Paper 119-2011

Using CALL SYMPUT to Simplify

W. Jodi Auyuen, BlueCross BlueShield of Minnesota, Eagan, MN

ABSTRACT

Automation is a commonly used word these days for companies who want to streamline their processes and for SAS® programmers who want to make significant improvements in their programs to increase efficiency, and to make work and life simple. The SAS system provides us with powerful and handy tools for automation. It helps to avoid potential human errors, to cut down editing time, to guarantee accuracy and consistency, to provide fast turnaround time, to require less human intervention, and last but not least, it offers great opportunities to gain knowledge and experience to share with colleagues.

This paper demonstrates one example using CALL SYMPUT, INTNX, and SAS macros to reduce manual steps so as to consolidate five to seven SAS programs into one SAS program. It simplifies the work process and life!

BUSINESS BACKGROUND OF MY EXAMPLE

This application reports members' additional dollar amounts in each month after their accumulated amount exceeded their group threshold limit. Each group renews their contracts annually, with a renewal date and threshold limit. Each report runs through 15 months, so a group could receive more than one report using both the current contract and previous year's contract.

OLD APPROACH

- Each month, a programmer had to update an Excel spreadsheet (just for documentation) to keep track of which groups are current, new, or retired, based on each group's renewal date, since not all groups renew their contract in the same month.

- He had a SAS program for each group, in which he edited date ranges each month.

```
%LET FromIncr='2010-07-01;
%LET FromPaid=201007;
%LET ToIncr='2011-04-30';
%LET ToPaid=201104;
%LET Threshold=250000;
%LET GroupName=NV;
```

- When a group received a new contract, he had to create a SAS program and edit these %LET statements.
- After the program was used for 15 months, he retired that SAS program.

SOLUTION

During the assignment transition, in which I took over the reporting task, this previous programmer kept reminding me, "you need to remember…", "you need to check…"
I noticed most of the programs were quite similar; at least the layout and the majority of the programs are the same. All programs used the same %LET statements to define each group's date ranges, threshold dollar amount, and GroupName. Business rules are quite straightforward. Since the "FromIncr" and "FromPaid" dates can be derived from the RenewalDate and "ToIncr" and "ToPaid" can be derived from RenewalDate or today's date, it's a perfect candidate to automate the process with data-driven macro variables and an iterative process. I can also translate those "you need to remember/check…" rules into IF…THEN statements.

NEW APPROACH

I am dividing my new approach into two parts:
First, in the data-driven part, I use CALL SYMPUT, INTNX, and SAS functions to define SAS macro variables to

create a report for a group with a renewal contract. Secondly, I use an iterative process to loop through the list of groups to create reports.

This paper focuses on the CALL SYMPUT, INTNX and a few SAS functions, but I thought it would be beneficial to briefly cover the iterative process to make the story complete.

PART I: to use the CALL SYMPUT routine and INTNX to create macro variables for each group, based on its renewal date or today's date.

Here are quick references from SAS Online Documentation.

- **CALL SYMPUT** routine either creates a macro variable whose value is information from the DATA step or assigns a DATA step value to an existing macro variable.
  Syntax: CALL SYMPUT(*macro-variable*, *value*);
- **DATE()** retrieves today's date.
- **INTNX** function increments dates by intervals. INTNX computes the date or datetime of the start of the interval a specified number of intervals from the interval containing a given date or datetime value.
  Syntax: INTNX(*interval, from, n <, alignment >*)
  where:
  interval is a character constant or variable containing an interval name.
  from is a SAS date value (for date intervals) or datetime value (for datetime intervals).
  n is the number of intervals to increment from the interval containing the from value.
  alignment controls the alignment of SAS dates, within the interval, used to identify output observations. Can take the values BEGINNING|B, MIDDLE|M, or END|E.
- **PUT** function**:** we can use the PUT function to convert numeric values to character values
- **TRIM** function removes trailing blanks from character expressions and returns one blank if the expression is missing.
  Syntax: TRIM(*argument*)

Let us apply these in the following examples.

