Paper OS02

Leveraging APIs in a Multilingual World: Run SAS Jobs from your R Program

Samiul Haque, SAS Institute, Cary, USA
Jim Box, SAS Institute, Cary, USA

ABSTRACT

A common practice for R programmers in the pharmaceutical industry is to ingest SAS datasets through third-party packages and generate desired tables, listings, and figures using R packages of their choice. However, there are situations where they want to mix and match R and SAS capabilities. For example, one may want to trigger an SDTM automation process written with SAS Macros from R Shiny applications. Another example is producing ggplot2 figures based on outputs from a SAS macro. SAS Viya provides public API endpoints to execute SAS processes, which can be leveraged to integrate R and SAS seamlessly. SAS also has an R package that allows R programmers to leverage SAS in-memory massively parallel processing. In this session, we will discuss how SAS jobs and publicly available SAS REST APIs can be leveraged to operationalize a seamless R and SAS workflow.

INTRODUCTION

There is an increasing demand for integrating SAS workflows with external tools. SAS Viya provides an easy integration mechanism through SAS Jobs. In this work, we leveraged SAS Job execution for executing SAS programs from R. However, this integration can be achieved using any external tools or programming languages.

WHAT IS SAS JOB

A SAS Viya job consists of a program, job definition, its parameters, and prompt. SAS Job objects can be created from SAS Studio, Job Execution Application, or programmatically through REST APIs. Each job definition comes with an execution URL. The execution URL can be used to execute the Job using a browser or programmatically using REST APIs.

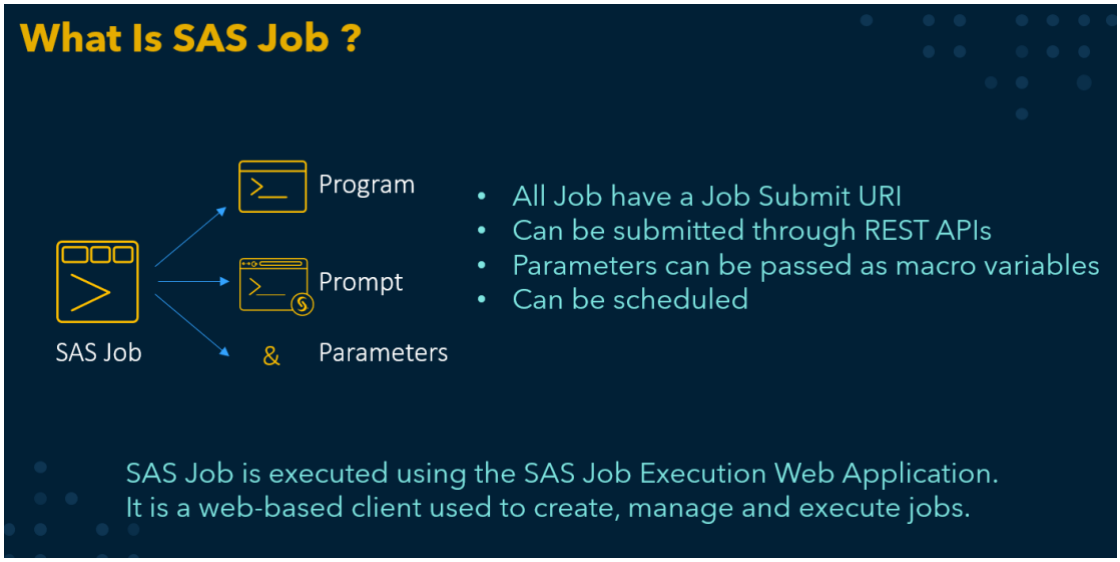

Figure 1 What is a SAS Job?

Jobs can have custom parameters, and parameters can have default values as well. In addition, there are options to choose the context or run time where the job will be executed. In this work, we created a job definition that calculates the mean MSRP values of cars by group. The data is available in SASHELP.CARS table.

The by variable for calculating the mean is decided during job submission by the ***byvar*** parameter declared in the job definition.

