Paper 153-31

The Dynamic SAS® Scheduler System (DSx3) Framework: Scheduling SAS Jobs Has Never Been Easier

Choon-Chern Lim, Mayo Clinic, Rochester, MN

ABSTRACT

Tired of manually scheduling all of your SAS jobs yourself? Tired of playing the "waiting game" with your system administrators so that they can schedule the jobs for you whenever they have free time? Worry no more… these problems will all be solved shortly.

This paper presents the Dynamic SAS Scheduler System (DSx3) framework which allows you to programmatically schedule your SAS jobs. This paper also explores the advantages and disadvantages of using this framework, and some workarounds to ensure this framework runs smoothly. The DSx3 framework is currently built for Windows environment, but this concept should work in different environments with slight modifications to the framework.

INTRODUCTION

As project deadlines get shorter and tighter, scheduling jobs to run off-work hours becomes more and more inevitable. Sometimes we face foreseeable barriers, such as the need to wait for system administrator to schedule jobs on our behalf. Sometimes it is just cumbersome to manually schedule/un-schedule each job. These barriers limit our maximum productivity.

The Dynamic SAS Scheduler System (DSx3) framework provides a simple yet powerful concept that allows you to programmatically schedule/un-schedule SAS jobs. The main goal of this paper is to provide you an alternative way to schedule your SAS jobs. This paper presents a detailed step-by-step approach to set up the framework where you can customize it to fit your needs.

STORING YOUR SAS JOBS

The best place to store the SAS jobs is in the SAS data set. The basic data set structure should contain (but not limited to) the following fields:-

Variable	Description
Date Scheduled	When the job is scheduled
LAN ID	User initial
Job Name	One word job initial
Job Description	A brief description of what the job does
Job Path	Where the SAS file is located

Although some fields such as "Date Scheduled", "LAN ID" and "Job Description" are not really used in this framework, they provide good documentations for future reference.

The below is an example of the data set:-

	DATESCHEDULED	LANID	JOBNAME	JOBDESC	JOBPATH
1	10/10/2005	USER_A	JOB_ONE	First test job	C:\sugi\scheduler\pgm\job_one.sas
2	11/24/2005	USER_B	JOB_TWO	Second test job	C:\sugi\scheduler\pgm\job_two.sas
3	12/03/2005	USER_C	JOB_THREE	Third test job	C:\sugi\scheduler\pgm\job_three.sas

Figure 1: Data Set Structure for Storing SAS Jobs

It is recommended to write a simple utility macro function that allows you to easily add/edit/delete the scheduled jobs.

DYNAMIC SAS SCHEDULER SYSTEM (DSx3) FRAMEWORK

The DSx3 framework constitutes one SAS file that acts as the main job controller and this file will be attached to the task scheduler. Once the task scheduler kick starts this SAS file, it will read and process each job from the SAS data set. Each processed job will have its own log file stored in the dated folder to allow you to locate the log files more efficiently.