Example 1: Let us start with the CALL SYMPUT routine. We can create a macro variable to replace
%LET u_THRESHOLD = 250000;

```
DATA _null_;
Threshold=250000;
 call symput('u_Threshold',trim(Threshold));
RUN;
```

Threshold is defined as 250000 in the DATA step and u_Threshold is created by TRIM(Threshold) as 250000. Personally, I like to name my user defined macro variables, starting with u_ in order to better identify whether they are automatic macro variables or user defined macro variables.

Example 2:

```
DATA _null_;
 x="07APR2011"d;
 call symput('u_ToPaid_SASdate',intnx('month',x,0)); ❶
 call symput('u_ToPaid_worddate',put(intnx('month',x,0),worddate.));  ❶
 call symput('u_ToPaid8',put(intnx('month',x,0),yymmddn8.));
 call symput('u_ToPaid6',put(intnx('month',x,0),yymmn6.));

 call symput('u_PreviousMonth_FirstDay',put(intnx('month',x,-1),yymmddn8.));
*-- some people prefer to use worddate for report title/footnote --*;
 call symput('u_PreviousMonth_LastDay',put(intnx('month',x,-1,'end'),worddate.));
*-- Sunday is the first day of the week --*;
 call symput('u_PreviousWeek_FirstDay',put(intnx('week',x,-1,'b'),  mmddyys10.)); ❷
*-- Saturday is the last day of the week --*;
 call symput('u_PreviousWeek_LastDay',put(intnx('week',x,-1,'e'),date9.));
RUN;

%put &u_ToPaid_SASdate. &u_ToPaid_worddate. &u_ToPaid8. &u_ToPaid6.
     &u PreviousMonth FirstDay. &u PreviousMonth LastDay.
```

The LOG shows:

```
SYMBOLGEN:  Macro variable U_TOPAID_SASDATE resolves to                    18718
SYMBOLGEN:  Macro variable U_TOPAID_WORDDATE resolves to                   April 1, 2011
SYMBOLGEN:  Macro variable U_TOPAID8 resolves to                           20110401
SYMBOLGEN:  Macro variable U_TOPAID6 resolves to                           201104
               &u_PreviousMonth_FirstDay. &u_PreviousMonth_LastDay.
SYMBOLGEN:  Macro variable U_PREVIOUSMONTH_FIRSTDAY resolves to            20110301
SYMBOLGEN:  Macro variable U_PREVIOUSMONTH_LASTDAY resolves to             March 31, 2011
               &u_PreviousWeek_FirstDay. &u_PreviousWeek_LastDay.;
SYMBOLGEN:  Macro variable U_PREVIOUSWEEK_FIRSTDAY resolves to             03/27/2011
SYMBOLGEN:  Macro variable U_PREVIOUSWEEK_LASTDAY resolves to              02APR2011

18718 April 1, 2011 20110401 201104   20110301     March 31, 2011    03/27/2011 02APR2011
```

❶ If we want to create a macro variable as the first day of the current month; in the INTNX function, we can use 'month' as 'interval', x (="07APR2011"d) as 'from', and 0 as 'n'. u_ToPaid_SASdate is then resolved to 18718, which is the SAS Date for April 1, 2011. SAS Date is numeric and works for date calculations.
However, 18718 as a date is not very intuitive to non SAS programmers, so you can use the PUT function with your preferred Date Format to make it meaningful. Among those Date Formats, WORDDATE is nice for report titles or footnotes.

❷ U_PreviousWeek_FirstDay is another example using the INTNX function to create a macro variable as the first day of the previous week. In that INTNX function, we use 'week' as 'interval', x as 'from', and -1 as 'n'. The week starts with Sunday. U_PreviousWeek_FirstDay is then resolved to 03/27/2011, with mmddyys10. Date Format. It is another popular choice for report titles or footnotes.