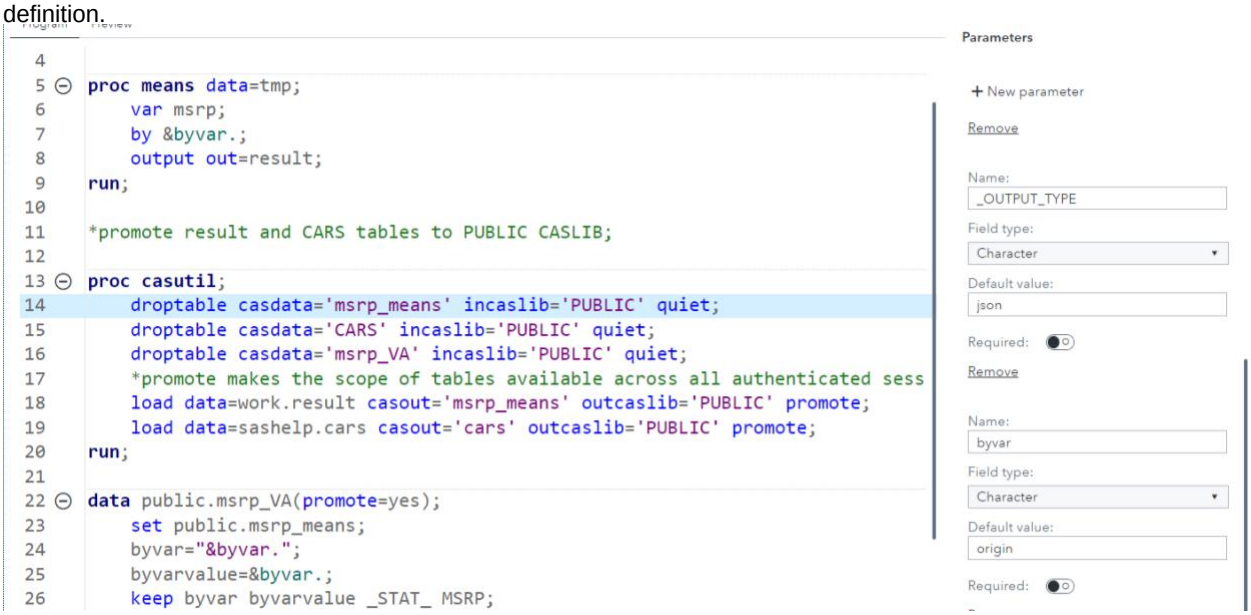

Figure 2 A Job definition in SAS Studio

In figure 2, we can see the default of byvar parameter is origin. If no parameter is supplied during the job submission byvar will be assigned as 'origin'.

JOB SUBMIT URL

A job submit URL is created when the job definition is saved; the URL can be retrieved from job properties. In the figure below, we have blurred the full URL for security purposes.

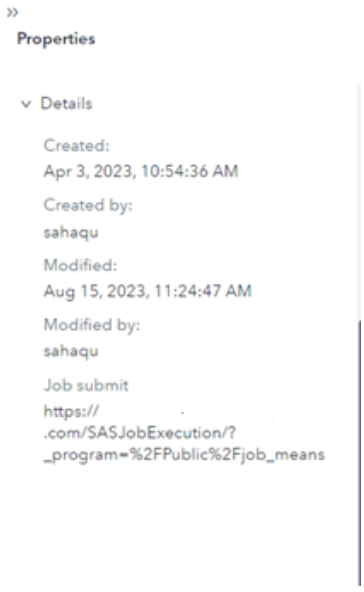

Figure 3 Job Properties contains the URL for job submission

REST API

Application Programming Interface (API) is a mechanism for communication between different software components and programming languages. Representation State Transfer (REST) is an architectural guiding principle for developing APIs. REST is a defined set of instructions for submitting and receiving responses via HTTP actions. When a client receives information from a server, it is done using a GET method. When a client submits a request to

the server, it is done using a POST method. SAS Viya provides public API endpoints to all the modules. Complete documentation with examples can be found on https://developer.sas.com/

EXECUTING SAS JOB FROM R

Once a job definition has been created, the job can be submitted from any client. In this work, we used R Studio as our client application. We explain the process step by step.

