Paper 221-31

Sleepless in Wherever – Resolving Issues in Scheduled Jobs

Faron Kincheloe, Baylor University, Waco, TX

ABSTRACT

Even with the capability of Windows scheduler or other utilities to schedule your SAS® jobs to run automatically, life is not always a dream. Dataset names and variables that must be changed periodically along with perpetually growing log files can keep you awake at night. This paper discusses techniques to solve these problems using macro variables and various date formats. It will also discuss a method for time-stamping data when you have non-standard time periods. Perhaps you've lain awake at night worrying about whether your jobs completed successfully or maybe you worry about the effects of leaving your computer on all the time. This paper will describe a relatively simple way to send an e-mail message when a job fails and will show you how to automatically shut off your computer when your jobs finish running. These techniques can be implemented using the tools available with a typical SAS® installation on the Windows platform without the need for complex macros or additional software.

INTRODUCTION

This paper addresses several issues that, at first glance may appear to be unrelated. However, these are all issues that have arisen when attempting to schedule SAS® programs to run automatically. This discussion is primarily directed toward the Windows platform. The solutions involving dates are relevant to any operating system.

ARCHIVED DATASETS

One possible reason for scheduling a SAS® program is to routinely create reports or archive snapshots from live data. How can you do this or even make it worthwhile to schedule jobs if the dataset names have to be changed every day, week, or month? SAS® provides a number of date functions and formats that will return all or part of the current date and time. These can be used along with any combination of arithmetic and logic functions to assign date specific values for your program. These values are stored in macro variables that are called for text substitution throughout the program. The sample code below constructs a dataset name for monthly receipts. The program is scheduled to run every night. The dataset is overwritten every night and will contain all month-to-date receipts. On the first day of every month, a new dataset is created for the current month. The datasets are named with the convention R2005asofJul.

```
%let thisyr=%sysfunc(today(), year4.);

data _null_;
name1="R&thisyr"||'asof'||put(today(),MONNAME3.); /*create name text*/
call symput('filename1',name1);/* store name in a macro variable */
run;

data archive.&filename1;
```

The %let statement creates a macro variable named thisyr. %SYSFUNC is a special macro function that provides immediate access to SAS® language functions. In this case it returns the value of TODAY() which is the current date. DATE() is an alternative way of writing the TODAY() function. By default, both functions return the number of days since January 1, 1960. A date format is usually required to present the date as a useable, readable value. Here, the format year4. is used to produce only the 4 digit year portion of the current date. Now the macro variable can be used anywhere in the program by preceding the name with an ampersand (i.e.: &thisyr). If the value of the macro variable needs to be part of a quoted string, the variable name must be within double quotes as shown on the name= line above. Otherwise, the macro variable should be referenced without any quotation marks as shown in the last line of code above.

The data _null_ step above shows how the full dataset name is created. Most of these functions could be nested together but are shown here in a data step for clarity. The concatenation operator || is used to join together the various pieces of our dataset name. Again the TODAY() function is called but this time only the month portion is used. The MONNAME3. format produces the first three letters of the name of the month. If the length of the format is not specified, the full name of the month will be supplied. However, short month names will be padded with leading spaces. These spaces must be removed by enclosing the entire PUT function inside a COMPRESS() function. The CALL SYMPUT() function does, within a DATA step, essentially the same thing the %let statement does outside of the DATA step. It creates a macro variable named filename1 and assigns its value based on the contents of the variable name1.

