Writing Flexible SAS® Codes: Exploring the value of global macro variables, conditional statements and %SYSFUNC
Victoria E. Porterfield

ABSTRACT

Many SAS codes are developed to be run routinely and not always on a case by case basis. Therefore, it becomes more important to write flexible code which requires only minor changes, if any, by the user and that allows for a wider range of possibilities. For instance, a code may be run on a monthly basis to examine the relationship between the purchase of a particular item and the various age groups of the buyers. However, what happens if there are no buyers from one of the age groups for a particular month and as a result unwanted warnings, notes, and/or errors appear in the log? Instead of rushing through and revising the code after the fact, it may be more beneficial to be proactive in safeguarding your code.

This paper explores various methods that can be used to increase the flexibility of routine codes. These methods include using global macro variables, conditional statements, and %SYSFUNC to execute DATA and PROC steps within a macro. This paper assumes the user is familiar with macro programming.

SAMPLE DATA

The dataset, WORK.VideoGameSales, as shown in Figure 1 will be used throughout this paper. WORK.VideoGameSales contains two character variables, Month and AgeGroup, and one numeric variable, Sales.

<table>
<thead>
<tr>
<th>Month</th>
<th>AgeGroup</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>June</td>
<td>&lt;18</td>
<td>216</td>
</tr>
<tr>
<td>June</td>
<td>18-25</td>
<td>153</td>
</tr>
<tr>
<td>June</td>
<td>26-35</td>
<td>87</td>
</tr>
<tr>
<td>June</td>
<td>36-45</td>
<td>35</td>
</tr>
<tr>
<td>June</td>
<td>45&lt;</td>
<td>10</td>
</tr>
<tr>
<td>July</td>
<td>&lt;18</td>
<td>199</td>
</tr>
<tr>
<td>July</td>
<td>18-25</td>
<td>134</td>
</tr>
<tr>
<td>July</td>
<td>26-35</td>
<td>69</td>
</tr>
<tr>
<td>July</td>
<td>36-45</td>
<td>27</td>
</tr>
<tr>
<td>July</td>
<td>45&lt;</td>
<td>6</td>
</tr>
<tr>
<td>August</td>
<td>&lt;18</td>
<td>181</td>
</tr>
<tr>
<td>August</td>
<td>18-25</td>
<td>126</td>
</tr>
<tr>
<td>August</td>
<td>26-35</td>
<td>70</td>
</tr>
<tr>
<td>August</td>
<td>36-45</td>
<td>13</td>
</tr>
</tbody>
</table>

The following code creates the dataset above:
DATA WORK.VideoGameSales;
INPUT Month $06. AgeGroup $06. Sales;
DATALINES;
June <18  216
June 18-25 153
June 26-35  87
June 36-45 35
June 45<   10
July <18  199
July 18-25 134
July 26-35  69
July 36-45 27
July 45<   6
August <18 181
August 18-25 126
August 26-35 70
August 36-45 13
;
RUN;

CREATING GLOBAL MACRO VARIABLES

Global macro variables are very useful as they can be created anywhere in the SAS code and are available to be referenced throughout the entire SAS session.

The %LET statement is one way to define a global macro variable. Below you will see the general syntax of the %LET statement:

%LET <macro-variable> = <value>;

- **Macro-variable** is either the name of a macro variable or a text expression. This must follow the same naming conventions as a typical SAS variable.
- **Value** is a character string or a text expression. Omitting value produces a null value (0 characters). Leading and trailing blanks in value are ignored.

As with any macro variable, you must precede the name of the macro variable with an ampersand (&) each time it is referenced after it is defined. Additionally, a period must be used to separate a nonblank value when macros variables are combined with other characters or numbers in a string. Multiple macro variables do not need a period if combined together. Also, double quotation marks are required for a macro variable to resolve correctly as a variable value. This can be seen in the WHERE statement in the DATA step example shown below. Keep in mind that a %LET statement can only define one macro variable at a time.

