Automating SAS Job Streams with the Power of VB Script

Lindsey Whiting, Joey Kaiser, Kohler Co.

ABSTRACT

When making important business decisions it is crucial to have the ability to manipulate and analyze large amounts of data. With the amount of data available to us in the world today, automating SAS jobs has become a necessity to provide efficient and critical business improvements. A great strategy to automate SAS jobs and give the ability for custom error checking is to leverage VB Scripting with your SAS code. This paper will discuss job stream automation and utilizing SAS to check files' statuses, notify users of program errors and automatically send out the results of your code.

INTRODUCTION

Effectively automating code can save countless hours of manual work and a lot of headaches from troubleshooting errors. Using VB Script to kick off your SAS jobs in conjunction with task scheduling software to decide the timing and frequency of your jobs is an effective way to automate all your SAS jobs. There are multiple task scheduling programs out there, but every computer should have a basic task scheduler software pre-installed on it. The task scheduler used in this paper is the basic pre-installed Windows scheduler software named Task Scheduler. Unfortunately, you can't directly schedule SAS code from the task scheduler. Therefore, utilizing VB Script using Notepad++ is a different method that enables SAS execution. Notepad++ is an easily downloadable tool to write VB script; however, other programs can be used in this instance. After kicking off SAS by scheduling the VB script in Task Scheduler you can then utilize your SAS code to create job streams that email errors, check input files, wait for input files, and automatically send emails.

VB SCRIPT FOR KICKING OFF SAS

VB Script is a scripting language that can be used to automate the running of virtually anything on your computer. We will show how you can run SAS jobs, create custom logs and control where your SAS log/output are saved. VB Script runs SAS by first opening the command prompt and then telling the command prompt to run SAS. In order for the command prompt to run SAS, there are some required and some optional arguments to pass to the command prompt.

The following arguments are required:

1. Where SAS is installed.
2. Where the SAS code you want to run is saved.

The following arguments are optional:

1. Location of where you want the output of the SAS code saved.
2. Location of where you want the log of the SAS code saved.

If you don't pass the optional arguments then the log and output will be saved in the same directory where the SAS code is saved. Figure 1 shows an example of running a SAS program straight from the command line that specifies where the output and log will be saved.

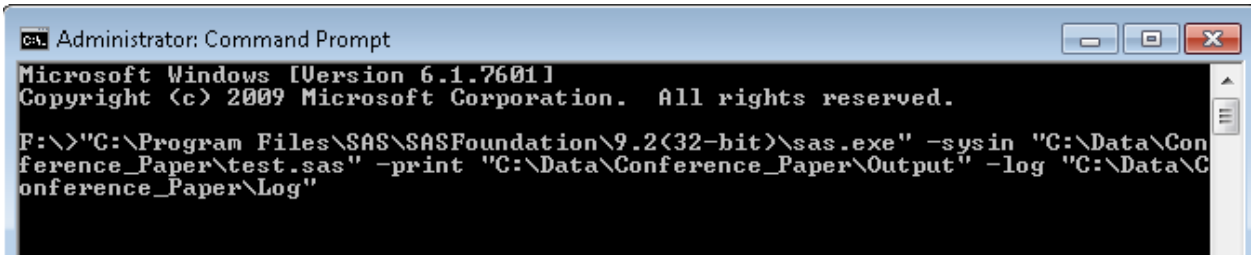

Figure 1

The location where SAS is installed always needs to be listed first. After that, the order doesn't matter as long as you have the appropriate option listed. The -sysin option needs to be listed directly before the line of code where the SAS program is saved, the -print option needs to be listed directly before the output location and the -log option needs to be listed directly before the log location. Again, the -log and -print options are not required. If you don't list them then they will be saved in the same location the SAS program is saved.