Example 3: Back to the application that I consolidated.

```
%LET ReportGeneration=0; ❹

DATA _null_;
 call symput('u_ToPaid_fixed',put(intnx('month',date(),0),yymmn6.)); ❸

 call symput('u_ToPaid',put(intnx('month',RenewalDate,14-&ReportGeneration), yymmn6.));  ❹

RUN;
```

❸ In order to replace %LET ToPaid=201104, I used the SAS DATE () function to derive a text string like 201104 (assuming that I am running on April 7, 2011). INTNX('month', DATE (),0) is resolved to 18718, which is the SAS date for April 1, 2011.
And adding the PUT function converts the numeric value 18718 to a formatted date and stores it as a character 201104 for the macro variable u_ToPaid_fixed.

❹ There were times that I had to re-create previous reports. By adding a %LET statement in this example, I could just change from `ReportGeneration=0` to `ReportGeneration=1` to create the previous generation reports. Since we want the report to run for 15 months, `14 - &ReportGeneration` covers that business rule.

Example 4: The DATA step below counts the number of rows in the GroupList_2010 data set and stores the value as a macro variable u_row_cnt, which you can use for the iterative process in Part II.

```
DATA _null_;
 SET GroupList_2010 end=eof;
 count+1;
 if eof then CALL SYMPUT('u_row_cnt',count);
RUN;
```

PART II:

Before getting into the iterative process, let me introduce the control data set that is driving the application. This data set lists the groups and their parameters, like GroupName, RenewalDate, and Threshold. Personally, I like to have a flag variable for business rules (for example, WHERE RetiredFlag ne 'retired') or for testing.

GroupList_2010.sas7bdat

GroupSeq	GroupName	RenewalDate	Threshold	RetiredFlag
1	MN	1/1/2009	100000	retired
2	AL	1/1/2009	150000	retired
3	NV	7/1/2009	200000	retired
4	WA	9/1/2009	100000	retired
5	MN	1/1/2010	150000	
6	AL	1/1/2010	200000	
7	NV	7/1/2010	250000	
8	WA	9/1/2010	200000	
9	MN	1/1/2011	150000	
10	AL	1/1/2011	200000	

In my Example 4 above, u_row_cnt is the macro variable which counts the number of rows in my control data set, GroupList_2010.sas7bdat. As long as RetiredFlag is not "retired", it will execute the %OneReport macro (in the SAS Program session below) for each observation (group).

