Extending the umbrella of compliance: LSAF with R Distributed Computing:

Paper 32

Ben Bocchicchio, Sandeep Juneja, Rick Wachowiak SAS Institute

R studio is being utilized today to perform more and more highly specialized analysis in the Health and Life Industry. The R user's ability to 'pull' the latest package from a web-based R repository makes it easy to develop with the latest R code packages. To provide repeatability for R coding, once the R code is production ready, the R code and data are stored in Life Science Analytical Framework under version and audit control. Users can extend LSAF's compliant environment to include R code processing on a remote system, all that is needed is the LSAF API and a deployed web service.

Setup phase:

As part of the initial design, a web service is setup on the R server, this allow SAS code written in LSAF to POST metadata information to the R Server through PROC HTTP.

- Develop SAS code with definition for all inputs / outputs and list of R programs that need to be executed.
- Setup R Executor LSAF Userid
- Execute SAS Jobs from LSAF

```
data _null_;
 file input recfm=f lrecl=1;
 put '{';
    put '"url": "https://XXXXXXXX.sas.com",';
    put '"log": "/SAS/Files/Utilities/R_webservice/log",';
    put '"items": [';
                    put '{';
                    put '"input" : ["/SAS/Files/Utilities/R_webservice/xlsx_files/create_pfs_Manual.xlsx"],';
                    put '"program": "/SAS/Files/Utilities/R_webservice/programs/firstR.r",';
                    put '"output":["/SAS/Files/Utilities/R_webservice/output"]';
                    put '}';
             put ']';
 put '}';
run;
proc http
  method="POST"
  url="&r_server/&r_execute_url"
  in=input
  headerout=headers
  out=resp
  HEADEROUT_OVERWRITE;
  headers 'Content-Type' = 'application/json';
run;
```

In addition, to support the automation of the process, the LSAF API is installed on the R server.

R Code development phase:

To support the execution of the R code, the R developer designs their code using relative paths. This allows the R code to be executed in a non-interactive session. Through the use of relative paths, the process knows where to temporarily place the R code, and all required input for the code to execute. In addition, the results area is known, this allows the API to write back the results to LSAF

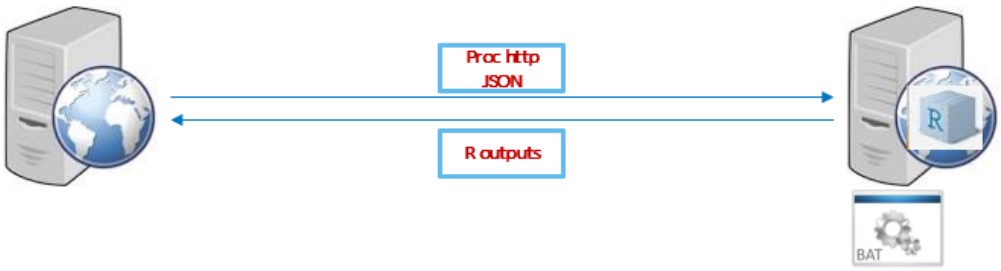
For example:

```
require(xlsx)
df = read.xlsx("../xlsx_files/create_pfs_Manual.xlsx",  1, header = TRUE)

sink(file="../output/results.txt", append=FALSE, split=FALSE)
print(df)
lapply(df, write, "../output/test.txt", append=TRUE, ncolumns=1000)

# Writing mtcars data
write.table(df, file = "../output/df.txt", sep = "\t", row.names = TRUE, col.names = NA)
```

Execution Phase:



A JOB is executed in LSAF, this establishes a connection to the R service using PROC HTTP. As part of this initial handshake, metadata information is provided to the R server. The connection then calls a BAT file that call the LSAF API and uses the metadata information to pull the exact files from LSAF and make a local temporary copy. The next step in BAT file is to execute the R code that was copied over. Since the code was written with relative paths, the R code know the location of the data and where to store the output. Upon completion of the R code, the LSAF API is once again called on to store results of the R code, including the LOG file, back into LSAF. The BAT/script file then cleans up any local files that were generated on the remote server

Compliance:

The LOG that is posted back to LASF will contain the results of the entire process of the run: the version of the data in LSAF that was used, the version of the R code used along with the versions of R packages used to generate the output. With this all this information, the output could be re-generated later supporting the needs of compliance.