Paper AD114

The Power of SAS® Macro Programming – One Example

Milorad Stojanovic
RTI International

ABSTRACT

When we are using SAS macro programming, macro tools and features can make our life more difficult at the beginning and a lot easier at the end. We should envision the work of macros in different combinations of input data and relationships between variables. Also macro code should prevent the processing data if 'critical' files are missing. In this paper we present examples of creating macro variables, using one or more ampers (&), creating dynamic SAS code, using %Sysfunc, %IF, %THEN, %ELSE and delivering flexible reports. Code is data driven by using macro programming tools.

KEYWORDS Macro, %sysfunc, %if, %then, %else, %do, %to

INTRODUCTION

SAS Macro language gives us a powerful tool to create reusable, dynamic and more complex SAS programs. Concept of macro programming (languages) is not a new one. At one point in time I was working with Assembler for IBM 360 mainframe computer and he was at that time he was using Macro Assembler for IBM 360. As demonstrated by the examples the concept is very old but has proven very useful over the last 45+ years.

Macro variables are almost always preceded by an ampersand (&) (see more in lit <1>). For more complex use we can use, more than one ampersand as a prefix. That opens the world of very sophisticated SAS macros as well as SAS programs.

%SYSFUNC

SAS Macro language provides the %SYSFUNC function – a powerful tool for a large number of applications. Let's say you need to insert date (and time) of processing into the title of a report(s).

Old way

```
Data _null_ ; length dateC $ 10 timec $ 8 ;
Date = date() ; time = time() ;
dateC = put(date, mmddyy10.) ; timeC = put(time, time8.) ;
call symput('dateC', datec) ; call symput('timeC', timeC) ;
run ;
title "&dateC. &timeC." ;
```

New way,

```
%let today_DT = %sysfunc(date(), yymmdd10.) %sysfunc(time(), time8.) ;
Title "&today_DT." ;
```

"2014-07-28 11:21:51"

Also, you could avoid the use of &today_DT global macro variable by put "sysfunc" directly into title.

```
Title "%sysfunc(date(), yymmdd10.) %sysfunc(time(), time8.)" ;
```

This 'new' way is more elegant and if you are using macro variable for formats (date and time) it is extremely flexible.

```
%let datefm = date9 ; %let timefm = time5 ;
Title "%sysfunc(date(), &datefm..) %sysfunc(time(), &timefm..)" ;
```

"28JUL2014 11:43"

For more info please take a look at lit <7> or in SAS Macro language manuals. A lot of information is also available online.

DYNAMIC PROGRAMS

Very often there is a need to execute parts of your program and to skip some steps based on the outcome of the step(s) prior to that point in the program. SAS macro programming language again supplies us with useful tools. Please look at the

%IF, %THEN, %ELSE, and %END

It looks similar with SAS statements IF, THEN, ELSE and END but with a big difference. These macro statements would dynamically allow or reject the execution of few SAS statements, several DATA or PROC steps or any combination of before based on condition (value of data). This they would be visible for SAS compiler or it would be skipped during the compilation.

```
%IF &out1 = Yes %THEN %DO ;
        Proc print data = report ;
        Var var1 var2 var3 ;
        Run ;
END;
%ELSE %DO ;
        Proc print data = report ;
        Title "There is no data for this run" ;
        Run ;
%END ;
```

In this simplistic example you can see the advantage of %IF, %THEN, %ELSE SAS macro statements. If macro variable &out1 has value of Yes (this means there are data to report) the report would be printed or if there is no data, second PROC step would be executed and user would have info what is occuring with processing.

Same as DO loops in SAS there are similar statements in macro language like %DO, %TO, %END. You can refer to lit (1) in references or in SAS macro language manuals.

Let's say you would like to read in and create new SAS data set by initially unknown data sets. Number of data sets would be determined by macro variable &NumDataSets prior to creating NEW SAS data set. You can do this by using macro DO loop statements.

```
Data NEW ;
Set %DO I = 1 to &NumDataSets. ; Data&i. %END;;
Run ;
```

By combining previously listed SAS Macro statements you could create highly dynamic and flexible SAS programs.

ONE EXAMPLE FROM REAL LIFE

Imagine frustration you would feel after looking at the SAS LOG of failed a program because the following has occured:

```
2800 %macro name (table name(XXXX),project(XXXX), libref.dataset(XXXX));
ERROR: CLI cursor fetch error: [Oracle][ODBC]Invalid datetime format.
NOTE: The SAS System stopped processing this step because of errors.
```

What happened? What should you do? More than sure you should not be happy at all or even frightened if it's happened in mission critical application.