SAS PROGRAM:

```
*-- To recreate previous month's report, I only need to change this %LET statement, from %LET
ReportGeneration=0; to %LET ReportGeneration= 1; --*;

%LET ReportGeneration=0;

%MACRO OneReport;

*-- reading in data and calculated individuals' total DollarAmount. --*/;
PROC SQL;
CREATE TABLE MyData AS
   SELECT *
   FROM CONNECTION TO DB2
  (SELECT
            Element1
           ,DollarAmount
            ,ElementN
   FROM     DataSource
   WHERE
           &&u_where_clause&i.
            AND DATEINCR BETWEEN &&u_FromIncr&i. AND &&u_ToIncr&i.
           AND PAIDMNTH BETWEEN &&u_FromPaid&i. and &&u_ToPaid&i.
  );
QUIT;

PROC SORT data= MyData; by Element1;RUN;

PROC MEANS data= MyData noprint;
 by Element1;
 var DollarAmount;
 output out=DataSumm (drop=_type_ _freq_) sum=;
RUN;

*-- Keeping the ones that exceeded Threshold --*;
```

```
DATA DataSumm2;
 SET DataSumm;
 if DollarAmount > &&u_Threshold&i.;
RUN;

PROC SQL noprint;
 select count(*) into:obscount
 from DataSumm2;
QUIT;

*-- Creating a permanent dataset if DollarAmount exceeds threshold amount --*;
%IF &obscount.>0   %then %do;
DATA mydata.data_&&u_GroupName&i.&&u_FromPaid&i.._&&u_ToPaid&i.;
  SET DataSumm2;
RUN;

*-- If previous month file exists, bring it in to compare. --*;

%IF %sysfunc(exist(mydata.data_&&u_GroupName&i.&&u_FromPaid&i.._&&u_ToPaid_prev&i.))
%then %do;

DATA PreviousMonthDataSet;
 SET mydata.data_&&u_GroupName&i.&&u_FromPaid&i.._&&u_ToPaid_prev&i.;
 … code not shown …
RUN;
%END;

%else %do;
 … code not shown …
%END;

*-- Below is one way of writing PDF files to your network drive automatically. --*;
*-- using ODS to output PDF file to DirectoryLocation --*;
FILENAME NOVELL FTP "&&u_GroupName&i.&&u_FromPaid&i.._&&u_ToPaid&i...PDF"
  CD="//DirectoryLocation/&u_ToPaid&i."
  HOST=”HostName”
  USER="UserName"
  PASS="UserPassWord"
  DEBUG;

ODS PDF BODY=NOVELL NOTOC STYLE=PRINTER;

PROC REPORT DATA=FINAL SPLIT='*';
 … code not shown …
RUN;

ODS PDF CLOSE;
%END;

%MEND OneReport;;

*-- Read in parameters for each group.  --*;
DATA GroupList;
 SET GroupList_2010 end=eof;
 if RetiredFlag ne ‘retired’;
 count+1;
 if eof then call symput('u_row_cnt',count);
RUN;

%MACRO RunAllReports;
DATA _null_;
 SET GroupList;
 call symput('u_FromIncr'||left(trim(put(_n_,2.))),"'"||put(intnx('month',RenewalDate,0),
yymmddd10.)||"'");
 call symput('u_ToIncr'||left(trim(put(_n_,2.))),
"'"!!put(intnx('month',  RenewalDate,11-&ReportGeneration,'end'),YYMMDD10.)!!"'");
 call symput('u_FromPaid'||left(trim(put(_n_,2.))),put(intnx('month',RenewalDate,0),yymmn6.));
 call symput('u_ToPaid'||left(trim(put(_n_,2.))),put(intnx('month', RenewalDate,14-
&ReportGeneration),yymmddn6.));
 call symput('u_Threshold'||left(trim(put(_n_,2.))),trim(Threshold));
 call symput('u_where_clause'||left(trim(put(_n_,2.))),’cat(STATE=”’,trim(GroupName),’”’));
```

```
RUN;
*-- iterative process --*;
%local i;
%DO i =1 %TO  &u_row_cnt.;
 %OneReport;
%END;
%MEND RunAllReports;

%RunAllReports;
```

The iterative process will require having different sets of macro variable names for each observation (group); otherwise a set of macro variables for one observation will be overwritten through the iteration. We can number each set of macro variables by concatenating the macro variable with a data counter variable, _n_. The chart below explains why and how I concatenate `'FromIncr'` with `||left(trim(put(_n_,2.)))`.

Group Seq	Group Name	Renewal Date	Threshold	Retired Flag	_n_	FromIncr'|| left(trim(put(_n_,2.)))	"""||put(intnx('month',RenewalDate,0), yymmddd10.)||"""
1	MN	1/1/2009	100000	retired			
2	AL	1/1/2009	150000	retired			
3	NV	7/1/2009	200000	retired			
4	WA	9/1/2009	100000	retired			
5	MN	1/1/2010	150000		1	FromIncr1	'2010-01-01'
6	AL	1/1/2010	200000		2	FromIncr2	'2010-01-01'
7	NV	7/1/2010	250000		3	FromIncr3	'2010-07-01'
8	WA	9/1/2010	200000		4	FromIncr4	'2010-09-01'
9	MN	1/1/2011	150000		5	FromIncr5	'2011-01-01'
10	AL	1/1/2011	200000		6	FromIncr6	'2011-01-01'

The macro %RunAllReports loops through the data set GroupList_2010 to create numbered sets of macro variables. The macro %OneReport is executed within a %DO loop for each observation in the control data set (GroupList_2010). It summarizes total DollarAmount by member, only keeps members with DollarAmount exceeding the threshold amount, and compares it to the previous month's DollarAmount, if it is applicable. At the end of each iteration, it creates a PDF and routes it to its designated location.

