PharmaSUG 2018 - Paper SI10

Techniques for Global, Adaptable, and Efficient Programming

Sunil Kumar Pusarla, Omeros Corporation
Paul Hamilton, Omeros Corporation

ABSTRACT

SAS programmers in the pharmaceutical industry have to perform many common tasks across different study programs. Re-usability of the code we produce goes a long way to make our lives easier. Many situations arise when we need to modify our own code. In such times, rather than inspecting each line of a program for statements which need modification, we can constrain our attention to one easily identified location, which affects all subsequent code automatically. There are many simple yet effective techniques that will increase coding productivity and produce highly robust programs. This paper addresses some of those techniques: legacy procedures like FORMAT; new enhancements to procedures like PROC SUMMARY; obscure functions and options; as well as more recent resources such as PROC FCMP.

INTRODUCTION

In our day-to-day programming, we often modify or update our codes due to the dynamic nature of the data or a change in the specifications. Sometimes we may forget to make changes at a later time, or overlook places where the change is required. So, it is preferable to have flexible and robust code to help reduce or prevent many common mistakes. In some other situations, it is not possible to create a final version due to changeability of rules (ex: change of decimal places etc.). While under development with dummy randomization codes some programs run very well and produce the expected results, but when executed in production with real codes errors or unexpected results often occur. Like the saying ‘many ways to skin a cat’ (pardon me animal lovers) there are many ways to reduce the above problems. This paper will address some of those ways. This paper will not explain the details of any procedure, option, or function used, but rather *how* they can be utilized to achieve our goals.

OVERVIEW

PROC FORMAT reminds me of the saying, “Old is Gold”. It is an old and simple, yet extremely useful procedure. In clinical SAS programming, we often need to modify our own code (whether for formatting issues, lack of foresight, etc.). We can easily handle some of these issues and increase the efficiency of our programs by using formats in different ways. Many tend to use formats only for date/time variables but by extending their usage in other places we can make our lives better. Here are some elegant uses:

EXAMPLE 1

PROC FORMAT reduces our code space by cutting many ‘IF-THEN-ELSE’ or ‘SELECT-WHEN’ statements. It enhances the readability of the code and simplifies changing the code, if needed.

Without using the FORMAT	Using FORMAT
`if (race eq 'American Indian or`	`proc format;`
`Alaska Native')       then order =1;`	`value $race`
`else if (race eq 'Asian')`	`'American Indian or Alaska Native'`
`                      then order =2;`	`                                = 1`
`else if (race eq 'Black or African`	`'Asian'                         = 2`
`American')            then order =3;`	`'Black or African American'     = 3`
`else if (race eq 'Native Hawaiian or`	`'Native Hawaiian or Other Pacific`
`Other Pacific Islander')`	`Islander'                       = 4`
`                      then order =4;`	`'White'                         = 5`
`else if (race eq 'White')`	`'Other'                         = 6;`
`                      then order =5;`	`run;`
`else if (race eq 'Other')`	
`                      then order =6;`	`order = input(put(race,$race.),best.);`

Table 1

EXAMPLE 2

Some statisticians prefer to look at the summary statistics separately. If our table shells are specified this way, there is no need to create many character variables before transposing in order to achieve the required decimal places. With proc format, along with the PUTN function we can easily follow the decimal specification. This is a good and fast approach when the rules for decimal places change or when we need to create multiple versions of tables with different decimals for different end users. This way changing the format values in one place suffices rather than checking our whole program and reviewing output to see where to change our code.

Shell	Without using format	Using format
n xx	`cn    = strip(put(n,6.));`	`proc format;`
mean xx.xx	`cmean = strip(put(mean,6.2));`	`  value $stfmt`
std xx.xxx	`cstd  = strip(put(std,6.3));`	`    'n'      = 6`
median xx.xx	`cmed  = strip(put(med,6.2));`	`    'mean'   = 6.2`
min xx.x	`cmin  = strip(put(min,6.1));`	`    'std'    = 6.3`
max xx.x	`cmax  = strip(put(max,6.1));`	`    'med'    = 6.2`
		`    'min'    = 6.1`
		`    'max'    = 6.1;`
		`run;`
		`trt =`
		`putn(var,put(_name_,$stfmt.));`
		`(_name_ contains transposed`
		`variables)`

Table 2

EXAMPLE 3

At times creating formats in the traditional way is time consuming and looks very lengthy. But, we can make it easy to create a format with the CNTLIN option. This reduces a big chunk of code into a small number of data steps.