It is recommended that the following line be placed at or near the top of your programs:

```
options symbolgen;
```

This option displays in the log file the results of macro variables as they are resolved within your program. A sample log entry is shown below. This can be very useful in determining whether your macro functions are working properly.

```
SYMBOLGEN:  Macro variable FILENAME1 resolves to R2005asofJul
```

GROWING LOG FILES

Unless otherwise specified, SAS® creates, in a default location, a log file bearing the same name as the SAS® program. This log file is overwritten each time the program is executed. If your scheduled jobs run frequently and you do not review the log each time, it may be desirable to retain the logs as an audit trail or for comparison purposes if something goes wrong. By using the PROC PRINTTO statement, within the program, to specify the name and location of the log file SAS® appends the log to any log that is currently in the file. This statement should be placed at the very beginning of the program because SAS® will continue to send output to the default log file until this statement is executed. The down side of this method is that the log file can quickly become large and unwieldy if you are running a long program. It can also be quite cumbersome to remove old, unneeded log entries. This can be overcome using a technique similar to the one described above for archived datasets. By using a variable filename that changes every month, we can limit the size of the file to the log entries for one month. We can also delete a month's worth of log entries by simply deleting the log file for the appropriate month. The program code is shown below:

```
%let curmy = %sysfunc(today(),MONYY7.);

FILENAME LOGFILE "E:\wh8jobs\inquiry&curmy..log";

proc printto log=logfile; run;
```

This time, as matter of preference, the MONYY7. format was used. This format produces the month and year portions of the current date as the first three letters of the month name and the full 4 digit year. Notice that two periods are used in the FILENAME statement. The first period delimits the macro variable name and the second period becomes a literal part of the file name. If only one period is used, it delimits the macro variable name and log becomes part of the filename rather than a filename extension. When using PROC PRINTTO to redirect the log file, be sure to include the line below at the bottom of the program. This statement closes the log file and redirects all output back to the default locations.

```
proc printto; run;
```

NON-STANDARD TIME PERIODS

In the business world all time periods do not necessarily coincide with the calendar. If the fiscal year begins in July, the first month is 7 and not 1. This can present some challenges when trying to order and group periodic data for reporting purposes. This section presents a scenario from the academic world and a solution using date functions, date math, and macro variables.

At Baylor University, the admissions tracking cycle begins and ends on the twelfth class day of the fall semester. The academic year typically has 53 weeks because the first week of this year is also the last week of the previous year. The year number corresponds to the year in which a student enters Baylor as a new freshman. Benchmark statistics are recorded each weekday so that point-in-time comparisons can be made from one year to the next. Calendar dates do not work in this situation because October 17 may be a Sunday one year and a Monday the next and February 29 only occurs once every 4 years. A customized date value is created to overcome this phenomenon. The date follows the pattern 2006-05.2 where 2006 is the recruiting year, 05 is the week, and 2 is the day. It is important that the week contain the leading zero because this is a text field and weeks are grouped together across years based on a substring of the text value. Each record of historical data includes a field containing the customized date. The code below shows how this date is created. The program has to be manually updated once each year to change the start date and the academic year. There are too many factors that influence the start date so that it is not practical to attempt to calculate it. The academic year can be calculated as will be shown later, but for simplicity it is supplied in this example.

```
%let startdate='06sep05'd; /* Twelfth class day of most recent fall semester */
%let acadyr=2006; /* Academic year for tracking */

data _null_;
```

```
WKSPAN = INTCK('week',&startdate,today())+1;
dateval="&acadyr"||'-'||put(wkspan,z2.)||'.'||put(today(),eurdfdn1.);
call symput('BUdate',dateval);
run;
```

Several new items are introduced in this example. The INTCK function is a special date function that returns the number of time intervals between two dates or datetime values. In this case the interval is a week that begins on Sunday. INTCK accepts a wide range of interval values and offsets. For instance, week.2 specifies week intervals with Monday as the boundary. See SAS® Help for a more extensive listing of interval values. When we begin running the program, the current day and the start date occur in the same week so INTCK returns a value of 0. We add 1 to this value because we want our week numbering to begin with 1. The format z2. ensures that our week value is always 2 characters wide and is padded with a leading zero when necessary. The eurdfdn1. format is a special date format that presents the day as a single digit number of 1 for Monday through 7 for Sunday. When the DATA step is complete we have a macro variable named BUdate containing our custom date that can be used anywhere in the program.