Now we can look at calling the command line from VB Script. Figure 2 shows how you pass code from VB Script to the command line.

```
1  Dim oShell
2  Set oShell = WScript.CreateObject ("WScript.Shell")
3  oShell.run "Insert text to run in command line"
4  Set oShell = Nothing
```

Figure 2

Next, we just need to pass the correct code to the command line. The double quotes can cause errors, so you need to make sure those get passed correctly. Everything that goes after the word run in line 3 from Figure 2 needs to be in character format which means you will need everything to be inside quotes or to be a variable that is a string. Where it gets tricky is that we also need some double quotes to be passed to the command line. In order to accomplish this, we need to use two double quotes in a row to signify that we are using the quotes character rather than starting/ending a string. To illustrate this point, Figure 3 shows the VB Script and Figure 4 shows the line on the command prompt that it equates to.

```
oShell.run """C:\Program Files\SAS\SASFoundation\9.2(32-bit)\sas.exe""
-sysin ""C:\Data\Conference_Paper\test2.sas"""
```

Figure 3

Note: Figure 3 is really one line in the VB Script. It is only copied as two lines into this document to be able to read it clearly. If you want to split one line of code into multiple lines in VB Script you need to use an ampersand and an underscore separated by a space at the end of the first line to connect it to the second line. This will be shown later in Figure 5.

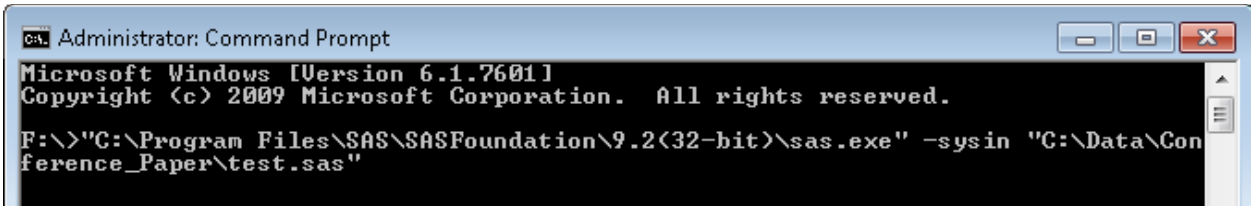

Figure 4

Everywhere in the VB Script where there are two double quotes in a row is converted to one double quotes in the command line. The VB Script starts and ends with three double quotes in a row because the entirety of the code needs to be surrounded in double quotes and the code that we need to pass to the command line happens to start and end with double quotes.

In order to keep your code cleaner and easier to read, you can use an ampersand and an underscore to write one line of VB Script onto multiple lines in your editor. Spacing and indenting does not affect VBS code at all, but entering to a new line will affect your code. An ampersand is used to concatenate two different strings and the underscore is used to concatenate two lines together. Figure 5 produces the full code passed to the command line that was shown in Figure 1.

```
Dim oShell
Set oShell = WScript.CreateObject ("WScript.Shell")
oShell.run """C:\Program Files\SAS\SASFoundation\9.2(32-bit)\sas.exe""" & _
          " -sysin ""C:\Data\Conference_Paper\test.sas""" & _
          " -print ""C:\Data\Conference_Paper\Output""" & _
          " -log ""C:\Data\Conference_Paper\Log"""
Set oShell = Nothing
```

Figure 5

CUSTOM LOG CREATION

For troubleshooting purposes, the log can be your best friend. Unlike SAS, VB Script does not create any sort of log when you run it so creating your own custom log can be very helpful. The following VBS code in Figure 6 can be used to create a log file:

```
Dim objFSO, objFSOText, objFolder, objFile, objTS
Dim logDirectory, logFile
logDirectory = "C:\Data\Conference_Paper\Testing\Log\"
logFile = logDirectory & "Log.txt"

Set objFSO = CreateObject("Scripting.FileSystemObject")

Set objTS = objFSO.OpenTextFile(logFile, 8, true)
objTS.WriteLine("Line 1")
objTS.WriteLine("")
objTS.WriteLine("Line 2")
objTS.close
```

Figure 6

The first argument of the OpenTextFile function is your log file you want to update. The second argument can be a 1 for reading, 2 for writing and 8 for appending. We want 8 in this instance because it will always put the new line we insert at the end of the file. The third argument is true if you want the program to create the txt file if it doesn't already exist.

