PhUSE US Connect 2019

Paper CT03

# A Method for Automating Jobs in SAS Life Science Analytics Framework (LSAF)

Nicholas Masel, Janssen R&D, Raleigh, NC, US

## ABSTRACT

SAS® Life Science Analytics Framework (LSAF) requires jobs to be created to produce output, including datasets and TLFs, in the production environment (Repository). The job basically defines what inputs are required to make an output. As a programmer first begins in LSAF, job creation can seem confusing which often leads to jobs being defined inefficiently due to lack of understanding of what is happening behind the curtain, or due to the complexity of defining all input files accurately to produce a complex output. Automating the job process allows for accurate job creation, increased performance in job run time, and a reduction in the time needed for new LSAF programmers to be able to execute production code. Using code to return a file list along with a PROC SCAPROC, SCAPROC parser, and an XML job template allow you to quickly and accurately define a job or all your jobs with ease.

## INTRODUCTION

The goal of this paper is to provide a method of job automation that can easily be updated to suit your specific needs. The scope will include an outline of the steps to automate the job creation process with details on determining inputs and outputs as well as programmatically populating jobs. Most of the steps have multiple, well documented options, which will not be covered in detail in this paper. This paper will assume you know what a job is in LSAF.

The method outlined in this paper is specific to a process that creates one job per program and can be modified to meet your needs.

## PROCESS OVERVIEW

The process is very simple and can be implemented quickly in your organization to create one job or a job for each file in a directory.

1. Store the PROC SCAPROC output.
2. Parse the PROC SCAPROC output for inputs and outputs.
3. Plug the inputs and outputs into the job XML code and write out the job file.
4. Apply this process to one file or loop through multiple files.

### STORE THE PROC SCAPROC OUTPUT

This is outlined clearly in "QA and Compliance Insights Using the SCAPROC Procedure" by Ben Bocchicchio and Sandeep Juneja. In this paper you will find a simple macro you can use to run the PROC SCAPROC and capture the results in a file.

### PARSE THE PROC SCAPROC OUTPUT FOR INPUTS AND OUTPUTS

Ben and Sandeep's paper also touches on this, but an updated version can be found from SAS support. Sample 58047 provides code that can be used to parse the PROC SCAPROC output. For the purpose of LSAF job inputs and outputs, the code from SAS actually provided too much information. A minor adjustment to only keep what is needed was necessary.

**PLUG THE INPUTS AND OUTPUTS INTO THE JOB XML CODE AND WRITE OUT THE JOB FILE**

If you use infile to read in a job you will see the XML. This can be used to see the expected syntax to define a job correctly. Knowing the XML syntax along with the inputs and outputs, you can now programmatically build the code and plug this in to the job.

First, we create a job template program, templateJob.sas, to use as our starting point. Here we can define a standard job shell that accounts for all of the items you need for your specific process and environment. In my organization, this requires an environment setup program in all jobs, writing the log, lst, and manifest to subfolders, and defining a few parameters. You can also add placeholders for program name, inputs, outputs, and parameter default values. These are later replaced with actual values.

Sample code to read in the template job:

```
data tempjob;
   infile "&_SASWS_/testFolder/jobs/templateJob.sas" length=len;
   input string $varying32767. len;
   string=strip(string);
   put string;
run;
```

Contents of job shown by line number for reference with placeholders in bold:

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <job releaseVersion="1.0" description="" executionMode="SEQUENTIAL" LSAFVersion="4.0"
    log="./log" lst="./lst" mnf="./mnf">
3.  <tasks>
4.  <task path="../programs/envsetup.sas" />
5.  <task path="../programs/PROGRAMPLACEHOLDER.sas" />
6.  </tasks>
7.  <taskSpecs>
8.  <inputSpec path="../programs/envsetup.sas" type="FILE" version="*"
    includeSubFolders="false" />
9.  <inputSpec path="../programs/PROGRAMPLACEHOLDER.sas" type="FILE" version="*"
    includeSubFolders
10. </taskSpecs>
11. <otherSpecs>
12. <XMLINPUTPLACEHOLDER>
13. </otherSpecs>
14. <outputPaths>
15. <XMLOUTPUTPLACEHOLDER>
16. </outputPaths>
17. <outputSpec enableVersioningForNewFiles="true">
18. <versionType type="MAJOR" />
19. </outputSpec>
20. <parameters>
21. <parameter name="source_ds" type="CHARACTER" defaultValue="SOURCEDSPLACEHOLDER"
    label="Source data" />
22. <parameter name="source_meta" type="CHARACTER" defaultValue="SOURCEMETAPLACEHOLDER"
    label="Source metadata" />
23. </parameters>
24. </job>
```

On line two, you can see we set the log, lst, and manifest to write to subfolders.

Lines four and five add the environment setup program and a placeholder for the program that will link to the job. As a reminder, this process is to create a one program to one job relationship due to my organization's process. You can change line five to an <XMLPROGRAMPLACEHOLDER> and plug in multiple programs if you like following the same process used for inputs and outputs. The programs used in lines four and five will also need to be placed as inputs to the job as seen on lines eight and nine.

Line 12 is our placeholder for inputs and line 15 is our placeholder for outputs.

Lines 21 and 22 have our parameters that we will need for all jobs based off organization specific process. You can plug in whatever you need by modifying this example.

Now to replace the placeholders with actual values. We do this in two different ways using macro variables and merging. For the macro variables, you can plug these in while reading the job template into SAS.

For example:

```
data tempjob;
   infile "&_SASWS_/testFolder/jobs/templateJob.sas" length=len;
   input string $varying32767. len;
   string = tranwrd(string, "PROGRAMPLACEHOLDER","&&file&jobnumber");
   string = tranwrd(string, "SOURCEDSPLACEHOLDER","&source_ds");
   string = tranwrd(string, "SOURCEMETAPLACEHOLDER","&source_meta");
   ord=_n_;
   string=strip(string);
   put string;
run;
```

For inputs and outputs, we build the XML syntax in a separate working dataset and merge this onto the job template. Here is an example of building the input and output working dataset. In the example, the dataset scaResults contains the output from your SCA parse. So this is a dataset that contains type of file and the path to the file.

```
data feedToJob;
set scaResults;

   length string $32767;
   if lowcase(type) = "input" then do;
      xml = '<inputSpec path="'|| strip(tranwrd(file_path,"&_SASWS_",""))
            ||'" type="FILE" version="*" includeSubFolders="false" />';
      string = "<XMLINPUTPLACEHOLDER>";
      *(optional) remove program from input;
      if strip(lowcase(file_path)) = strip(lowcase("&programpath")) then delete;
   end;

   if lowcase(type) = "output" then do;
      xml = '<outputPath path="'||tranwrd(substr(file_path , 1, find(file_path,
'/', -length(file_path))-1) ,"&_SASWS_" ,"")||
            '" includeSubFolders="false" />';
      string = "<XMLOUTPUTPLACEHOLDER>";
   end;

run;
```

You can now merge this dataset with your job by string and populate the job with your inputs and outputs.

```
   proc sort data=tempjob;
      by string;
   run;

   proc sort data= feedToJob;
```

```
   by string;
run;

data finaljob;
merge tempjob
      feedToJob(in=b);

    by string;

    if b then string = strip(xml);

run;

proc sort data=finaljob;
   by ord;
run;
```

Finally, write the file out to your job folder.

```
filename final "&_SASWS_/testFolder/jobs/&&file&jobnumber...job";

data _null_;
   file final;
   set finaljob(keep=string);
   put string;
run;
```

**APPLY THIS PROCESS TO ONE FILE OR LOOP THROUGH MULTIPLE FILES**

Wrapping this process in a loop allows you to create multiple jobs. Adding a macro to return all files in a directory expands this further and provides flexibility. Piecing this all together gives you a solution where you provide a directory path, this is searched and returns a list of SAS programs, then the process outlined is repeated once for each SAS program in the directory.

This code can be expanded further to provide options to combine multiple SAS programs into one job as well. This was not necessary for my needs but could come in handy depending on your organization's process.

## CONCLUSION

This method allows programmers to continue to focus on analysis programming rather than the intricacies of creating efficient jobs. It also expands the run and populate of a single job to a programmatic solution that can be run to create all your jobs at once. On the server side, efficient jobs help to reduce the load and in turn makes all jobs run faster. The combination of reduced programmer effort and faster runtimes enhances the user experience.

## REFERENCES

SAS Institute 2014, *Sample 58047: SAS® Life Science Analytics Framework - Parse output from PROC SCAPROC to create a data set with inputs and outputs*, viewed 22 AUG 2017, <http://support.sas.com/kb/58/047.html>.

Bocchicchio, Ben and Juneja, Sandeep 2016, *QA and Compliance Insights Using the SCAPROC Procedure*, PharmaSUG 2016.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Nicholas Masel
Janssen R&D
920 Route 202
Raritan, NJ 08869
Work Phone: (919) 917-7690
Email: nmasel@its.jnj.com