The %LET statement can come in handy when running routine analyses. For instance, if WORK.VideoGameSales is a cumulative dataset, but routine analyses are run monthly, it may be useful for your code to include a %LET statement with a macro variable that can be edited each month rather than rewriting the code. For example:

%LET Month = August;

DATA WORK.&Month;
SET WORK.VideoGameSales;
WHERE Month = "&Month";
RUN;

If the macro variable named in the %LET statement already exists, the macro variable value will resolve to the newest value without warning or error. This becomes more evident by using the system option SYMBOLGEN:
options SYMBOLGEN;

%LET Month = July;
%LET Month = August;

DATA &Month;
SET WORK.VideoGameSales;
WHERE Month = "&Month";
RUN;

From the SAS log:

23   %LET Month = July;
24   %LET Month = August;
25
SYMBOLGEN:  Macro variable MONTH resolves to August
26   DATA &Month;
27   SET WORK.VideoGameSales;
28   WHERE Month = "&Month";
SYMBOLGEN:  Macro variable MONTH resolves to August
29   RUN;

NOTE: There were 4 observations read from the data set WORK.VIDEOGAMESALES.
 WHERE Month='August';
NOTE: The data set WORK.AUGUST has 4 observations and 3 variables.
NOTE: DATA statement used (Total process time):
real time 0.00 seconds
 cpu time 0.00 seconds

If you want to modify the above code to include July and August in the new WORK dataset, you could create two different %LET statements. For example:

%LET Month1 = July;
%LET Month2 = August;

DATA &Month1&Month2;
SET WORK.VideoGameSales;
WHERE Month in ("&Month1" "&Month2");
RUN;

It is also important to mention that the %LET statement has some limitations when used to initially define a macro variable inside of a macro. In this case, the macro variable is defined as local. That is, if the macro variable is referenced outside of the macro in which it is defined, you will receive a warning that the "Apparent symbolic reference <macro-variable-> not resolved". In order to get around this and be able to use the local macro variable throughout the code, the %GLOBAL statement must be used. Below you will see the general syntax of the %GLOBAL statement:

%GLOBAL <macro-variable-1> <...macro-variable-n>;

- Macro-variable-1 ...macro-variable-n is the name of one or more macro variables that generates one or more macro variable names.

The %GLOBAL statement creates one or more global macro variables and must precede one or more %LET statements to make the macro variable(s) global. For example:
%macro MonthlyData;
  %GLOBAL Month1 Month2;
  %LET Month1 = July;
  %LET Month2 = August;
%mend MonthlyData;

%MonthlyData;

DATA &Month1&Month2;
SET WORK.VideoGameSales;
WHERE Month in ("&Month1" "&Month2");
RUN;

CONDITIONAL STATEMENTS AND %SYSFUNC

Additionally, the macro function %SYSFUNC paired with conditional statements can be extremely useful in safeguarding code. Alone, conditional statements such as IF and THEN are very good methods for writing flexible code, but the %SYSFUNC macro function used in conjunction with conditional statements strengthens the flexibility of your code. The %SYSFUNC macro can be used for a variety of purposes, but this section of the paper will focus on how the macro function is used to check the existence of datasets and conditionally execute DATA and PROC steps.

An issue that may come up in routine codes is that the SAS code will continue to process even when a dataset may not exist or has zero observations. This can lead to a variety of problems including unnecessary processing time, unwanted errors and warnings showing up in the log, even though these nonexistent or empty datasets may not be problematic.

In order to check whether a dataset exists, the following syntax must be used within a macro:

%IF %SYSFUNC(EXIST(<dataset name>)) = <rc>; %THEN %DO;
...
%END;

- The possible values of the return code (rc) are 0 and 1. An rc of 0 indicates that the dataset does not exist and an rc of 1 indicates that the dataset does exist.

We could use the following example to conditionally sort a dataset and create a new dataset only if the original dataset exists. If the dataset does not exist, the PROC SORT and DATA steps will not execute.

%macro ExistExample (Dat=);
%IF %SYSFUNC(EXIST(&Dat)) = 1 %THEN %DO;
  PROC SORT DATA = &Dat;
  BY Month;
  RUN;

  DATA WORK.AugustSales;
  SET &Dat;
  WHERE Month = 'August';
  RUN;
%END;
%mend;

%ExistExample(Dat=WORK.VideoGameSales);

To check whether an existing dataset is empty takes a little more finesse with %SYSFUNC. To do this, the dataset must first be opened with the OPEN function. Then, it needs to be evaluated using another function and finally closed with the CLOSE function. To open, evaluate and close a dataset, the following syntax must be used within a macro:
%LET <macro-variable1> = %SYSFUNC(OPEN(<dataset name>));
%LET <macro-variable2> = %SYSFUNC(<function>(<macro-variable1>,<attribute>));
%LET <macro-variable3> = %SYSFUNC(CLOSE(<macro-variable1>));
%IF <macro-variable2> (<gt/ge/eq/lt/le> <rc> %THEN %DO;
...
%END;

