Paper 351-2013

The Surprisingly "Sym"ple Alternative to Hardcoding

Ruchi G. Sharma and Rachel E. Carlson, Mayo Clinic

INTRODUCTION

As a frequent SAS® user, do you often feel that you are spending too much time looking up procedure results or hardcoding the values into programs? Does your data often change causing a need to rerun analyses, forcing you to repeat steps? Save time rerunning analysis programs by reducing the amount of hardcoded variables and formats. Our paper will demonstrate how to effectively use the CALL SYMPUTX routine and the SYMGET function to make your code more flexible and minimize the possibility of data calculation errors.

CHANGE IS GOOD, KEEPING UP WITH IT IS EVEN BETTER!

In the health care industry there often is a need to summarize data from a database or clinical trial. In some cases, the data is updated on a continuous basis. Often, problems arise when you are required to run updated summary reports and the task involves more than pressing a button.

Age is a common variable that is collected in the healthcare setting and is often used as a categorical variable for a predictor in statistical modeling. We can create a categorical variable in which the classes depend on summary values such as the minimum, maximum or median of the continuous variable of interest. In such cases common practice is to first calculate these values and then hardcode them to create this categorical variable. This practice can become tedious and error prone, especially when you can have frequent data updates. SAS® provides us datastep tools that can be used to automate and simplify these processes. In the end, you will save time and reduce the possibility of errors.

Using The CALL SYMPUT Routine

The CALL SYMPUT routine allows the user to create a global macro variable in a datastep. The routine contains two arguments, the first is the name of the macro variable you want to create and the second argument is the value that you want to load into the macro variable. We have included 4 leading and trailing blanks in the example below.

Since macro variables store character values, the result of CALL SYMPUT routine contains any leading and trailing blanks in the value of the variable. These can be removed using the LEFT and TRIM functions.

```
data _null_;
 length event $15;
 event='    SGF    ';
 CALL SYMPUT ('where', event);
 CALL SYMPUT ('where2', left(trim(event)));
 run;

%put I enjoy &where
I enjoy     SGF
%put I enjoy &where2;
I enjoy SGF
```

Alternatively, we can use the CALL SYMPUTX routine which automatically removes the leading and trailing blanks.

```
data _null_;
 length event $15;
 event='    SGF    ';
 CALL SYMPUTX ('where3', event);
run;

%put I enjoy &where3;
```

I enjoy SGF

From this point forward, we will be using the CALL SYMPUTX routine instead of CALL SYMPUT without loss of generality.

Using the SYMGET function

Once we create the macro variable using CALL SYMPUTX it is necessary to retrieve the value within a subsequent datastep in order to create a categorical variable. The SYMGET function retrieves values from both global and local macro variables into a current datastep. Here is an example using the SYMGET function:

```
data _null_;
  aa=symget('where3');
 put aa=;
run;
```

aa=SGF

Combining the use of CALL SYMPUTX and SYMGET gives us the perfect tool to seamlessly pass values from one datastep to another.

DO YOU FEEL LIKE A PROGRAMMING ROBOT? CALL SYMPUTX AND SYMGET TO THE RESCUE!

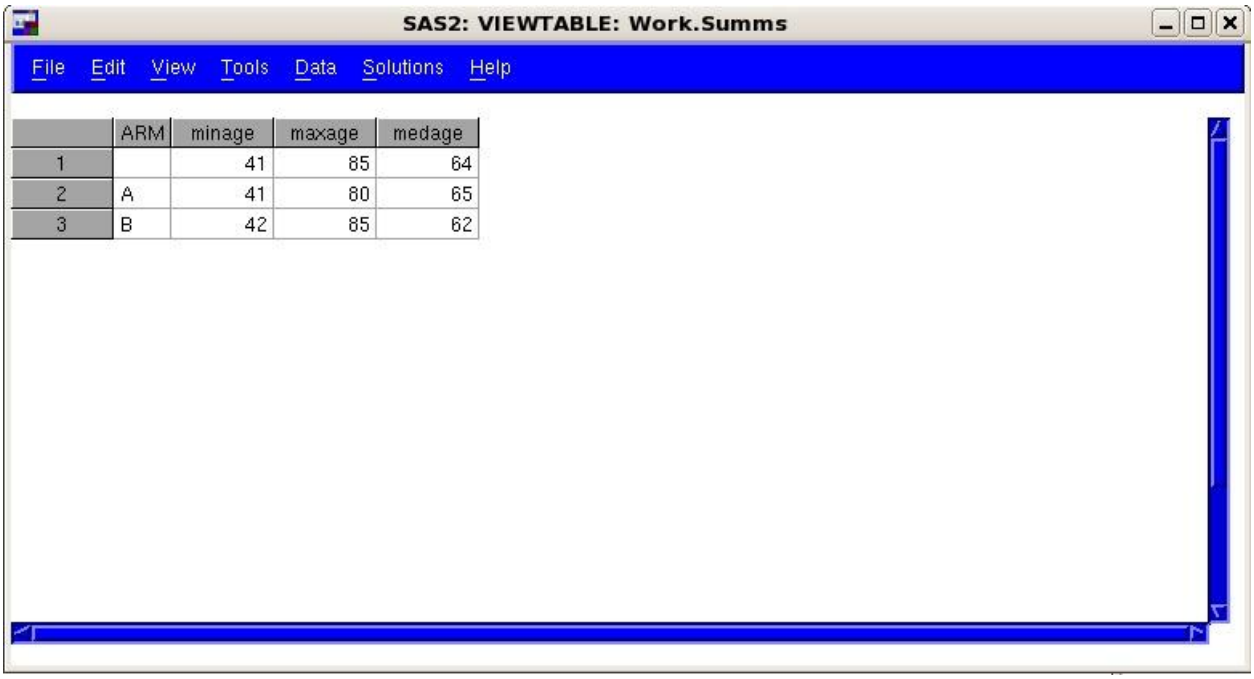
As previously mentioned, it is often common in many industries to summarize data and to use the summary values in reports and/or analyses. When your data changes frequently there is a need to create updated reports with the latest summary values. As you change the hardcoded values to accomplish these tasks, you might think "A robot can do this!" Hardcoding all these values into the program is time consuming, as well as more susceptible to human error, not to mention more difficult when handing off to someone else. The CALL SYMPUTX routine, along with the SYMGET function offers a quick way to keep you from turning into your child's robotics science project!