This will create a txt file named Log shown in Figure 7:

```
Line 1

Line 2

```

Figure 7

One negative about creating a file named Log.txt is that it will always overwrite itself when the job is re-ran. To avoid this, it is a good practice to include the date and time of run in the name of the log file. This can be accomplished with the following code:

```
logFile = logDirectory & "Log" & month(Now()) & day(Now()) & hour(Now()) & minute(Now()) & ".txt"
```

After declaring the variables needed at the top of your code, you will now be able to put the following code throughout your VB Script to update your custom log:

```
8   Set objTS = objFSO.OpenTextFile(logFile, 8, true)
9   objTS.WriteLine("Insert text to add to log")
10  objTS.close
```

Figure 8

If the VB script has multiple parts throughout, inserting the custom log text can help the flow and help in trouble shooting your Job Stream.

SCHEDULING IN TASK SCHEDULER

After writing the VB script you can schedule it to run in the Task Scheduler. Within your Task Scheduler Library, you can create a folder called Automated Jobs (or any name you choose). You can click Create Task from the right-hand pane and start scheduling your task. There are five tabs in the Create Task wizard: General, Triggers, Actions, Conditions and Settings. The first 3 are necessary for all scheduled tasks and the last two give you extra capabilities but are not required to go into for most basic tasks. General has basic information such is name and description of job. Triggers is where you set the timing/frequency of your job. Actions is where you put the location of your VB Script. Arguments can also be passed to the VB Script within the Actions tab. This functionality will not be used in this paper, but can be useful if you want your code to act differently if it's ran manually or from the Task Scheduler. Conditions allows you to set special conditions that must be met for the job to run. Figure 9 shows the basic create task display window.

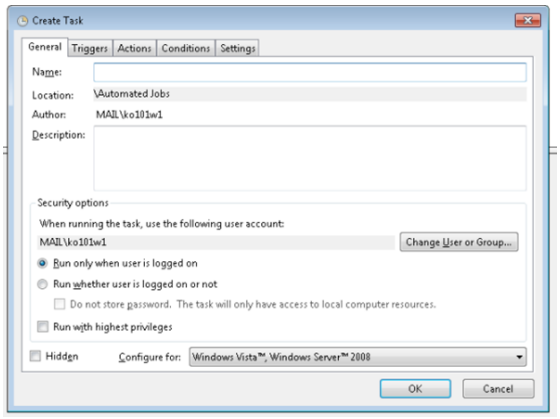

Figure 9

CREATING JOB STREAMS

When you want to run multiple jobs for a specific session you can create a job stream within your SAS code. With special conditions in your SAS code, you can set a VB Script to run daily but have the jobs in the stream only kick off on weekly or daily intervals. As shown in the following code a simple job stream set up will result with multiple include statements that each kick off a sub program:

```
%inc 'C:\Data\Conference_Paper\Automated SAS Code\Daily\Subcode_1.sas';
%inc "C:\Data\Conference_Paper\Automated SAS Code\Daily\Subcode_2.sas";
%inc "C:\Data\Conference_Paper\Automated SAS Code\Daily\Subcode_3.sas";
```

Figure 10

In Figure 10 there are no conditions set so every time the job stream is kicked off all the resulting sub codes will run.

WEEKLY JOBS

When running a job stream every day you may want to have certain programs run only weekly but remain in the main job stream. By utilizing pre-existing SAS functions, one can have a program run on a specific weekday. Logic needed to execute weekly run is:

- %SYSFUNC – macro function that allows code to evaluate SAS functions
- TODAY() – SAS function will return the value of today. If no format is specified it will show the number of days since January 1, 1960
- WEEKDAY() – SAS function that outputs a value from 1 to 7 for the current day of week, e.g. 4 for Wednesday