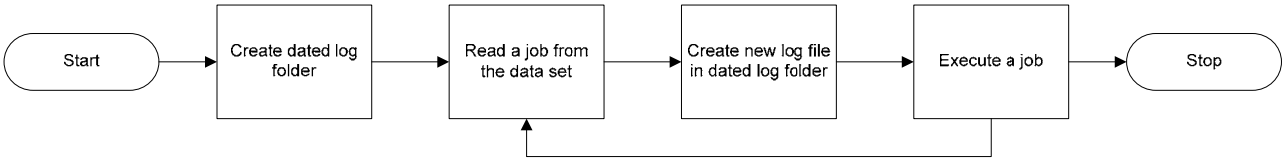

Figure 2: DSx3 Framework Overview

Below is the source code for the DSx3 framework. The macro function and macro variables are prefixed with “dsx3” in order to make them unique.

```
/*************************************************************
 *** Clear log and output before executing the script
 *************************************************************/
dm 'clear log; clear output';

options mrecall mprint symbolgen mlogic;

%macro dsx3_scheduler;
      /*************************************************************
       *** Initializing macro variables to be local is always a good
       *** practice.
       *************************************************************/
      %local dsx3TotalJobs dsx3DateLog dsx3LogFolder;

      /*************************************************************
       *** Data library for the scheduler data set.
       *************************************************************/
      libname dsx3Lib 'C:\sugi\scheduler\data';

      /*************************************************************
       *** Initialize the total scheduled jobs to be zero first.
       *************************************************************/
      %let dsx3TotalJobs = 0;

      /*************************************************************
       *** Date today in YYYY-MM-DD format. This is used for creating
       *** the dated log folder.
       *************************************************************/
      %let dsx3DateLog = %sysfunc(today(), yymmddd10.);

      /*************************************************************
       *** Set up a dated log folder path.
       *************************************************************/
      %let dsx3LogFolder = C:\sugi\scheduler\log\&dsx3DateLog.;

      /*************************************************************
       *** Create dated log folder to store the log files for the
       *** executed jobs.
       *************************************************************/
      x "cmd /C mkdir &dsx3LogFolder.";
```

```
/*****************************************************************
 *** Get number of scheduled jobs from the scheduler data set and
 *** store the value in macro variable "dsx3TotalJobs".
 ***
 *** The dictionary.tables are special read-only tables that
 *** contains many useful information about SAS data sets.
 ***
 *** The library name and data set name must be all in capital letters.
 *****************************************************************/
proc sql noprint;
      select nlobs into :dsx3TotalJobs
      from dictionary.tables
      where libname = "DSX3LIB" and memname = "SCHEDULER";
quit;

/*****************************************************************
 *** Execute each scheduled job.
 *****************************************************************/
%do dsx3LoopCount = 1 %to &dsx3TotalJobs.;

      /*************************************************************
       *** For each loop, get the job information and store them
       *** in the macro variables.
       ***
       *** It is HIGHLY IMPORTANT to put a stop statement at the
       *** last line of the data step to prevent an infinite loop.
       *************************************************************/
    data _null_;
          length currentRow 8.;
          currentRow = input("&dsx3LoopCount.", best8.);
          set dsx3Lib.scheduler point = currentRow;

          call symput("dsx3LanId", trim(left(lanid)));
          call symput("dsx3JobName", trim(left(jobName)));
          call symput("dsx3JobPath", trim(left(jobPath)));
          call symput("dsx3JobDesc", trim(left(jobDesc)));

          stop;
      run;

      /*************************************************************
       *** Clear work library before executing the job.
       *************************************************************/
      proc datasets lib=work kill nolist;
      quit;

      /*************************************************************
       *** Set up the file reference to the log file stored in the
       *** previously created dated log folder.
       *************************************************************/
      filename log&dsx3LoopCount. "&dsx3LogFolder.\&dsx3JobName..log";

      /*************************************************************
       *** Write SAS log into the specified log file.
       *************************************************************/
      proc printto log=log&dsx3LoopCount. print=log&dsx3LoopCount.;
      quit;

      /*************************************************************
       *** Turn off the symbolgen and mlogic options so that the
       *** log header is not going to get messed up.
       *************************************************************/
      options nosymbolgen nomlogic;
```

```
            /*************************************************************
             *** Create log header for the job.
             *************************************************************/
            %put ############################################################;
            %put ### Job Number : &dsx3LoopCount.;
            %put ### Datetime   : %sysfunc(datetime(), datetime16.);
            %put ### LAN ID     : &dsx3LanId.;
            %put ### Job Name   : &dsx3JobName.;
            %put ### Job Path   : &dsx3JobPath.;
            %put ### Job Desc   : %superq(dsx3JobDesc);
            %put ############################################################;
            %put ;

            /*************************************************************
             *** Turn on the symbolgen and mlogic options.
             *************************************************************/
            options symbolgen mlogic;

            /*************************************************************
             *** Execute the job.
             *************************************************************/
            %include "&dsx3JobPath.";

            /*************************************************************
             *** Although it is not neccessary to suspend the program
             *** execution for 5 seconds, it is recommended since SAS
             *** may get tied up on the previous job (for example, if the
             *** previous job interacts with Excel through DDE).
             *************************************************************/
            %let rc = %sysfunc(sleep(5));
        %end;

        /*****************************************************************
         *** Clear data library.
         *****************************************************************/
        libname dsx3Lib clear;

%mend dsx3_scheduler;

%dsx3_scheduler;
```

TESTS AND RESULTS

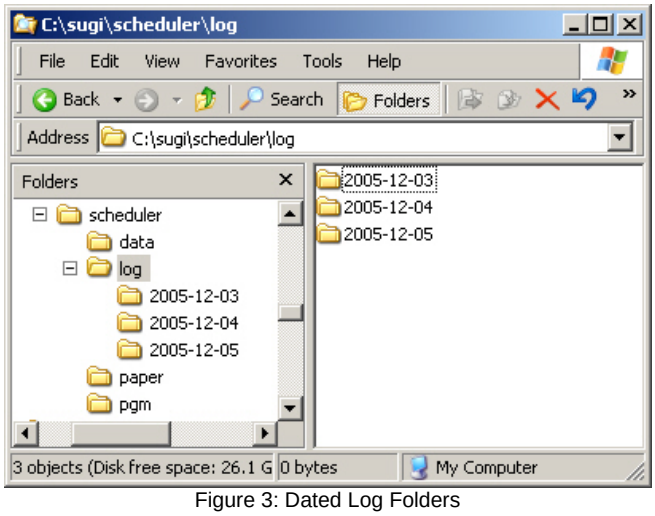

Figure 3: Dated Log Folders

When the SAS codes representing the DSx3 framework are executed, a dated log folder is created.

