Paper 334-2013

Running Routine Programs? How to Process Your Inputs Faster

Jason Wachsmuth, Pearson

ABSTRACT

The purpose of this paper is to demonstrate how to process multiple input files and routine programs with one click through the use of two macros. These macros combined with the %INCLUDE statement, CALL SYMPUT, and SCAN functions in a control program will eliminate updating various %LET statements, and physically opening, running, and closing each program. Most people who process data for routine operations will appreciate this solution.

INTRODUCTION

Instead of manually running a program for every step, it is more efficient to run multiple external programs from a main program by utilizing the %INCLUDE statement (Mendez, 2011). This paper will expand on that concept by using two macros: SAS and INPUT. The SAS macro calls the programs to be included; the INPUT macro calls the inputs to be processed and is wrapped around SAS.

This solution allows a person to process inputs and sequence dependent programs in the proper order with one click. But besides implementing these two macros, it is necessary to eliminate the manipulation of %LET statements. This can be achieved by incorporating specific information in the input filenames and capturing the values using the CALL SYMPUT and SCAN functions. The information below will show you how this technique has evolved, and discuss the advantages and disadvantages across the three stages.

STAGE 1

The %LET statements sometimes contain various parameters, and updating them all for numerous inputs is time-consuming. Although, using them makes for a quick program to write.

```
%LET DIR = C:\DATA\NEWCO\ACH\MAY13;    /*DIRECTORY*/
%LET PROJ = ACH;    /*ACH*/
%LET ADMIN = MAY13; /*MAY13*/
%LET SUBJ = MATH;   /*MATH, HIST, WRIT, READ, SCIE*/
%LET TEST = IS;     /*PT, IS*/
%LET GRADE = GR03;  /*GR03-GR08*/
%LET CVER = V00;    /*VERSION OF CTM INPUT, V00 IS INITIAL FILE*/
%LET PVER = P01;    /*VERSION OF PTM OUTPUT, STARTS AT P01*/
%LET XVER = X01;    /*VERSION OF XML OUTPUT, STARTS AT X01*/
                    /*NOTE: XML IS DERIVED FROM THE PTM. NOT ALL UPDATES IMPACT XML*/

%MACRO SAS(PROGRAM);
       %INCLUDE "&DIR.\SASCODE\&PROGRAM..sas";
%MEND SAS;
 %SAS(1_READIN_V01)          /*READ IN INPUT*/
*%SAS(2_COMPARE_V01);        /*COMPARE NEW VERSION INPUT TO OLD VERSION INPUT*/
 %SAS(3_CREATE_REPORTS_V01)  /*CREATE REPORTS, RUN AS NECESSARY*/
 %SAS(4_CREATE_PTM_V01)      /*CREATE PTM OUTPUT, RUN EVERY TIME*/
 %SAS(5_CREATE_XML_V01)      /*CREATE XML OUTPUT, RUN AS NECESSARY*/
```

Consider if the current task is to complete the initial processing for all math grades and tests. It would take 12 iterations to define the %LET statements and run. That would not be very difficult because all the input and output version numbers would be aligned. On the other hand, imagine processing updates for a variety of subjects, grades, and tests. That is where the complexity increases because there are more conditions to accommodate. However, efficiency can be increased by using two functions and a filename strategy.

But before moving on, note the SAS macro above which is used throughout this paper. It allows one program to be run, all programs, or any combination of programs as appropriate with one click, instead of one by one.

