples used with it and upload the analysis results.

5.1 Creating a new project

To create a new project we use the `createProject()` method. In this process it is a good practice to check whether a project with the same name already exists.

```
> pname <- "Test Apps"
> ## shows the name of all projects the user can see
> Name(myProj)
```

```
[1] "BaseSpaceDemo"       "ResequencingPhixRun" "My Project 1"        "My Project X"
[5] "Test Apps"
```

```
> idx <- which(Name(myProj) %in% pname)
> projId <- if(length(idx) == 0L) {
+   Id(createProject(aAuth, name = pname))
+ } else {
+   Id(myProj)[idx]
+ }
```

We can take a closer look at the (newly created) project

```
> newProj <- Projects(aAuth, id = projId, simplify = TRUE)
> newProj
```

```
Projects object:
{
        "HrefSamples" : "v1pre3/projects/206206/samples",
        "HrefAppResults" : "v1pre3/projects/206206/appresults",
        "Id" : "206206",
        "Name" : "Test Apps",
        "Href" : "v1pre3/projects/206206",
        "DateCreated" : "2012-11-22T15:53:49.0000000",
        "UserOwnedBy" : {
                "Id" : "660666",
                "Href" : "v1pre3/users/660666",
                "Name" : "Adrian Alexa",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/2ca61f7edbd417b45780c8f849622559
        },
        "HrefBaseSpaceUI" : "https://basespace.illumina.com/project/206206/Test Apps"
}
```

If this project was created before, most likely there are `AppResults` associated with it. We can browse the existing results within this project using the `listAppResults()` method.

```
> listAppResults(newProj)
```

```
AppResultsSummary object:
Collection with 1 appResultItem objects (out of a total of 1 objects).
{
        "Id" : "691691",
        "Name" : "Q-score dsitribution",
        "Href" : "v1pre3/appresults/691691",
        "DateCreated" : "2013-03-07T10:14:47.0000000",
        "UserOwnedBy" : {
                "Id" : "660666",
                "Href" : "v1pre3/users/660666",
                "Name" : "Adrian Alexa",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/2ca61f7edbd417b45780c8f849622559
        },
        "Status" : "Complete"
}
```

5.2 Creating a new AppResult

We next need to create a new `AppResults` instance which will hold our analysis data. To do this we create a JSON object with a well defined set of fields (see *Writing back to BaseSpace* for more details).

```
> ar <- list(Name = "Q-score dsitribution",
+            Description = "Simple stats on the Q-score at each cycle.",
+            "References" = list(Rel = "using",
+              HrefContent = Href(inSample)))
> cat(toJSON(ar, pretty = TRUE))
```

```
{
        "Name" : "Q-score dsitribution",
        "Description" : "Simple stats on the Q-score at each cycle.",
        "References" : {
                "Rel" : "using",
                "HrefContent" : "v1pre3/samples/16018"
        }
}
```

We are now ready to create the AppResults instance.

```
> newResult <- createAppResults(newProj, value = toJSON(ar))
```

```
AppResults:
 {
 "Name": "Q-score dsitribution",
"Description": "Simple stats on the Q-score at each cycle.",
"References": {
 "Rel": "using",
"HrefContent": "v1pre3/samples/16018"
}
}
successfully created. Assigned Id: 710710
```

If the user doesn't have write access to the current project, then the newResult object will be NULL and an error message from the REST server will be shown to the user. In this case we can launch the OAuth2 process with the required scope and prompt the user (this requires the user interacting with a web browser).

```
> if(is.null(newResult)) {
+   initializeAuth(aAuth, scope = paste("write project", projId))
+   requestAccessToken(aAuth)
+ }
```

Assuming that we have the proper permissions, we can inspect the newly created AppResults instance associated with our project.

```
> listAppResults(newProj)
```

```
AppResultsSummary object:
Collection with 2 appResultItem objects (out of a total of 2 objects).
{
        "Id" : "691691",
        "Name" : "Q-score dsitribution",
        "Href" : "v1pre3/appresults/691691",
        "DateCreated" : "2013-03-07T10:14:47.0000000",
        "UserOwnedBy" : {
                "Id" : "660666",
                "Href" : "v1pre3/users/660666",
                "Name" : "Adrian Alexa",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/2ca61f7edbd417b45780c8f849622559
        },
        "Status" : "Complete"
}
{
        "Id" : "710710",
        "Name" : "Q-score dsitribution",
        "Href" : "v1pre3/appresults/710710",
        "DateCreated" : "2013-03-08T12:09:13.0000000",
        "UserOwnedBy" : {
                "Id" : "660666",
                "Href" : "v1pre3/users/660666",
                "Name" : "Adrian Alexa",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/2ca61f7edbd417b45780c8f849622559
        },
        "Status" : "Running"
}
```

Please note that at this stage the status of the `AppResults` instance is set to *Running*.

5.3 Uploading data/files

The function used for data/file uploads is `putFiles()`. At the moment this function implements only the POST method. Multiple file uploads will be soon supported using the same interface.