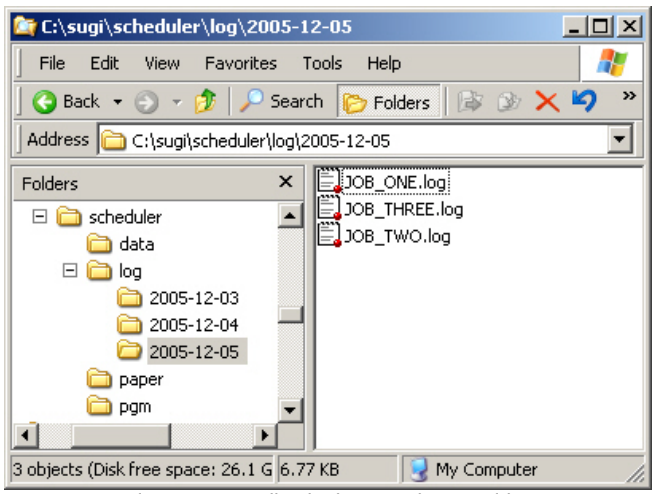

Figure 4: Log Files in the Dated Log Folder

Each dated log folder contains one log for each executed job. In this example, three jobs are executed on 12-05-2005.

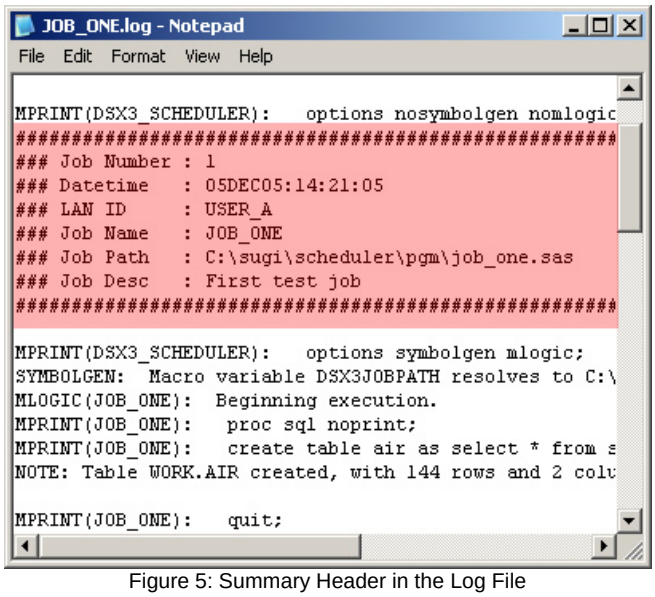

Figure 5: Summary Header in the Log File

Each log file contains a summary header (highlighted in red).

ADVANTAGES / DISADVANTAGES

ADVANTAGES

1. Schedule future SAS jobs without system administrator's intervention.
2. Provide you with more controls on the scheduled jobs (especially if they are running on a different server). It is easier to add/edit/delete the scheduled jobs since the job properties are stored in SAS data set.
3. Centralize all the SAS jobs in one location.
4. No need to manually schedule SAS jobs through the task scheduler, since they can now be programmatically scheduled through SAS.
5. Systematic way of creating and storing the log files for easier lookups in the future.

DISADVANTAGES

1. Scheduled jobs run sequentially. Although the framework can be tweaked so that the jobs run at different days, there is no efficient way to modify the time the job should run.
2. Potential collisions (and values overriding) on the macro variables and macro functions. Since each job are executed through %include, all macro functions and global macro variables are made available to the next available job. This will cause a major debugging headache if the scheduled jobs start behaving strangely.

HELPFUL TIPS

1. If you plan to schedule this task on a server, it is highly recommended to consult and get the system administrator’s consent before doing so.
2. Prefix unique naming convention to all macro variables and macro functions in each job. This will prevent potential collisions and values overriding.
3. Make each job a macro function and declare the macro variables in the macro function. This ensures that the macro variables would only live in the scope of the macro function.
4. With slight modification to the code, you can schedule jobs to run at the same time, but on different days.

CONCLUSION

The purpose of this paper is not only to get you started right away with your own scheduling system, but to introduce you to a simple yet powerful concept that allows you to programmatically schedule SAS jobs. The DSx3 framework also provides an efficient way of storing the log files for each executed job. By carefully taking some precautions listed above, you can easily overcome potential problems to avoid debugging headaches later.

ACKNOWLEDGEMENTS

Special thanks to Jon Kosanke for taking his precious time to review my paper.

CONTACT INFORMATION

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

Choon-Chern Lim
Mayo Clinic
200 First Street SW
Rochester, MN 55905
Email: limc@mayo.edu