- The names of the macro variables can be anything as long as they follow typical macro variable naming conventions
- The rc of macro-variable2 is dependent on the function-attribute pair used
- A useful function for retrieving information about empty datasets is the ATTRN function. It returns the value of the numeric attributes of the dataset. Some of the attributes are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOBS</td>
<td>The number of physical observations in a dataset</td>
</tr>
<tr>
<td>Nlobs</td>
<td>The number of logical observations in a dataset</td>
</tr>
<tr>
<td>NVARS</td>
<td>The number of variables in a dataset</td>
</tr>
<tr>
<td>NDEL</td>
<td>The number of observations marked for deletion in the dataset</td>
</tr>
<tr>
<td>ANY</td>
<td>A value of 1 indicates that the dataset has both observations and variables. A value of 0 indicates that the dataset has no observations. A value of -1 indicates that the dataset had no observations or variables.</td>
</tr>
</tbody>
</table>

We could use the following example to conditionally sort the dataset and create a new dataset only if the original dataset has any observations:

```
%macro ObsExample(Dat=);
%LET OpenDs  = %SYSFUNC(OPEN('&Dat'));
%LET NumObs  = %SYSFUNC(ATTRN(&OpenDs,NOBS));
%LET CloseDs = %SYSFUNC(CLOSE(&OpenDs));
%IF &NumObs gt 0 %THEN %DO;
   PROC SORT DATA = &Dat;
      BY Month;
   RUN;
   DATA WORK.AugustSales;
   SET &Dat;
   WHERE Month = 'August';
   RUN;
%END;
%mend;
%ObsExample(Dat=WORK.VideoGameSales);
```

If the dataset meets the conditions specified above, then the log will indicate that the PROC SORT and the new dataset, WORK.AugustSales, were processed for both examples. However, if the dataset does not meet the specified conditions, then nothing will be produced. Seeing nothing produced in a SAS session can be concerning for new users. A technique that can be used is including a note in the log when the dataset did not exist or was empty. Below is an example for how to put a message in the log when WORK.VideoGameSales has zero observations:
%macro ObsExample(Dat=);
%LET OpenDs = %SYSFUNC(OPEN(&Dat));
%LET NumObs = %SYSFUNC(ATTRN(&OpenDs,NOBS));
%LET CloseDs = %SYSFUNC(CLOSE(&OpenDs));
%IF &NumObs gt 0 %THEN %DO;
   PROC SORT DATA = &Dat;
   BY Month;
   RUN;
   DATA WORK.AugustSales;
   SET &Dat;
   WHERE Month = 'August';
   RUN;
%END;
%ELSE %IF &NumObs eq 0 %THEN %DO;
   %PUT NOTE:&Dat is empty;
%END;
%mend;
%ObsExample(Dat=WORK.VideoGameSales);

CONCLUSION

There are many approaches that one can take to writing flexible code. This paper is only a brief summary of some of the methods a SAS user can employ. In my own experience, I have learned that the tactics discussed in this paper have cut down long-term time and costs for business usage. I am hopeful this paper will be valuable for your own needs.

Additional examples of complete code are included in Appendix A. Please feel free to run these examples in your own SAS session. I would suggest closing SAS between each run in order to get the desired results.

ACKNOWLEDGEMENTS

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are registered trademarks or trademarks of their respective companies.

REFERENCES


Yindra, Chris. 1998. “%SYSFUNC - The Brave New Macro World.” In SUGI 23 Conference Proceedings, Nashville, TN.


ADDITIONAL RESOURCES

The world of global macros is quite large and the scope of this paper only touches on some of the fundamentals. To find out more information on global macro variables, refer to http://support.sas.com/documentation/cdl/en/mcrolref/61885/HTML/default/viewer.htm#a001072159.htm which is a great resource in SAS Support.
For more information on %SYSFUNC functions, please refer to the following SAS Support page: http://support.sas.com/documentation/cdl/en/mcrolref/61885/HTML/default/viewer.htm#a001962663.htm.

This paper only listed a few of the possible attributes used by the ATTRN function. For a complete list of attributes, please check out the following SAS Support page: http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000212040.htm.