The following code shows an example of logic that needs sub programs to run on day 4 of the week (Wednesday) and day 2 of the week (Monday):

```
%Macro DOWtdy;
%put today is &sysday;
%If %SYSFUNC(WEEKDAY(%SYSFUNC(TODAY()))) = 4 %then %do;
        %inc 'C:\Data\Conference_Paper\Automated SAS Code\Weekly\Weekly_code1.sas';
%end;
%If %SYSFUNC(WEEKDAY(%SYSFUNC(TODAY()))) = 2 %then %do;
        %inc "C:\Data\Conference_Paper\Automated SAS Code\Weekly\Weekly_code2.sas";
        %inc "C:\Data\Conference_Paper\Automated SAS Code\Weekly\Weekly_code3.sas";
%end;
%Mend DOWtdy;
%DOWtdy;
```

Figure 11

MONTHLY JOBS

There may be times when running a job on a specific day of the month is needed. Following similar logic as in the weekly code you can set programs to run monthly. Instead of utilizing the WEEKDAY function, the DAY function can be used instead. The DAY function is a pre-existing SAS function that returns the day of the month. The following code shows logic used to kick off jobs on either the 1st day of the month or the 26th day of the month.

```
%Macro DOWtdy;
%put today is &sysday;
%If %SYSFUNC(DAY(%SYSFUNC(TODAY()))) = 1 %then %do;
        %inc "C:\Data\Conference_Paper\Automated SAS Code\Monthly\Monthly_code2.sas";

%end;
%If %SYSFUNC(DAY(%SYSFUNC(TODAY()))) = 26 %then %do;
        %inc "C:\Data\Conference_Paper\Automated SAS Code\Monthly\Monthly_code3.sas";

%end;
%Mend DOWtdy;
%DOWtdy;
```

Figure 12

Having the ability to run jobs Daily, Weekly, or Monthly within a single job stream allows for the elimination of multiple job streams and a streamlined process where you only have to check one main code.

CHECKING INPUT FILES

In addition to only running certain jobs on certain days, you may only want to run certain jobs if they have input files that are ready. We will show two options to deal with input files that are not ready: having SAS sleep until the file is ready or skipping that job and sending an automated email saying the file isn't ready. This section will utilize the following SAS functions:

- FILENAME(fileref, file_name) – fileref is a variable you name enclosed in quotes, file_name is the file you want to read in enclosed in quotes.
- FOPEN(fileref) – fileref needs to be named the same thing used in the filename function, again enclosed in quotes.
- FINFO(varname, option) – varname needs to be what you set equal to the fopen function, this is illustrated in the figure below. Option can be one of a few pre-determined SAS options. For our use, we will be using 'Last Modified'.
- TODAY(): will return the value of today. If no format is specified it will show the number of days since January 1, 1960

The first step is to pull in the date modified of the input file. In Figure 13 the variable mdfy will show the date modified but it will be a character variable which makes it not very useful to use. The variable mdfydte reads in mdfy and is converted into numeric form which is needed to compare to other datetimes. The value of mdfydte will be the number of seconds since January 1, 1960.

```
data _null_;
dummy=filename('fle',"C:\Data\Conference_Paper\test.txt");
fid=fopen('fle');
mdfy=finfo(fid,'Last Modified');
mdfydte=input(mdfy,datetime.);
run;
```

Figure 13

In order to create the mdfydte variable we need a few steps prior to read in the file and its information with the filename, fopen and finfo functions. Walking through the code, the variable that is set equal to the filename function doesn't matter what it is called because it is not used again, but you need to set something equal to the filename function. The fopen function will return a value of 0 if the file doesn't exist. The argument in the fopen function has to match the first argument in the filename function. The first argument the the finfo function has to match the variable you set equal to the fopen function.

