Let SAS® Write and Execute Your Data-Driven SAS Code
Kathy Hardis Fraeman, United BioSource Corporation, Bethesda, MD

ABSTRACT
Some SAS® programming statements are based on specific values of data, such as the conditions given in IF… THEN…; ELSE IF…THEN…; blocks. If the values of the data are static and the number of data values is small, a SAS programmer can simply hard code the possible data values directly in the program. However, in instances where hundreds of data values need to appear in “IF… THEN…; ELSE IF…THEN…;” blocks, or if specific data values can change with different executions of the same program, then a SAS program can generate data-driven SAS code and immediately execute that code as part of the program, as long as the data values are in a SAS data set. This technique involves using DATA _NULL_ to generate the SAS code and then a %INCLUDE statement to immediately run the code SAS generated with DATA _NULL_. The programming example of this technique shows how SAS can automatically generate the programming code needed to turn a character variable with hundreds of different text values into a formatted numeric variable, where the labels of the attached format are the same as the original text. All of the programming in this example is done without ever having to write out any of the character variable’s text values, or even having to know the number of unique text values. This general programming technique can be used in many other applications where SAS programming is dependent on actual values of data.

INTRODUCTION
A lot of SAS programming is “data-driven.” “Data-driven” means that actual values of data need to be included in the SAS code for the programming to work. A simple example of data-driven programming is the conversion of a text value of gender into a formatted numeric variable, as shown in the following example:

```
proc format;
  value genfmt 1 = "Female"
                 0 = "Male";
run;

data reformat_patient_vars;
  set original_patient_vars;
  format gender_cat genfmt.;
  label gender_cat = "Gender Category";
  if gender = "Male" then gender_cat = 0;
  else if gender = "Female" then gender_cat = 1;
run;
```

The programmer needs to know that the actual data values for the variable GENDER in the data are “Male” and “Female” for the program to work, and these actual data values need to be hard coded in the program.

In many instances, the number of actual data values of a variable is not very large, and these values are “static”, meaning they don’t change with different versions of the data. Data driven programming is easy in these many instances.

DATA DRIVEN PROGRAMMING BECOMES DIFFICULT
However, a SAS programmer can run into problems if the variable need for data-driven programming has hundreds of values that need to be hard coded in the SAS program. These problems will be compounded if the hundreds of data values can change each time the program is run, due to factors such as different input data and/or changes in programming rules. Writing the SAS programming that includes the hundreds of constantly changing data values is practically impossible!
I was faced with this situation when using Electronic Medical Records (EMR) data to try to identify complex chemotherapy regimens given to cancer patients. We had each patient's individual drug administration data for the hundreds of possible individual chemotherapy drugs and the dates these drugs were administered. However, because chemotherapy regimens consist of combinations of different drugs administered concurrently and in patterns, we needed to define rules for determining what individual drugs were included in the multi-drug chemotherapy regimens. These hundreds of individual chemotherapy drugs, when combined in different combinations, could produce even more hundreds of types of combination regimens. Further complicating the programming was the need to use different programming rules to define what constituted a regimen. These different drug combinations changed every time the regimen programming rules changed.

Each patient in the study needed to have a single defined “initial chemotherapy regimen.” All of my standard report macros require a formatted numeric variable as input, and I obviously couldn’t have hundreds of lines of

```
if text_regimen = "Cyclophosphamide/Doxorubicin" then reg_cat = 1;
else if text_regimen = "Fludarabine/Mitoxantrone/Rituximab" then reg_cat = 2;
else if ...
else if text_regimen = "Doxorubicin/Vincristine" then reg_cat = 237;
```

especially if these hundreds of different text values of chemotherapy regimen could change, either based on our input data or programming rules used to define a regimen.

**IF THESE DATA VALUES ARE IN A SAS DATA SET, HAVE SAS WRITE THE CODE**

Fortunately these hundreds of unique derived regimen names are found in a text variable `REGIMEN_TEXT_NAME` in a SAS data set. This SAS data set was created by taking all of the unique values of `TEXT_REGIMEN` in the patient-level data and then renaming the variable to `REGIMEN_TEXT_NAME`.

This SAS data set also has a variable with the sequential number `REG_CAT_NO` that was to be the value of the variable `REG_CAT` as in the example above, reflecting the sort order (not necessarily alphabetic) that the regimen names are to appear in the report.

Because these data values are in a SAS data set, they don’t need to be written out by the programmer in hundreds of “IF... THEN...; ELSE IF...THEN...;” statements. SAS can write out these hundreds statements and then execute them! If the actual values of the regimen name change for different runs of the program, SAS will always use the current values of the regimen names found in the patient-level data. Finally, as an added bonus, SAS can even automatically generate the format for the numeric categorical variable. And all of this can be done without ever needing to know or hard code any of the regimen names.