STEP 1 USERNAME AND PASSWORD FOR AUTHENTICATION

```
library(httr)
library(jsonlite)

uname<-rstudioapi::askForPassword("SAS username")
pass<-rstudioapi::askForPassword("SAS password")
```

STEP 2 POST AUTHENTICATION REQUEST

At this step, we construct headers and body of a POST request and submit the request to a SAS Viya instance.

```
#REST API call with credential for getting access_token

headers = c(
  'Authorization' = 'Basic c2FzLmNsaTo=',
  'Content-Type' = 'application/x-www-form-urlencoded'
)

body = list(
  'grant_type' = 'password',
  'username' = paste0(uname),
  'password' = paste0(pass)
)

response <- POST(url = "https://sasviya.example.com/SASLogon/oauth/token", body = body, add_headers(headers), encode = 'form')
```

response is a JSON object that contains an access token for the SAS instance. We need to use that access token for all future REST API calls. In the next step, we convert the JSON object into an R data frame and retrieve the access token to construct the following API calls.

```
response_df=fromJSON((content(response,'text')))

print(response_df)

#Use access token for subsequent REST API calls

headers = c(
  'Authorization' =  paste0("Bearer  ",response_df$access_token," ") )
```

STEP 3 POST JOB SUBMIT URL

At this step, we leverage the URL obtained from JOB properties and make a POST call. The job execution result is returned in the res variable as JSON.

```
#Run a Pre-Defined Job in SAS Viya using REST API this can also be submitted as Asynchrounous call
res <- POST(url = "https://sasviya.example.com/SASJobExecution/?_program=%2FPublic%2Fjob_means", add_headers(headers))
```

STEP4 RETRIVE AND PRINT RESTULS

```
result_df=fromJSON((content(res,'text')))
print(result_df)
```

```
> print(result_df)
   Origin _TYPE_ _FREQ_ _STAT_      MSRP
1    Asia      0    158      N    158.00
2    Asia      0    158    MIN  10280.00
3    Asia      0    158    MAX  89765.00
4    Asia      0    158   MEAN  24741.32
5    Asia      0    158    STD  11321.07
6  Europe      0    123      N    123.00
7  Europe      0    123    MIN  16999.00
8  Europe      0    123    MAX 192465.00
9  Europe      0    123   MEAN  48349.80
10 Europe      0    123    STD  25318.60
11    USA      0    147      N    147.00
12    USA      0    147    MIN  10995.00
13    USA      0    147    MAX  81795.00
14    USA      0    147   MEAN  28377.44
15    USA      0    147    STD  11711.98
```

As no ***byvar*** parameter was passed the default by variable origin was used for calculating means. In our next step, we pass a by variable as parameter.

STEP5 POST JOB SUBMIT URL WITH BY VARIABLE

```
#Now run the job with a parameter / macro we will change the by variable
res <- POST(url = "https://mipistvk.eastus.cloudapp.azure.com/SASJobExecution/?_program=%2FPublic%2Fjob_means&byvar=make", add_headers(headers))
result_df=fromJSON((content(res,'text')))
print(result_df)
```

Note how &byvar=make is appended to the URL and the following result is generated in the R Studio console.

```
> result_df=fromJSON((content(res,'text')))
> print(result_df)
            Make _TYPE_ _FREQ_ _STAT_        MSRP
1          Acura      0      7      N      7.0000
2          Acura      0      7    MIN  23820.0000
3          Acura      0      7    MAX  89765.0000
4          Acura      0      7   MEAN  42938.5714
5          Acura      0      7    STD  22189.0077
6           Audi      0     19      N     19.0000
7           Audi      0     19    MIN  25940.0000
8           Audi      0     19    MAX  84600.0000
9           Audi      0     19   MEAN  43307.8947
10          Audi      0     19    STD  13533.6632
11           BMW      0     20      N     20.0000
12           BMW      0     20    MIN  28495.0000
13           BMW      0     20    MAX  73195.0000
14           BMW      0     20   MEAN  43285.2500
15           BMW      0     20    STD  12459.7565
16         Buick      0      9      N      9.0000
17         Buick      0      9    MIN  22180.0000
18         Buick      0      9    MAX  40720.0000
19         Buick      0      9   MEAN  30537.7778
20         Buick      0      9    STD   6371.6560
21      Cadillac      0      8      N      8.0000
22      Cadillac      0      8    MIN  30835.0000
```

CONCLUSION

We presented a flexible and easy-to-implement way for integrating SAS jobs with R or any other programming language. This work can be extended to incorporate SAS Jobs in a Shiny App, allowing R Shiny users to leverage SAS PROCS and Macros.

REFERENCES

1. SAS® Job Execution Web Application: What It Is & When to Use It.
2. Jobs: Stored processes in Viya