Taking you back in time for a moment, forty years ago the author was involved in quite a different project dealing with mainframes from different vendors (IBM, Honeywell, and Univac). At that time problem was in discrepancy in word size (32 bit IBM vs 36 bit Honeywell). It had influence on precision. At first glance what is the connection between different word sizes and dates. In this moment there is no connection but it is very similar pattern of doing things differently by different companies in their own way. When dates are in question we are talking about today three main

SW companies – Microsoft, Oracle, and SAS. There is no direct link (between word sizes and dates) but one similarity still exists. Each of the large SW companies was/is doing many software internal solutions in their own specific way. Let's look at a range of dates and how the different companies handle it.

	Microsoft <3>	SAS <5>	Oracle <4>
Minimum Date	01/01/1753	01/01/1582	01/01/4712 BC
Maximum Date	12/31/9999	12/31/20000	12/31/9999

Table above tells us that some dates are the same (just two out of six) and others are quite different. This means that a date that is valid in one system may not be valid in the other. If you have a situation where data was entered into Oracle with no range checks and later used in SAS you would have the introduction to the perfect storm. Obviously we are using data from different software systems and let's imagine that the data in the example was entered into Oracle but later the data was used in SAS system.

Date **2013/10/22** was entered clearly by mistake as **0013/10/22** into Oracle. It did not show any note/message at the time of data entry. But, when you try to read that date into SAS serious problems would arise. Depending how it is handled, it could halt your processing and have serious implications if you need to produce final results imminently and you don't have time to correct invalid dates in DB.

Why has this happened? It was simply an INVALID date (completely out of range for SAS date formats) and SAS expects to deal with VALID dates only.

After this occurred a decision was made to check and correct values of all date-time variables in all affected projects. This means in all projects with a mix of data input (and with no range check). To be more efficient checking was done on project by project basis in order of priority. First to be run the most urgent projects and later other projects.

Now let's look at how it could be prevented in the proper way. It is possible to skip 'invalid dates' but in this case user is losing some data (maybe critical data). To avoid losing data, application was created to check dates in all active projects. As it was earlier said data were entered into Oracle data base from various external sources.

Crucial step was to download data from Oracle tables into SAS by changing format of DATETIME variables to STRING variables. Doing so the "**ERROR: CLI cursor fetch error: [Oracle][ODBC]Invalid datetime format.**" was avoided and looking for invalid dates commenced. The powerful tool was **TO_CHAR** Oracle function, see lit. <8>.

```
proc sql;
  connect to ORACLE as oradb (user=userID. password=password path=path) ;
  create table Table1 as
    select * from connection to oradb
    (select project_ID,ID, to_char(DateNameVar1, 'yyyy/mm/dd') DateNameVar1
    from Table1 where project_ID = &project.) ;
  disconnect from oradb;
quit; run;
```

Now DateNameVar1 contains string value corresponds to DateNameVar1 numeric value in Oracle table. SAS would easily accept any string value and could handle it with no problems. In our example it was **string '0013/10/22'** which was quite easy for further handling and no notes or warnings were issued.

Organization of Oracle tables used in this application.

Project	Project_Table	Table
Project_ID	Project_ID	Table_ID
Status	Table_ID	ID
Participants	Etc.	var1, var2,
Start_DT		varN
Etc.		

Column Status had values: '1-Active', '2-Archived', '3-Canceled'. For efficiency only tables for Active projects were checked.

Macro Problem_dates was created with several internal steps so data processing to be executed as efficient as possible with avoidance of processing of non-existent data.

```
%Problem_dates(project=&project., Location=C:\SESUG2014\&project.\, userID=XXX,password=SESUG_14)
```

Step 1.
As soon as %Problem_Dates was called value of Project was checked against Project table. If project_ID was wrong message was printed and processing was terminated.
If project existed but project was not 'Active' message was printed and processing was terminated. Also if the number of sample members was 0, all further processing was terminated and message was issued.

```
%obscnt(dsn=project,numobs=ds_exist)

%IF &ds_exist. = 0 %THEN %DO ;
  %put ***** ------------------------------------------------------------- ;
  %put ***** project=&project. does not exist. Probably typo,    ;
  %put *****   please correct it and run program again.          ;
  %put ***** ------------------------------------------------------------- ;
  %GOTO TERM_Processing ;
%END ;
```

Step 2.
From variables Start_DT and by using today() SAS function &Beg_DT and &End_DT macro variables were created.