We will use a dataset containing age of participants for a cancer treatment trial with two treatment arms. The objective is to create a two level categorical variable for age split by the median. We do this for each treatment arm and overall. The variables will be numeric with the appropriate formats. These will need to be defined as well.

The sample dataset *demotable* contains two treatment arms, A and B, plus an ID variable called extrefid and age. We use PROC MEANS to find the minimum, maximum and median age by each treatment arm and overall. This is output into a dataset called *summs*. The colon modifier in the drop dataset option will drop all variables that begin with an underscore. In our example, these variables are not needed.

```
proc means data=sgf2013.demotable;
    var age;
    class arm;
    output out=summs (drop=_:)
           min (age)=minage
           max (age)=maxage
           median (age)=medage;
run;
```

Figure 1 below provides a view of the output dataset from PROC MEANS.



	ARM	minage	maxage	medage
1		41	85	64
2	A	41	80	65
3	B	42	85	62

Figure 1

We use CALL SYMPUTX in a datastep for each variable that we want to use in our report. We need 9 macro variables, 3 each for arms A, B and O, namely minimum, maximum and median.

```
data _null_;
 set summs;
 if arm='' then arm='O';
 CALL SYMPUTX("minage"||arm, minage);
 CALL SYMPUTX("maxage"||arm, maxage);
 CALL SYMPUTX("medage"||arm, medage);
run;
%put _global_;
```

Figure 2 shows the log output of the global macro variables we've created.

Figure 2

That was easy, wasn't it?! Summary values are now stored in macro variables which can be used to replace the hardcoded values. As a side note, you may notice that the overall median is displayed as a value of 64 in the summs dataset, but actual value is 63.5 (shown in the %put _global_). This is because the output dataset *summs* inherits the original formatting for the age variable.