To upload the PNG file we need to specify the `AppResults` ID and the file location on the disk.

```
> newFile <- putFiles(aAuth, resultId = Id(newResult), fIn = gfile)

File: 'Qscore_per_cycle.png' successfully uploaded! Assigned Id: 64259636
```

If the above command succeeds, the file is uploaded to BaseSpace and a new `Files` instance is created for this file.

```
> newFile

Files object:
{
        "UploadStatus" : "complete",
        "HrefContent" : "v1pre3/files/64259636/content",
        "Size" : 39644,
        "Path" : "Qscore_per_cycle.png",
        "ContentType" : "application/octet-stream",
        "Id" : "64259636",
        "Name" : "Qscore_per_cycle.png",
        "Href" : "v1pre3/files/64259636",
        "DateCreated" : "2013-03-08T12:09:14.3298323Z"
}
```

We can check the `UploadStatus` given by the `newFile` and if this is set to *Complete*, we can then finalize the session.

```
> if(newFile$UploadStatus != "complete")
+   stop("Problem upload the result ...")
> ## delete the PNG file from the local storage
> unlink(gfile)
```

To finalize the session we use the `updateAppSessions()` method to set the status of the `AppSessions` to *Complete*.

```
> ## Get the session Id
> sessionId <- Id(AppSessions(newResult))
> ## Complete the session
> invisible(updateAppSessions(aAuth, id = sessionId, status = "complete"))
```

```
App statuts successfully updated. New status: Complete
```

If updating the `AppSessions` is successful the user will receive a confirmation e-mail.

We can further inspect the `AppResults` to check if the status is properly set.

```
> myResults <- listAppResults(newProj)
> myResults
```

```
AppResultsSummary object:
Collection with 2 appResultItem objects (out of a total of 2 objects).
{
        "Id" : "691691",
        "Name" : "Q-score dsitribution",
        "Href" : "v1pre3/appresults/691691",
        "DateCreated" : "2013-03-07T10:14:47.0000000",
        "UserOwnedBy" : {
                "Id" : "660666",
                "Href" : "v1pre3/users/660666",
                "Name" : "Adrian Alexa",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/2ca61f7edbd417b45780c8f849622559
        },
        "Status" : "Complete"
}
{
        "Id" : "710710",
        "Name" : "Q-score dsitribution",
        "Href" : "v1pre3/appresults/710710",
        "DateCreated" : "2013-03-08T12:09:13.0000000",
        "UserOwnedBy" : {
                "Id" : "660666",
                "Href" : "v1pre3/users/660666",
                "Name" : "Adrian Alexa",
                "GravatarUrl" : "https://secure.gravatar.com/avatar/2ca61f7edbd417b45780c8f849622559
        },
        "Status" : "Complete"
}
```

```
> Status(myResults)
```

```
[1] "Complete" "Complete"
```

6 Functions used

Bellow are the functions used for computing the Q-score statistics in Section 4. We use the `readFastq()` function to read the FASTQ files and `quality()` to extract the base qualities.

```
> getQscoreCounts <- function(fIn, maxS = 42L) {
+   ## read the fastq file and keep the qualities
+   r <- quality(quality(readFastq(fIn, withIds = FALSE)))
+   r_row <- width(r)[[1L]]
+   r_col <- length(r)
+
+   ## transorm the qualities to integers from 2 to maxS
+   x <- as.integer(unlist(r, use.names = FALSE)) - 33L
+   dim(x) <- c(r_row, r_col)
+
+   ## tabulate each row in the matrix
+   qtab <- matrix(0L, nrow = maxS, ncol = r_row,
+                  dimnames = list(paste0("Q", seq_len(maxS)), NULL))
+   for(i in seq_len(r_row))
+     qtab[, i] <- tabulate(x[i, ], nbins = maxS)
+
+   return(qtab)
+ }

> getQscoreStats <- function(x) {
+   nr <- nrow(x)
+   nc <- ncol(x)
+   qstat <- matrix(0L, nrow = nc, ncol = 4L,
+                   dimnames = list(NULL, c("5%", "median", "95%", "mean")))
+   rleval <- seq_len(nr)
+   for(i in seq_len(nc)) {
+     r <- Rle(values = rleval, lengths = x[, i])
+     qstat[i, ] <- c(quantile(r, probs = c(0.05, .5, 0.95)), mean(r))
+   }
+
+   return(qstat)
+ }
```

7 Session Information

The version number of R and packages loaded for generating the vignette were:

- R version 2.15.2 (2012-10-26), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB.UTF-8`, `LC_COLLATE=en_US.UTF-8`, `LC_MONETARY=en_GB.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=C`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_GB.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: BaseSpaceR 0.98.2, BiocGenerics 0.4.0, Biostrings 2.26.3, bitops 1.0-5, GenomicRanges 1.10.6, IRanges 1.16.5, lattice 0.20-13, latticeExtra 0.6-24, RColorBrewer 1.0-5, RCurl 1.95-3, RJSONIO 1.0-1, Rsamtools 1.10.2, ShortRead 1.16.3
- Loaded via a namespace (and not attached): Biobase 2.18.0, grid 2.15.2, hwriter 1.3, parallel 2.15.2, stats4 2.15.2, tools 2.15.2, zlibbioc 1.4.0