STAGE 2

Using the CALL SYMPUT and SCAN functions with structured input filenames relieves the programmer by reducing the number of %LET statements to define. Once the structure is in place, it will be useful for future administrations, and the concept will be easy to modify for other projects.

```
%LET DIR = C:\DATA\NEWCO\ACH\MAY13;          /*DIRECTORY*/
%LET CTM = CTM_ACH_MAY13_MATH_IS_GR03_V00;   /*INPUT FILE*/
%LET XVER = X01;   /*VERSION OF XML OUTPUT, STARTS AT X01*/

DATA _NULL_;
       DELIM1 = '_';
       CALL SYMPUT("PROJ", SCAN("&CTM.", 2, DELIM1));
       CALL SYMPUT("ADMIN", SCAN("&CTM.", 3, DELIM1));
       CALL SYMPUT("SUBJ", SCAN("&CTM.", 4, DELIM1));
       CALL SYMPUT("TEST", SCAN("&CTM.", 5, DELIM1));
       CALL SYMPUT("GRADE", SCAN("&CTM.", 6, DELIM1));
       CALL SYMPUT("CVER", SCAN("&CTM.", 7, DELIM1));
       CALL SYMPUT("PVER", PUT(SUM(1,SUBSTRN(SCAN("&CTM.",7,DELIM1),2,2)),Z2.));
RUN;
*%PUT _USER_;
%MACRO SAS(PROGRAM);
       %INCLUDE "&DIR.\SASCODE\&PROGRAM..sas";
%MEND SAS;
 %SAS(1_READIN_V01)
*%SAS(2_COMPARE_V01);
 %SAS(3_CREATE_REPORTS_V01)
 %SAS(4_CREATE_PTM_V01)
 %SAS(5_CREATE_XML_V01)
```

What makes this method helpful is that a person can copy the input filename, paste it into the control program, and the functions will capture the values dynamically. However, there are still inefficiencies. Processing a variety of updates in this manner still requires updating %LET statements, submitting run as many times as there are inputs, and it is hard to keep track of which files were processed.

STAGE 3

An even better solution is to wrap the INPUT macro around the SAS macro. INPUT parameters include the input filename and an output version number. Defining CTM and XVER macro variables in the INPUT macro up front eliminates the need for continuously updating %LET statements and processing inputs one by one.

```
%MACRO INPUT(CTM, XVER);
       %LET DIR = C:\DATA\NEWCO\ACH\MAY13;    /*DIRECTORY*/

       DATA _NULL_;
              DELIM1 = '_';
              CALL SYMPUT("PROJ", SCAN("&CTM.", 2, DELIM1));
              CALL SYMPUT("ADMIN", SCAN("&CTM.", 3, DELIM1));
              CALL SYMPUT("SUBJ", SCAN("&CTM.", 4, DELIM1));
              CALL SYMPUT("TEST", SCAN("&CTM.", 5, DELIM1));
              CALL SYMPUT("GRADE", SCAN("&CTM.", 6, DELIM1));
              CALL SYMPUT("CVER", SCAN("&CTM.", 7, DELIM1));
              CALL SYMPUT("PVER", PUT(SUM(1,SUBSTRN(SCAN("&CTM.",7,DELIM1),2,2)),Z2.));
       RUN;
       *%PUT _USER_;
       %MACRO SAS(PROGRAM);
              %INCLUDE "&DIR.\SASCODE\&PROGRAM..sas";
       %MEND SAS;
        %SAS(1_READIN_V01)
        %SAS(2_COMPARE_V01)
       *%SAS(3_CREATE_REPORTS_V01);
        %SAS(4_CREATE_PTM_V01)
        %SAS(5_CREATE_XML_V01)
```

```
%MEND INPUT;
 %INPUT(CTM_ACH_MAY13_MATH_IS_GR03_V01, X02)
*%INPUT(CTM_ACH_MAY13_MATH_IS_GR04_V00, X01);
 %INPUT(CTM_ACH_MAY13_MATH_IS_GR05_V01, X02)
 %INPUT(CTM_ACH_MAY13_MATH_IS_GR06_V01, X02)
*%INPUT(CTM_ACH_MAY13_MATH_IS_GR07_V00, X01);
*%INPUT(CTM_ACH_MAY13_MATH_IS_GR08_V00, X01);
```

This technique is valuable for a few different reasons. First, routine programs can process inputs with one click. Second, it is flexible; programs and inputs can be bypassed by commenting them out. Last, INPUT call statements track what was processed, what was not processed, and what the current version numbers are.