Step3.
Accessing Project_Table by Project_id all table names participated in the project were read in and corresponding macro variables were created. Project_tbl had only tables belonged to project_ID = &project.

```
data _null_ ;
set project_tbl end = eof nobs = numobs ; Ind + 1 ; length indC $ 2 ;
indC = compress(put(ind, 2.)) ;
call symput('table_name'||indC, table_ID); %* Builds macro variables for each ORACLE table ;
if eof then do ;
  call symputx('Num_Tables', numobs) ; %* The number of tables for particular project ;
end ;
run ;
```

Step 4.
If &table_NameindC had 0 rows processing for that table was stopped and the next table was checked.

Step5.
For each non-zero table proc content was created. This was one of ways to get the important characteristics of each table (NAME, TYPE, LENGTH, LABEL, FORMAT).

```
  data db.Contents_&&table_name&i. ;
    set db.Contents_&&table_name&i.(where=(FORMAT='DATETIME')) ;
  run ;
```

If number of DATETIME variables was 0 processing was stopped – program was going to end of loop and next table was examined. If there were DATETIME variables new set of macro variable(s) were created.

```
  %obscnt(dsn=db.Contents_&&table_name&i., numobs=NumDTVars)

  %IF &NumDTVars. = 0 %THEN %DO ;
     %put ------------------------------------------------------- ;
     %put ********* There is no DATETIME variables in &&table_name&i. ;
     %put ------------------------------------------------------- ;
  %GOTO End_Loop ;
  %END ;
  %ELSE %DO ;
  data _null_ ;
    set db.Contents_&&table_name&i. end = eof ;  length ind 8 indC $ 2 ;
    retain ind 0 indC '0 ';
    if FORMAT = 'DATETIME' then do ;
      ind + 1 ; * Counter of DATETIME variables ;
      indC = left(put(ind, 2.)) ; call symput('var_name'||indC, NAME) ;
    end ;
    if eof then do ; call symput('Num_vars', indC) ;  end ;
```

```
  run ;
 %END ;

Step  6.
After downloading table with DATETIME variables final check of date values started.
  data db.problem_dates_&&table_name&i.;
    set &&table_name&i;
     %DO L = 1 %TO &Num_Vars. ;
        %IF &Num_Vars. = 1 %THEN %DO ;
           if ((" " < &&var_name&L.  < "&Beg_DT.") or  (&&var_name&L. > "&End_DT." )
           then output db.problem_dates_&&table_name&i. ;
        %END ;
        %ELSE %DO ;
           %IF &L. = 1 %THEN %DO ; if %END ;
           %IF &L. = &Num_Vars. %THEN %DO ;
              ((" " < &&var_name&L.  < "&Beg_DT.") or (&&var_name&L. > "&End_DT.")
              then output db.problem_dates_&&table_name&i. ;
           %END ;
           %ELSE %DO ;
              ((" " < &&var_name&L.  < "&Beg_DT.") or
               (&&var_name&L. > "&_End_Date_.") or
           %END ;
        %END ;
     %END ;
  run;

%obscnt(dsn=db.problem_dates_&&table_name&i., numobs=Num_Inv_Inc)

Step 6.

Creation of output report(s) for sending to group responsible for correcting data.

   %IF &Num_Inv_Inc. = 0 %THEN %DO ;
      data nothing ;
      text = "There is no Invalid/Incorrect dates in &&table_name&i." ;
      run ;
      proc report data = nothing nowd ;
      title "Problem dates for project = &project. in table &&table_name&i. ";
      columns text ;
      define text / display f=$60. "Note to user" ;
      run;
   %END ;

   %ELSE %DO ;
      proc report data = db.problem_dates_&&table_name&i. nowd ;
      title "Problem dates in project = &project., table &&table_name&i. Start date = &Beg_DT. ";
         columns project ID %DO M = 1 %TO &Num_Vars. ; &&Var_Name&M. %END ; ;
         define project              / group   f=9.     "project" ;
         define ID                   / display f=9.     "ID" ;
         %DO M = 1 %TO &Num_Vars. ;
            define &&Var_Name&M. / display f=$10.   "&&Var_Name&M." ;
         %END ;
      run;
      ods rtf startpage = NO ;
      ods listing ;
   %END ;

   %End_Loop: ;
   %END ;      %* End of DO LOOP for tables - outer loop. ;

   %TERM_Processing: ;
      %put ---------------------------------------------------------------------------- ;
```

```
      %put ********* Processing of data for project_ID = &project was done. ;
      %put ---------------------------------------------------------------------------- ;
%mend Problem_dates ;
```