Now we will want to create a trinary macro variable, filesready, that signifies whether the file exists, the file is updated or the file did not get updated. The code below in Figure 14 illustrates the creation of that variable checking if the file was updated today:

```
today=today()*24*60*60;
if fid=0 then filesready=2;
else if mdfydte >= today then filesready=1;
else filesready=0;
call symput('filesready',filesready);
```

Figure 14

It will be shown later that we will need to call this code so we will combine Figures 13 and 14 into one macro to achieve the following code:

```
%Macro CheckFile;
Data _null_;
dummy=filename('fle',"C:\Data\Conference_Paper\test.txt");
fid=fopen('fle');
mdfy=finfo(fid,'Last Modified');
mdfydte=input(mdfy,datetime.);


today=today()*24*60*60;
if fid=0 then filesready=2;
else if mdfydte >= today then filesready=1;
else filesready=0;
call symput('filesready', filesready);
run;
```

Figure 15

If you wanted to check if the file has been updated in the last X hours you could utilized the time() function. The time() function returns the current time the code is ran so you could could implement something similar to “if mdfydte >= today + time() – 2*60*60” to check if it was modified in the last two hours.

Now that we have a macro variable that shows us the status of our input file we can use that information to control what we choose to run. The following is a simple macro code that allows you to run your desired program if the code is ready, or to run a code that sends out an email notifying you that the files are not ready:

```
%Macro RunJob;
%if &filesready=1 %then %do;
    %inc "C:\Data\Conference_Paper\Code.sas";
%end;
%else %if &filesready=0 %then %do;
    %inc "C:\Data\Conference_Paper\EmailInputFail.sas";
%end;
%else %if &filesready=2 %then %do;
    %inc "C:\Data\Conference_Paper\EmailInputDoesntExist.sas";
%end;
%Mend;
%RunJob;
```

Figure 16

Another option if the input file is not ready is to allow SAS to sleep until the file is ready. This can be accomplished by having code constantly loop to check if the file is ready and then sleep if it’s not ready and run the code if it is ready. The following code in Figure 17 will check if the file is ready by utilizing the %CheckFile macro we created in Figure 15 and if it’s not ready it will sleep for 15 minutes:

```
%Macro WaitForFile;
%do i=1 %to 4;
    %if &filesready=0 or &filesready=2 %then %do;
        data _null_;
        *Sleep for 15 minutes;
        z=sleep(15*60);
        run;
        %CheckFile;
    %end;
    %if &filesready=1 %then %do;
        %inc "C:\Data\Conference_Paper\Testing\FilesReady.sas";
        %let i=4;
    %end;
    %else %if &i=4 %then %do;
        %if &filesready=2 %then %do;
            %inc "C:\Data\Conference_Paper\Testing\EmailInputDoesntExist.sas";
        %end;
        %else %do;
            %inc "C:\Data\Conference_Paper\Testing\FileNeverUpdated.sas";
        %end;
    %end;
%end;
%Mend;
%WaitForFile;
```

Figure 17

After it's done sleeping it will then re-run the check and if the file is ready it will run the code. It will continue this loop 4 times re-running the check each time. If on the fourth time the file still isn't ready then it will check if the file even exists. It will send the automated email based upon whether the file exists or not.

AUTOMATE EMAIL RESULTS

When the job stream has finished running it is convenient to have the completed files automatically send to users. SAS has a pre-existing function: FILENAME that allows SAS to send an email electronically. A good practice is to set up two automatic email programs. One if there is an error in the code and the other to send the automated results. The &syscc macro variable returns a value of 0 if SAS ran successfully with no errors or warnings. It will return a value of 4 if there were warnings but no errors. All other values mean that there was an error within the process. The creation of the &syscc_old macro variable and overwriting of &syscc macro variable allows you to catch the error on one job but is necessary to allow

jobs later in the jobstream to not be incorrectly identified as having an error. The following code shows logic of two paths to take when sending an email:

```
%let syscc_old = &syscc;
%let syscc = 0;
%put &syscc_old;
%put &syscc;

%macro Checklog;
  %if (&SYSCC_old = 0 or &SYSCC_old = 4) %then %do;
   %inc "C:\Data\Conference_Paper\Called_Programs\Email_to_Dist_List.sas";
  %end;
  %else %do;
   %inc "C:\Data\Conference_Paper\Called_Programs\Email_Error_in_Log.sas";
  %end;
%mend Checklog;
%Checklog
run;
quit;
```