Conversely, there are a few caveats when using this technique. First, setup could be timely due to developing a filename structure for the inputs, or aligning the programs with standardized macro variable names. Second, a program could create a data driven global macro variable that retains its value while processing one input to the next. For example, a certain global macro variable value was not supposed to exist for the latter input, but because it was populated from the earlier input, the value was carried over. Third, this style is not meant to run a single program step by step. Fourth, there can be too many INPUT call statements to conveniently type. Last, there may be macro variables that cannot be incorporated into the input filename design.

ADDRESSING STAGE 3 CAVEATS

Here are ways to handle the cautions mentioned above when using this approach:

- The code provided in this paper shows the evolution of this style which should give the programmer good direction on how to proceed with the initial setup.
- If a program creates global macro variables, they can be deleted by implementing the %SYMDEL macro function at the end of the code (Harvey & Parker, 2011).
- One way to run code step by step is to define the macro variables within the DATA _NULL_ step and INPUT macro in the particular program. Another way is to create a macro in the program around the DATA steps that should not be processed. Then, omit the macro call statement, and next, run the control program as usual.
- Instead of copying and pasting numerous filenames for the INPUT call statements, this task can be completed programmatically. First, use the X command to create a text file listing of all the inputs in the specified folder (Harvey & Parker, 2011). Second, read in the listing and perform the necessary DATA steps to format the INPUT macro call statement values appropriately. Last, print out the variable in a text file so that all the INPUT call statements can be copied and pasted into the control program.
- If there are macro variables that cannot be incorporated into the filename, then additional parameters will have to be added to the INPUT macro and the values defined manually.

CONCLUSION

This paper shows how an efficient control program evolved by integrating the SAS and INPUT macros, %INCLUDE statement, and CALL SYMPUT and SCAN functions. Although there are a few cautions for the user to consider before implementing, a list of solutions is provided to address them. The advantage of this technique is the ability to process multiple input files at once because the need to update various %LET statements, and physically open, run, and close each program is eliminated.

REFERENCES

Harvey, B., & Parker, R. (2011, April). Using SAS® to enter variables in a DOS window. Paper presented at the annual meeting of SAS Global Forum 2011, Las Vegas, NV. Paper retrieved from http://support.sas.com/resources/papers/proceedings11/118-2011.pdf.

Mendez, L. (2011, April). Create a main program to run multiple SAS® programs: Utilizing the %INCLUDE statement. Paper presented at the annual meeting of SAS Global Forum 2011, Las Vegas, NV. Paper retrieved from http://support.sas.com/resources/papers/proceedings11/100-2011.pdf.

ACKNOWLEDGMENTS

I am thankful to have a great group of colleagues to work with for over the past five years: Miky Lee, Aaron McVay, Brandon Harvey, Brian Wrobel, Cindi Kreiman, Julie Miles, and Bob Parker.

RECOMMENDED READING

Cody, R. (2007, April). An introduction to SAS® character functions. Paper presented at the annual meeting of SAS Global Forum 2007, Orlando, FL. Paper retrieved from http://www2.sas.com/proceedings/forum2007/217-2007.pdf.

Slaughter, S. J., & Delwiche, L. D. (2004, May). SAS® macro programming for beginners. Paper presented at the annual meeting of SAS Global Forum 2004, Montréal, Canada. Paper retrieved from http://www2.sas.com/proceedings/sugi29/243-29.pdf.

KEYWORDS

%INCLUDE, CALL SYMPUT, SCAN, %LET, &&var&i alternative, batch processing alternative, automate

CONTACT INFORMATION

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

Jason Wachsmuth
Pearson
2510 North Dodge Street
Iowa City, IA 52245
(319) 339-6400 ext. 216005
jason.wachsmuth@pearson.com
http://www.pearsonassessments.com/pai/ai/research