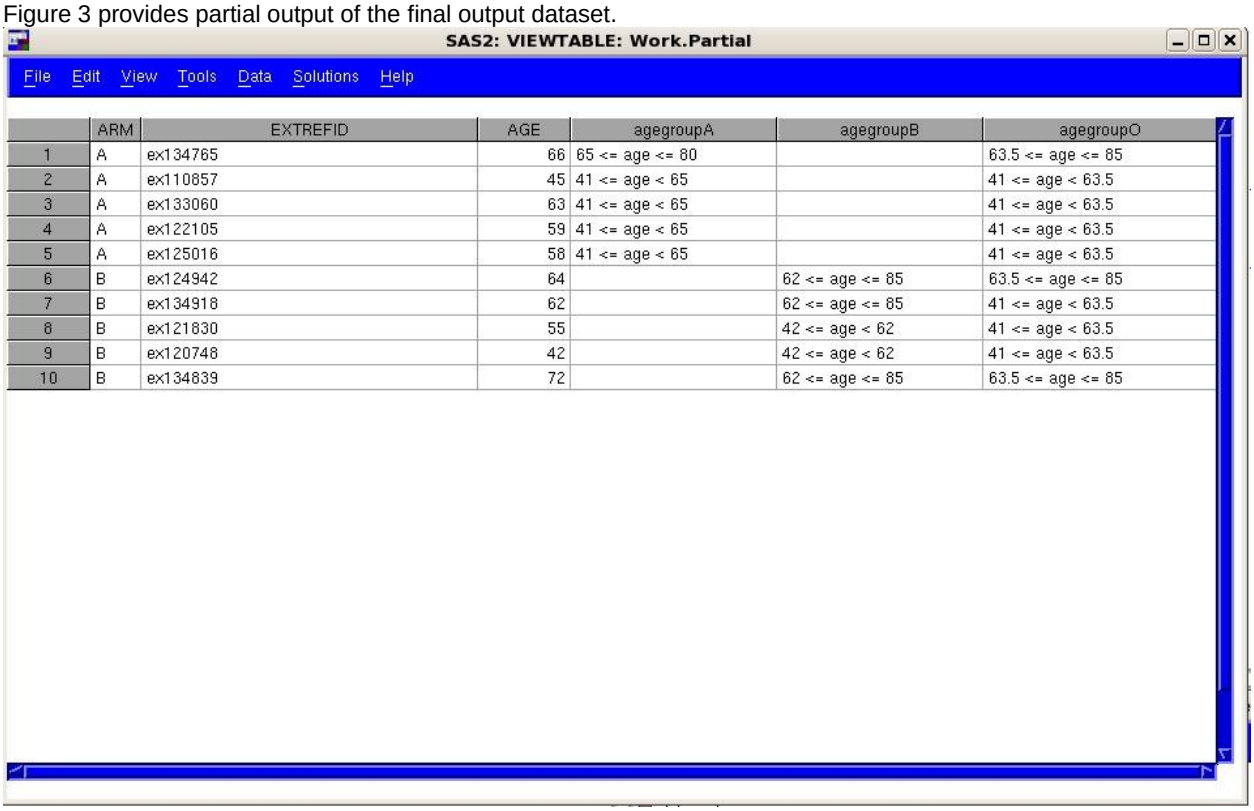
GETTING YOUR ROBOT TO CATEGORIZE IS AS "SYM"PLE AS 1, 2, 3

Next we will utilize these macro variables to create 3 new categorical variables using the SYMGET function. In our example, we will use SYMGET to create the following categorical groups, agegroupA, agegroupB and agegroupO. They will be formatted to display the range of values in which each age measurement lies.

```
proc format;
value agefmtA  0="&minageA <= age < &medageA"
               1="&medageA <= age <= &maxageA";

value agefmtB  0="&minageB <= age < &medageB"
               1="&medageB <= age <= &maxageB";

value agefmtO  0="&minageO <= age < &medageO"
               1="&medageO <= age <= &maxageO";

run;


data final;
 set sgf2013.demotable;
   if arm='A' then do;
   if symget("minageA") <= age < symget("medageA") then agegroupA=0;
   if symget("medageA") <= age <= symget("maxageA") then agegroupA=1;
  end;
  if arm='B' then do;
   if symget("minageB") <= age < symget("medageB") then agegroupB=0;
   if symget("medageB") <= age <= symget("maxageB") then agegroupB=1;
  end;
     if symget("minageO") <= age < symget("medageO") then agegroupO=0;
   if symget("medageO") <= age <= symget("maxageO") then agegroupO=1;
  format agegroupA agefmtA. agegroupB agefmtB. agegroupO agefmtO.;
```

```
run;
```

Figure 3 provides partial output of the final output dataset.



	ARM	EXTREFID	AGE	agegroupA	agegroupB	agegroupO
1	A	ex134765	66	65 <= age <= 80		63.5 <= age <= 85
2	A	ex110857	45	41 <= age < 65		41 <= age < 63.5
3	A	ex133060	63	41 <= age < 65		41 <= age < 63.5
4	A	ex122105	59	41 <= age < 65		41 <= age < 63.5
5	A	ex125016	58	41 <= age < 65		41 <= age < 63.5
6	B	ex124942	64		62 <= age <= 85	63.5 <= age <= 85
7	B	ex134918	62		62 <= age <= 85	41 <= age < 63.5
8	B	ex121830	55		42 <= age < 62	41 <= age < 63.5
9	B	ex120748	42		42 <= age < 62	41 <= age < 63.5
10	B	ex134839	72		62 <= age <= 85	63.5 <= age <= 85

Figure 3

CONCLUSION: UNPLUG YOUR INNER ROBOT

The CALL SYMPUTX routine helps to pass dataset variables into global macro variables which can then be used in subsequent data steps (using the SYMGET function), in procedure calls, as well as in macro processing. It is one of the many easy-to-use SAS® tools that help to improve efficiency in coding while guaranteeing accuracy of results. Who's the robot now?!!

REFERENCES

The topic for this presentation was predominantly motivated by the everyday work we do at the Mayo Clinic. The use of summary variables from frequently changing data is common place and this methodology is widely used by us. As we become more comfortable as SAS® users, the flexibility afforded by tools such as the CALL SYMPUTX routine becomes desirable.

SAS® online documentation, available at http://support.sas.com/documentation/

ACKNOWLEDGMENTS

Special thanks to Kari Anderson, Patrick Fitz-Gibbon, Alice Wang, Leigh Gomez-Dahl and Angelina Tan for their assistance, encouragement and support.

APPENDIX

A. OTHER USEFUL CODE

As an alternative to the SYMGET function, you can use double quotes around the macro variable name, which will retrieve the result that is stored in the macro variable. For example,

```
data _null_;
ab="&medageA";
put ab;
run;
65
```

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please feel free to contact the authors at:


Rachel Carlson
Mayo Clinic
200 First Street SW
Rochester, MN 55905
(507) 284-4645
Carlson.Rachel@Mayo.edu


Ruchi Sharma
Mayo Clinic
200 First Street SW
Rochester, MN 55905
(507) 293-0629
Sharma.Ruchi@Mayo.edu