CONTACT INFORMATION
The author welcomes comments, suggestions or questions:

Victoria Porterfield
Educational Testing Service
660 Rosedale Road
Mail Stop P-08
Princeton, NJ 08541
Email: vikkielle@gmail.com
APPENDIX A: CODING EXAMPLES

Example 1: Produces a note in the log that indicates the dataset does not exist:

```sas
%macro ExistExample1 (Dat=);
%IF %SYSFUNC(EXIST(&Dat)) = 1 %THEN %DO;
   %PUT Note: &Dat does exist;
   PROC SORT DATA = &Dat;
   BY Month;
   RUN;
%END;
%ELSE %DO;
   %PUT NOTE: &Dat does not exist;
%END;
%mend;
%ExistExample1 (Dat=WORK.VideoGameSales);
```

Example 2: Produces a note in the log that indicates the dataset does exist and sorts the information:

```sas
DATA WORK.VideoGameSales;
INPUT Month $06. AgeGroup $06. Sales;
DATALINES;
June <18  216
June 18-25  153
June 26-35  87
June 36-45  35
June 45<    10
July <18   199
July 18-25  134
July 26-35  69
July 36-45  27
July 45<    6
August <18  181
August 18-25 126
August 26-35 70
August 36-45 13
;
RUN;

%macro ExistExample2 (Dat=);
%IF %SYSFUNC(EXIST(&Dat)) = 1 %THEN %DO;
   %PUT Note: &Dat does exist;
   PROC SORT DATA = &Dat;
   BY Month;
   RUN;
%END;
%ELSE %DO;
   %PUT NOTE: &Dat does not exist;
%END;
%mend;
%ExistExample2 (Dat=WORK.VideoGameSales);
```
Example 3: Produces a note in the log that indicates the dataset has no observations:

```
DATA WORK.VideoGameSales;
INPUT Month $06. AgeGroup $06. Sales;
DATALINES;
June  <18  216  
June  18-25 153 
June  26-35  87 
June  36-45  35  
June  45<   10  
July   <18  199  
July   18-25 134 
July   26-35  69  
July   36-45  27  
July   45<   6  
August <18  181  
August 18-25 126 
August 26-35  70  
August 36-45  13  
;  
RUN;
```

```
DATA WORK.Over45AugustSales;
SET WORK.VideoGameSales;
WHERE Month = 'August' and AgeGroup = '45<';
RUN;
```

```
%macro ObsExample1(Dat=);
%LET OpenDs  = %SYSFUNC(OPEN(&Dat));
%LET NumObs  = %SYSFUNC(ATTRN(&OpenDs,nobs));
%LET CloseDs = %SYSFUNC(CLOSE(&OpenDs));
%IF &NumObs gt 0 %THEN %DO;
   %PUT NOTE: &Dat has &NumObs observations;
   PROC SORT DATA = &Dat;
   BY Month;
   RUN;
%END;
%ELSE %DO;
   %PUT NOTE: &Dat has no observations;
%END;
%mend;
%ObsExample1(Dat=Work.Over45AugustSales);
```
Example 4: Produces a note in the log that indicates the dataset has observations and sorts the information:

```sas
DATA WORK.VideoGameSales;
INPUT Month $06. AgeGroup $06. Sales;
DATALINES;
June <18 216
June 18-25 153
June 26-35 87
June 36-45 35
June 45< 10
July <18 199
July 18-25 134
July 26-35 69
July 36-45 27
July 45< 6
August <18 181
August 18-25 126
August 26-35 70
August 36-45 13
;
RUN;

DATA WORK.AugustSales;
SET WORK.VideoGameSales;
WHERE Month = 'August';
RUN;

%macro ObsExample2(Dat=);
%LET OpenDs  = %SYSFUNC(OPEN(&Dat));
%LET NumObs  = %SYSFUNC(ATTRN(&OpenDs,nobs));
%LET CloseDs = %SYSFUNC(CLOSE(&OpenDs));
%IF &NumObs gt 0 %THEN %DO;
   %PUT NOTE: &Dat has &NumObs observations;
   PROC SORT DATA = &Dat;
   BY Month;
   RUN;
%END;
%ELSE %DO;
   %PUT NOTE: &Dat has no observations;
%END;
%mend;
%ObsExample2(Dat=Work.AugustSales);
```