An academic year spans parts of two calendar years with term numbers that include the calendar year. This makes the term another non-standard time period that must be dealt with in automated programs. A census dataset containing demographic and academic data on each university student is created each term. The name of the dataset includes the term number. The spring term ends in 10 while the fall term ends in 30 (i.e.: 200530). The program code below was developed using the techniques described above to programmatically determine the name of the most recent census dataset based on the current month. The format month2. in the second line presents the current month as a number up to two digits long. The %sysevalf function on the third line evaluates arithmetic and logical expressions using floating-point arithmetic. This function enables us to calculate the value of the previous year outside a DATA step.

```
 %let thisyr=%sysfunc(today(), year4.);
 %let thmon=%sysfunc(today(), month2.);
%let lastyr=%sysevalf(&thisyr-1);

 data _null_;
    if &thmon = 1 then do;
     call symput('curr_term',put(&lastyr,4.)||'30');
    end;

    if &thmon ge 2 and &thmon le 8 then do;
         call symput('curr_term',put(&thisyr,4.)||'10');
    end;

    if &thmon ge 9 and &thmon le 12 then do;
          call symput('curr_term',put(&thisyr,4.)||'30');
    end;
  run;

 %let name2=St_census_&curr_term;/*file name for census dataset*/
```

TO BATCH, OR NOT TO BATCH

Batch files date back to the DOS days before Windows when personal computer functions were executed by entering statements on a command line. Batch files provided the user with the ability to enter a group of commands automatically. A batch file is nothing more than a text file containing a series of commands that are executed by the operating system. The name of the batch file needs to end with the extension ".bat" so the operating system knows how to process the file. The Windows operating system has greatly diminished the need for this functionality but it can still be very useful on occasion.

The statement below is from a batch file created by SAS® Data Warehouse Administrator. Data Warehouse Administrator first creates a SAS® program. Then it creates a batch file to launch the SAS® program. It then uses the Windows AT command to schedule the batch job. The batch file is necessary because the AT command has a 255 character limit which is easily exceeded by long path names and job parameters.

```
"E:\Progra~1\SAS\SAS9~1.1\sas.exe " -sysin "E:\wh8jobs\A000000U.sas"   -log
"E:\wh8jobs\A000000U.log"
```

One of the main advantages of the batch file is that the filename essentially becomes shorthand for any number of commands and system parameters. The example above shows SAS® being launched with two system options. The sysin option specifies the program to be executed when SAS® is launched. The log option specifies the name and location of the log file. Other popular options include config which specifies an alternate configuration file and print or noprint which control the destination and printing of SAS® procedure output.

Another advantage of the batch file is that it provides a straightforward method for monitoring SAS® jobs or error conditions. This will be explained in more detail later in this paper.

One of the main disadvantages of the batch file is that it requires a working knowledge of drive letters, path names, and Windows command prompt functions. The batch file does not support the use of long filenames to which we have become accustomed in the later versions of Windows. One relatively simple method of determining the short form of the path involves opening a command prompt window. Click Start -> Run. Type cmd and click OK. You will need to maneuver to the drive and folder where your program is located. If you need to change drives, type in the new drive letter followed by a colon such as e: and press enter. Type cd\ and press enter to get to the root of the drive. Type dir /x and press enter. In the right half of the window you will see the short names followed by the long names. You may need to scroll back up if the drive contains a lot of entries. Type cd followed by the short name of the next folder in the path and press enter. Continue this process until you have determined all of the short names in your path. For more information about filenames consult the Microsoft Knowledge Base article listed in the reference section of this paper.

Another significant disadvantage of the batch file is the loss of control should SAS® hang after being scheduled using a batch file. When you schedule multiple SAS® programs in a batch file, each is opened in a separate SAS® session. You can right click on a scheduled task and then click End task. This will terminate the batch but will not terminate any underlying SAS® processes. The SAS® processes will have to be terminated using Task Manager. Task Manager is invoked by right clicking on the task bar typically at the bottom of the window. Click the Process tab and look for processes named SAS. It can be almost impossible to identify the hung process if you are running other SAS® programs or services such as an object spawner or SAS® Share. Highlight the hung process and click the End Process button. NOTE: This can produce undesirable consequences if you end the wrong process. If you are scheduling jobs on a server, you should obtain the assistance of your server administrator to terminate the SAS® process.