**WRITING SAS CODE WITH SAS**

The name of the SAS data set with the regimen names is `REGIMENS`, and the variables in the data set are `REGIMENTextoTEXT_NAME` (text name of regimen) and `REG_CAT_NO` sequential regimen number, starting with 1.

The SAS code to create a SAS program with the hundreds of “IF... THEN...; ELSE IF...THEN...;” statements is:

```
/* Create SAS statements to be %included in the program that create the new */
/* categorical SAS variable to correspond with the chemotherapy regimen name variable */
/*****************************/
data _null_;  
set REGIMENS;  
file "PATHNAME\regimens.sas";  
if REG_CAT_NO = 1 then put @5 "if REGIMEN_TEXT = '' REGIMEN_TEXT_NAME
" then REG_CAT = " REG_CAT_NO ";
else put @5 "else if REGIMEN_TEXT = '' REGIMEN_TEXT_NAME
" then REG_CAT = " REG_CAT_NO ";
run;
```
The output data set named REGIMENS.SAS generated by the above code looks like:

```sas
if regimen_text = 'Cyclophosphamide/Doxorubicin' then reg_cat = 1;
else if regimen_text = 'Fludarabine/Mitoxantrone/Rituximab' then reg_cat = 2;
else if ...
else if regimen_text = 'Doxorubicin/Vincristine' then reg_cat = 237;
```

**RUNNING THE DATA-DRIVEN SAS CODE WITH SAS**

To then run the above code generated by SAS with SAS, the following code can be used:

```sas
/* ***********************/
/* * Dynamically create a SAS formats for the regimen name */
/* ***********************/
data regimens_fmt;
  set regimens;
  fmtname = "regfmt";
  start = reg_cat;
  label = regimen_text;
  run;

proc format cntlin = regimens_fmt;
  run;

/* ***********************/
/* * Create the patient data set with the reformatted variables */
/* ***********************/
data reformat_patient_vars;
  set original_patient_vars;
  format gender_cat genfmt.;
  label gender_cat = "Gender Category";
  if gender = "Male" then gender_cat = 0;
  else if gender = "Female" then gender_cat = 1;

  format reg_cat regfmt.;
  label reg_cat = "Regimen Category";
  %inc "PATHNAME\regimens.sas" / source2;
  run;
```

The “source2” option in the “%include” statement causes the included SAS statements to appear in the SAS log.
SINGLE QUOTES, DOUBLE QUOTES AND MACRO CODE

If using this technique with any form of SAS macro code, then the use of single quotes and double quotes matters.

If a macro variable needs to be evaluated when creating the output file, use double quotes as follows:

```
%let pathname = c:\myfile;

data _null_;  
  set regimens;  
  file "&pathname\regimens.sas";  
  if REG_CAT_NO = 1 then put @5 "if REGIMEN_TEXT = '" REGIMEN_TEXT_NAME 
  "' then REG_CAT = " REG_CAT_NO ";";  
  else put @5 "else if REGIMEN_TEXT = '" REGIMEN_TEXT_NAME 
  "' then REG_CAT = " REG_CAT_NO ";";  
run;
```

If this technique is being used to create macro code and the macro code just needs to be output as text, use single quotes as follows:

```
%let pathname = c:\myfile;

data _null_;  
  set macro_call_data;  
  file "&pathname\runmacro.sas";  
  put @5 '%mymacro(grpvar = &grpvar, colnum = &colnum, 
  ';  
  put @15 'rownum = ' combo_num ',
  ';  
  put @15 'rowvar = ' flag_var_name ',
  ';  
  put @15 'label = ' flag_var_label ');';  
run;

%inc "&pathname\runmacro.sas" / source2;
```

CONCLUSION

This general programming technique can be used in many other applications where SAS programming is dependent on actual values of data. This general technique involves using DATA _NULL_ to generate the SAS code into an output SAS program, and then a %INCLUDE statement to immediately run the code SAS generated with the DATA _NULL_ statement. This technique can be used with many applications, and the level of complexity of the dynamically generated SAS code can be much more complex than the “IF... THEN...; ELSE IF...THEN...;” block of SAS code.
ACKNOWLEDGMENTS
SAS and other SAS Institute, Inc. products 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.

CONTACT INFORMATION
Please contact the author with any comments or questions:
Kathy H. Fraeman
United BioSource Corporation
7101 Wisconsin Avenue, Suite 600
Bethesda, MD 20832
(240) 235-2525 voice
(301) 654-9864 fax
kathy.fraeman@unitedbiosource.com