Figure 18

If the process errors you can have it send an email to yourself so no user receives inaccurate data. Within each email code you can add to:, cc:, subject: and the attachment. By putting put statements you can also add in details that one would like users to read. The following code shows an example of logic to output an email:

```
%let blank=;

filename outbox email;
data _null_;
   file outbox
      to=('lindsey.whiting@kohler.com' 'joseph.kaiser@kohler.com')
      cc=("lindsey.whiting@kohler.com")
      subject="Project 1 Data "
        attach=("C:\Data\Conference_Paper\Automated SAS Code\Daily\Project.xlsx");
     put 'Please review the attached file for your daily metrics.';
     put &blank;
     put 'If you have any problems with this link please contact ';
     put 'Lindsey Whiting';

run;
```

Figure 19

Following this email logic can help streamline error checking and getting data out to users.

The above code will prompt you to allow the program to send an email on your behalf, which obviously isn't ideal for automation. In order to allow it to always send emails without the pop up you will need to adjust your config file. The config file is a text file named SASV9.CFG and will be saved in the same spot where SAS is installed. Figure 20 shows the code needed in your config file; however, this config might change depending on the email server used on your computer.

```
-EMAILHOST <enter host name in quotes>
-EMAILPORT <enter portnumber>
-EMAILSYS <enter email system>
-EMAILID <logon id>
-EMAILPW <password>
```

Figure 20

CONCLUSION

Automating SAS job streams with the power of VB script can make the morning routine of running jobs much smoother. Kicking off SAS using VB script and a task scheduler is one method to available to have a structured job automation. Having logic implemented to run jobs at daily, weekly, or monthly intervals allows the ability to use one main job stream for the bulk of programs. It streamlines making one job stream instead of having multiple job streams to run on different days. Checking for input files and looping to wait for selected files allows job streams to continue efficiently. With built in email automation one can know if files didn't update in a timely manner but can continue to let the job stream run. Automatically sending users emails when things run successfully and sending an error email when things do error can help improve process flow and help negate the effects of sending inaccurate information mistakenly. This baseline logic can help be the initial set up for any process where setting up efficient job streams is needed.

REFERENCES

Delwiche, Lora D., and Susan J. Slaughter. 2008. The Little SAS Book: A Primer, Fourth Edition. Cary, NC: SAS Institute Inc.

SAS. %SYSFUNC and %QSYSFUNC Functions. Accessed July 30, 2018. http://support.sas.com/documentation/cdl/en/mcrolref/61885/HTML/default/viewer.htm#z3514sysfunc.htm

SAS. FOPEN Function. Accessed July 30, 2018. http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000209683.htm

SAS. FILENAME Statement. Accessed July 30, 2018. http://support.sas.com/documentation/cdl/en/hostunx/61879/HTML/default/viewer.htm#email.htm

SAS. FILENAME Statement. Accessed July 30, 2018. http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000211297.htm

SAS. FINFO Function. Accessed July 30, 2018. http://support.sas.com/documentation/cdl/en/lefunctionsref/63354/HTML/default/viewer.htm#p0cpuq4ew0dxipn1vtravlludjm7.htm

SAS. Usage Note 19767: Using SAS software to send SMTP email. Accessed July 30, 2018 http://support.sas.com/kb/19/767.html

Pagé, Jacques. 2004. Automated Distribution of SAS® results. SAS Institute Inc. 2004. Proceedings of the Twenty-Ninth Annual SAS® Users Group International Conference. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

Scott Bohenstengel is a Kohler employee who helped create some of the VB Script processes that are described in this paper, in particularly the custom logging.

CONTACT INFORMATION

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

Joey Kaiser
Kohler Company
Joseph.Kaiser@kohler.com

Lindsey Whiting
Kohler Company
Lindsey.Whiting@kohler.com

Monday Tips

1. Creating a VB Script to kick off your SAS job and schedule the VB Script in Task Scheduler.
2. Use SAS code to check if input files are updated.
3. Use SAS code to send automated emails.