Batch files can be created using either Windows Notepad or the SAS® Program Editor. Type in the desired commands for your batch and then click File -> Save As. Browse to the desired location. Change the “Save as type” selection to All Files. Enter a file name and make sure it ends with .bat. Click the save button.

If you do not need to take advantage of the error monitoring provided by a batch file you might opt to create a SAS® program that can be scheduled to launch other SAS® programs sequentially. The text below is from a SAS® program named Friday.sas. As the name suggests, this job runs every Friday night. Each %INCLUDE statement specifies a SAS® program. A program can be temporarily removed from the schedule by inserting an asterisk at the beginning of the line as demonstrated by the last line of code. Notice that this does not require the use of short filenames. All programs run in a single SAS® session. If a program hangs, you can right click on the scheduled task and click End Task and the hung session will be terminated.

```
*<<< This program is used to submit several jobs in sequence for weekly processing>>>;

%INCLUDE 'E:\wh8jobs\PAWeekly Counts.sas';
*%INCLUDE 'E:\wh8jobs\mailrept.sas';
```

YOU’VE GOT MAIL

SAS® software enables you to send e-mail by way of a DATA step, SAS® procedure, or SCL. SAS® supports three types of electronic mail interfaces:
MAPI (Mail API, such as Microsoft Exchange, Lotus Notes Version 5, and Eudora)
VIM (Vendor Independent Mail, such as Lotus cc:Mail and Lotus Notes Version 4)
SMTP (Simple Mail Transfer Protocol).
Before you can use this feature, you will need to configure SAS® to communicate with your e-mail server. This is done by adding the lines below to the SAS® configuration file. The configuration file is named SASV9.CFG and is typically located in C:\Program Files\SAS\SAS 9.1\nls\en. The example shown below is for the SMTP interface. Search SAS® Help for e-mail to obtain additional information on setting up the other interfaces.

```
/* set up email server information*/
-emailsys SMTP
-emailhost mailserver.myorg.com
-emailport 25
```

When a SAS® session ends, it sends a status code to Windows to indicate the condition in which the session ended. A code of 0 means the session ended normally. A code of 1 indicates the session ended with a warning. A code of 2 means the session ended with an error. Codes of 3 and higher indicate other types of failures such as abends. The batch file processing provides the capability of monitoring this status code and reacting accordingly. The two statements below are the contents of a batch file that uses SAS® to send an e-mail if the first program ends with an error. The logic of the second statement means that if errorlevel is greater than or equal 2, execute the emailjob program. The status code level is evaluated by errorlevel. If the SAS® program completes normally or with a warning, no e-mail message is sent.

```
E:\Progra~1\SAS\SAS9~1.1\sas.exe -sysin 'K:\faron\errorprog.sas' -log
'k:\faron\errorprog.log'

if errorlevel 2 E:\Progra~1\SAS\SAS9~1.1\sas.exe -sysin 'K:\faron\emailjob.sas'
```

The code below represents the entire contents of the emailjob.sas program referenced above. The filename statement sets up the message parameters for recipient and subject. The DATA _null_ step supplies the message body and sends the message. The %sysfunc function is used to include the time and date of failure in the body of the e-mail message.

```
filename mymail email "Faron_Kincheloe@baylor.edu"
         subject="Job Failure";
data _null_;
   file mymail;
   put "One or more scheduled jobs ended with an error on
%sysfunc(datetime(),datetime.)";
run;
```