A FEW TIPS

- Don't try to write complicated SAS macros all at once. It will be easier to start by hard-coding your potential parameters and use this result as the baseline (Carpenter, 2010). Next, replace the hard-coded values with the %LET statements. Finally, replace %LET statements with SAS macros.
- Like many people have recommended, turning on the following OPTIONS. They will help you to see how your macros were resolved, and this will help you to debug problems.

  ```
  OPTIONS MPRINT MLOGIC SYMBOLGEN;
  ```

- %PUT statements display useful information in the SAS log.

  To display specific macro variables:
  Use %PUT statement to display specific macro variables. Like my Example 2 above.

  ```
  %put &u_ToPaid_SASdate. &u_ToPaid_worddate. &u_ToPaid8. &u_ToPaid6.
       &u_PreviousMonth_FirstDay. &u_PreviousMonth_LastDay.
       &u_PreviousWeek_FirstDay. &u_PreviousWeek_LastDay.;
  ```

  To display all macro variables, including automatic macro variables and user defined macro variables:
  `%PUT _ALL_;`

  This also gives you a list of automatic macro variables that are available. Frequently used are: SYSUSERID, SYSDATE/SYSDATE9, SYSTIME.

To display just user defined macro variables:

```
%PUT _USER_;
```

- I name my user defined macro variables starting with u_. This approach makes it easier to review my program or for the person who takes the task over to easily distinguish between automatic and user defined macro variables.
- I learned to write comments, like business requirements or brief instruction, in my programs both for my own benefit and also assist when transitioning the program to a different person.
- I have to admit that I never considered macro variable collisions until Mr. Carpenter pointed it out and shared his paper with me. (Carpenter, 2005) Keeping macro variables local to the macro being used prevents stepping into another macro symbol table while executing this macro.

CONCLUSION

If your application has repeated blocks of code or a series of similar programs, you will benefit from replacing them with SAS Macros, data-driven approaches, and iteration techniques. It avoids potential human errors, cuts down editing time, guarantees accuracy and consistency, provides fast turn around time, minimizes human intervention, and increases readability. Last but not least, you gain knowledge and experience to share with your colleagues and the opportunity to present your great ideas at a SAS Global Forum.

REFERENCE

SAS, OnlineDoc 9.1.3
Carpenter, Arthur L., 2010, “Manual to Automatic: Changing Your Program’s Transmission”, http://www.sas.com/offices/NA/canada/downloads/presentations/Van10/Manual.pdf
Carpenter, Arthur L., 2005, "Make 'em %LOCAL: Avoiding Macro Variable Collisions" http://www.caloxy.com/papers/62_TT04.pdf

ACKNOWLEDGMENTS

I would like to thank Dave Thul, manager of Health Economics at BlueCross BlueShield of Minnesota who encouraged me to implement creative ideas into my work. I also want to express my appreciation to Art Carpenter for his mentoring and valuable feedback on this paper. Many thanks are also due to Dr. Holly Rodin, Senior Healthcare Analyst of Health Economics at BlueCross BlueShield of Minnesota and Sana Husevold, Healthcare Analyst of Health Economics at BlueCross BlueShield of Minnesota for reviewing this paper.

CONTACT INFORMATION

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

W. Jodi Auyuen
BlueCross BlueShield of Minnesota
3400 Yankee Drive, R259
Eagan, MN 55121
Work Phone: 651.662.1639
E-mail: Wuong_j_auyuen@bluecrossmn.com