Creating the FORMAT in traditional way	Creating the FORMAT using the CNTLIN option
`proc format;`	`data test;`
`value avisit`	`  format stime 8.2;`
`0.0 - 1.5  = '1 minute from start'`	`  do stime = 1.5 to 50.5 by 1;`
`1.5 - 2.5  = '2 minutes from start'`	`  output;`
`2.5 -  3.5 = '3 minutes from start'`	`  end;`

```
3.5 -  4.5 = '4 minutes from start'
4.5 -  5.5 = '5 minutes from start'
5.5 -  6.5 = '6 minutes from start'
6.5 -  7.5 = '7 minutes from start'
7.5 -  8.5 = '8 minutes from start'
 ………
49.5 - 50.5 = '50 minutes from
start';
run;
```

```
run;

data test1 ;
  set test;
  length label $40;
  etime = lag(stime);
  if (etime eq .) then etime = 0;
  new+1;
  if (new eq 1) then label = '1
minute from start');
  else label = catx(' ',new,'minutes
from start');
  fmtname = 'AVISIT';
  start = etime;
  end   = stime;
  drop new ;
run;

proc format cntlin = test1 fmtlib;
  select AVISIT;
run;
```

Table 3

EXAMPLE 4

I would like to acknowledge Rick Langston for his help in enabling me to use PICTURE FORMAT. Even though the utility of the VALUE statement was obvious, the usefulness of PICTURE FORMATs in statistical programing was not. Using the PICTURE FORMAT is very useful when dealing with some date/time variables. Even though we have numerous date/time formats provided by SAS Institute, they don't always serve our needs (for instance, one cannot use a width of 16 with ISO E/B formats, especially when seconds were not collected). Using the PICTURE FORMAT while creating the date/time variables in the SDTM datasets, reduces the need for many character functions like SUBSTR etc.

Using ISO formats

```
rfstdtc = put(rfstdt, is8601dt19.);
if substr(rfstdtc,17)=":00"
then
rfstdtc=left(substr(rfstdtc,1,16));
```

Using picture format

```
proc format;
  picture sdtmdt (default=16)
    other = '%Y-%0m-%0dT%0H:%0M'
(datatype=datetime);
run;

rfstdtc = put(rfstdt, sdtmdt.);
```

Table 4

EXAMPLE 5

The Frequency procedure is very useful to produce counts, but it only counts data values present in the data. Often our specifications (CRF, Table Shells) pre-specify a list of desired values. It is easy to create a format and use it to produce all desired values, whether present in the data or not. This is accomplished by using options COMPLETETYPES and PRELOADFMT in PROC MEANS/SUMMARY. The advantage of this technique is, there is no need to worry about whether all desired data values occur as the correct outcome is guaranteed.

Without using the FORMAT	**Using FORMAT**

`(Assume there are no subjects of race = 'American Indian' and 'Native Hawaiian' in your study at a specific time point).`
 	
`data dummy;`
`  length race $40;`
`  race = 'American Indian'  ;`
`  …`
`  output;`
`  race = 'Native Hawaiian';`
`  …`
`  output;`
`run;`
	`proc format;`
	`value $race`
	`'American Indian' = 'American Indian'`
	`'Asian'           = 'Asian'`
	`'Black or African American'`
	`                  = 'Black or African American'`
	`'Native Hawaiian' = 'Native Hawaiian'`
	`'White'           = 'White' ;`
	`run;`
	
	`proc means data= adsl COMPLETETYPES NWAY;`
	`  class trt01p race / PRELOADFMT;`
	`  var dummy ;`
	`  format race $race.;`
	`  output out = race`
	`           n = an ;`
	`run;`

Table 5

The examples above solve commonly encountered problems in clinical programming. There are other more obscure issues that have solutions as well, e.g., multi-label formats for non-discrete category tables; use of the CNTLOUT statement to extend existing formats programmatically and producing code lists from metadata using CNTLOUT.

During initial program development, situations often occur that cannot be resolved until additional specifications are defined. However, it is difficult to remember which code to revisit and resolve. One way to resolve this is to use environmental variables or warning/error messages. In these situations, a log parser greatly facilitates detection of such self-feedback.