Example of report.

```
  %IF &Num_Inv_Inc. = 0 %THEN %DO ;
    data nothing ;
    text = "There is no Invalid/Incorrect dates in &&table_name&i." ;
    run ;
    proc report data = nothing nowd ;
    title "Problem dates for project = &project. in table &&table_name&i. ";
    columns text ;
    define text / display f=$60. "Note to user" ;
    run;
  %END ;

  %ELSE %DO ;
    proc report data = db.problem_dates_&&table_name&i. nowd ;
    title "Problem dates in project = &project., table &&table_name&i. Start date = &beg_date. ";
       columns project_ID ID &Beg_dt
       %DO M = 1 %TO &Num_Vars. ;
          &&Var_Name&M.
       %END ; ;
       define project_ID          / group   f=9.     "project" ;
       define ID                  / display f=9.     "ID" ;
       define &beg_dt             / display f=yymmdds10. "Start_DT" ;
       %DO M = 1 %TO &Num_Vars. ;
          define &&Var_Name&M. / display f=$10.   "&&Var_Name&M." ;
       %END ;
    run;
    ods rtf startpage = NO ;
    ods listing ;

    proc export data = db.problem_dates_&&table_name&i.
          OUTFILE= "&Location.Problem_dates_&project._Table_&&table_name&i...xls"
          DBMS=EXCEL REPLACE;
    run;
  %END ;

  %End_Loop: ;
  %END ;      %* End of DO LOOP for tables - outer loop. ;

  %TERM_Processing: ;
     %put ---------------------------------------------------------- ;
     %put ********* Processing of data for project = &project was done.   ;
     %put ---------------------------------------------------------- ;
%mend Problem_dates ;
```

Example of report (Assume today is 2014/10/22).

PROBLEM_DATES_Table_SESUG_2014, Table T555 Start date = 2012/12/25

Project_ID	ID	Start_DT	DateVar1	DateVar2	
11111	99998	2012/12/25	2013/02/20	0013/03/20	← invalid date
11111	99999	2012/12/25	2013/08/19	2014/12/31	← incorrect date (future date)
11111	99999	2012/12/25	2013/05/24	2012/10/10	← incorrect date (date before start date)

LIMITATIONS

Code was tested to run under SAS 9.3 (TS1M2) under Windows 7. It was not tested on other platforms or other versions/releases of SAS.

CONCLUSION

Through Macro programming SAS gives to users a powerful tool. Programs written with SAS macros are more flexible and ready for more than one time usage. Users are relieved from unnecessary changes hence potential errors are lower. Also it requests programming discipline and planning ahead. At the end user has better product and usually more other users could use it with no extra programming or modifying. Later payback is significant.

ACKNOWLEDGEMENTS

The author would like to thank Jean Lennon and Christopher Alexander from the Education Studies Division (RTI) for all their help, comments and support in producing this paper.

REFERENCES

1. Art Carpenter, “Carpenter’s Complete Guide to the SAS Macro Language” Second edition, 2005, SAS Institute, Cary, NC
2. G. Polya, “How to Solve It” A new Aspect of Mathematical Method, 1973, Princeton University Press, Princeton, NJ
3. MS SQL date range: http://msdn.microsoft.com/en-us/library/ms187819.aspx
4. Oracle date range: http://docs.oracle.com/cd/A97630_01/appdev.920/a96624/03_types.htm
5. SAS date range: https://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a002200738.htm
6. %SYSFUNC, SAS documentation, http://support.sas.com/documentation/cdl/en/mcrolref/62978/HTML/default/viewer.htm#n1mix0b315l1dpn0zmhebdzifjne.htm
7. Chris Yindra, “%SYSFUNC – The Brave New Macro World”, SUGI23, 1988 http://www2.sas.com/proceedings/sugi23/Advtutor/p44.pdf
8. TO_CHAR: http://docs.oracle.com/cd/B19306_01/server.102/b14200/functions180.htm

DISCLAIMER

All opinions and suggestions stated in the paper on how to do processing of data do not necessarily reflect the opinions and suggestions of the Education and Workforce Development (RTI International).

CONTACT INFORMATION

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

Milorad Stojanovic
Education and Workforce Development
RTI International
3040 Cornwallis Rd
RTP, NC, 27909
Phone : (919) 541-7376
E-mail: milorad@rti.org

TRADEMARK INFORMATION