Another use of the SAS® e-mail feature is to automatically send as attachments reports that are generated by SAS® jobs. The code below is a small part of a large program that extracts data from a transactional system. The extraction program checks for certain types of errors that must be corrected and stores these errors in a temporary dataset named DATA_ERRORS. The first three lines of code use the %sysfunc function to open the dataset, retrieve the number of observations attribute and close the dataset. The number of observations is stored in the macro variable named nm_num. This value will be tested later in the code and if it is greater than 0, the error file will be sent as an e-mail attachment. Put statements are used to either send the message or abort the message as explained in the comments below.

```
%let dsid=%sysfunc(open(DATA_ERRORS));
%let nm_num=%sysfunc(attrn(&dsid,nobs));
%let rc=%sysfunc(close(&dsid));
filename mymail email from='Faron_Kincheloe@baylor.edu'
       to='Person_Whocares@baylor.edu' subject='Errors to be Corrected'
       content_type='text'
       attach=('F:\MyJobs\data_errors.rtf');
       data _null_;
       file mymail;
  if &nm_num >0 then do;
     /* Send the message. */
  put '!EM_SEND!';
       end;
       else do;
       /* Abort the message if there are no errors */
  put '!EM_ABORT!';
       end;
/* This line is required to keep the message from being sent again when the run
statement is executed.*/
  put '!EM_ABORT!';
  run;
```

THE COMPUTER THAT NEVER SLEEPS

The computer must remain powered up in order for scheduled tasks to run. You do not have to be logged in but the computer must be on. This is obviously not an issue if the jobs are scheduled on a server since servers are on all the time. However, it may not be desirable to leave a workstation computer running all the time. Windows has a shutdown command that can be scheduled independently to run after all jobs have had time to finish, executed at the end of a scheduled batch, or executed at the end of a scheduled SAS® program. To perform a scheduled shutdown, enter the command below in the Run line of the Scheduled Task properties or on the last line of the batch file that you have scheduled.

```
C:\WINDOWS\system32\shutdown.exe -s
```

Note that on some computers, depending on where the operating system is installed, you may need to replace WINDOWS with WINNT in the above statement.

If you are scheduling the SAS® program without the use of a batch file, there are three options available for executing a shutdown at the end of the SAS® program. These options can actually be used to execute any Windows command from within SAS® and each provides slightly different options. However, for the purposes of shutting down the computer, there appears to be no difference in the three methods. The simplest of the three is to use the "X" statement. This provides immediate execution of the specified command.

```
X "C:\WINDOWS\SYSTEM32\shutdown.exe -s";
```

The SYSTASK command provides added functionality such as the ability to monitor the status of the task. Unlike the X statement, SYSTASK runs these commands as asynchronous tasks, which means that these tasks execute independently of all other tasks that are currently running. Asynchronous tasks run in the background, so you can perform additional tasks while the asynchronous task is still running. These added features are of little use when shutting down the computer.

```
systask command "C:\WINDOWS\SYSTEM32\shutdown.exe -s";
```

The behavior of the CALL SYSTEM routine is similar to that of the X statement and the SYSTASK command. It is useful in certain situations because it can be conditionally executed. This can be especially useful if you want to test for certain conditions before shutting down the computer or executing the specified Windows command.

```
data _null_;
call system ("C:\WINDOWS\SYSTEM32\shutdown.exe -s");
run;
```

CONCLUSION

You can overcome many of the obstacles to scheduling SAS® programs by using the standard features of Windows and SAS®. By using the techniques described in this paper, you can schedule your jobs to run automatically and rest comfortably while they run.

REFERENCES

Carpenter, Art, 2005 "Looking for a Date? A Tutorial on Using SAS® Dates and Times" SUGI 30 Conference Proceedings, paper #255-30

Kincheloe, Faron, 2005 "While You Were Sleeping - Scheduling SAS® Jobs to Run Automatically" SCSUG 2005 Conference Proceedings

Microsoft Knowledge Base Article, "Converting Filenames from NTFS to FAT (8.3) Convention" http://support.microsoft.com/default.aspx?scid=kb;en-us;101601, November 20, 2003.

ACKNOWLEDGEMENTS

CONTACT INFORMATION

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

Faron Kincheloe
Baylor University
One Bear Place #97032
Waco, TX 76798
Phone: (254) 710-8835
Email: Faron_Kincheloe@baylor.edu