Example 1	**Example 2**
Search current folder path, if in production issue WARNING in SASLOG to examine critical code	During development no duplicates were observed; however as data changes the code will find and report them

`if (index(&pgmpath), 'dev')) then do;`
` ………;`
`end;`
`else put 'WAR' 'NING';`
	`if (col1 ne col2) then … ;`
	`else put 'ER' 'ROR';`

Table 6

USER DEFINED FUNCTIONS

SAS software comes with a huge number of functions; however, using them may not always produce a direct solution. SAS Institute now (9.2 onwards) provides users the flexibility to create their own functions to establish a straightforward solution. Initial impressions of PROC FCMP and user defined functions seemed of limited utility, especially in a world with many production macros. However, combinations of simple formats or standard functions and PROC FCMP have produced very elegant and compact solutions to everyday activities. Our user-defined functions are very easy to use and the code complexity is greatly reduced. Below are some examples.

Using PROC FCMP and our good friend PROC FORMAT, we can create flexible functions.

Syntax:

```
libname ufun 'C:\...\FCMP';
```

/* One program creates NPCT function to produce the results like xx (xx.x%) */

```
 proc fcmp outlib = ufun.func.report;
   function npct(x,y) $40;
     length want x1 y1 $100;
     if (n(x) eq 1) then x1 = strip(put(x,best.));
     else                x1 = '0';
     if (n(x,y) eq 2 and y ne 0) then pct = (x/y)*100;
     y1 = cats(putn(pct, put('Percent', $statfmt.)));
     if (n(x) ne 1) or (x = 0) or (y = 0) then  y1 = ' ';
     if (substr(y1,1,3) eq '100') then y1 = '100';
     want = cats(x1,'(',y1,'%)');
     if (y1 eq ' ') then want = '0';
     want = tranwrd(want,'(',' (');
     return(want);
   endsub;
run;
```

/* Inside calling program (for table)*/

```
proc format;
  value $statfmt
    'N'       = 6.
    'Percent' = 6.1;
run;
```

Without using the user defined function	**Using user defined function**

`if n(placebo) eq 1 then col1 = strip(put(placebo,4.)) || ' (' ||strip(put((placebo /&col1)*100,6.1)) || '%)';`
`else col1 = '0';`
`if (col1 eq '100.0%') then col1 = '100%';`
	`Col1 = npct(placebo,&col1);`
	⇓
	`43 (100%)`
	`27 (62.8%)`
	` 0`

Table 7

The built in PROPCASE function is very useful, but often produces undesirable results. However, combining it with TRANWRD makes it a perfect solution.

/* Title case function */

```
proc fcmp outlib = ufun.func.report;
  function tcase(string $) $;
     length changed $32767;
     changed = propcase(string);
     changed = tranwrd(changed, ' In ', ' in ');
     changed = tranwrd(changed, ' And ', ' and ');
     changed = tranwrd(changed, ' Or ', ' or ');
     changed = tranwrd(changed, ' By ', ' by ');
     /* add as many as you need */
     want = tranwrd(changed, ' Hiv ', ' HIV ');
     return(want);
   endsub;
run;
```

Without using the user defined function	**Using user defined function**

`col1 = propcase(col1);`
`col1 = tranwrd(col1, ' Of ', ' of ');`
`col1 = tranwrd(col1, ' And ', ' and ');`
`col1 = tranwrd(col1, ' Or ', ' or ');`
`etc.`
	`col1 = tcase(col1);`

Table 8

SOME OPTIONS THAT SIMPLIFY CODING

SELECTING THE APPROPRIATE P-VALUE

WARN = OUTPUT

Often, clinical programmers need to capture Pearson Chi-square p-value or Fisher's exact p-value based on the expected cell frequencies for different levels of data. For example, we need to perform these tests for different visits or ages or any other 'by' variable. For people without a background in statistics it may be difficult to remember to calculate expected cell frequencies or its formula. We can rely on a SAS produced warning message in the output window (when the expected cell frequency is less than 5), but if there are many levels it is very difficult to check for warnings in the output window. Automating this process through use of a macro eliminates this difficulty. This can be achieved in two ways. One way is to create a macro to calculate expected cell frequencies and based on these numbers choose to calculate either a Pearson Chi-square or Fisher's exact test. Another way is to capture the SAS provided warning message and utilize it to calculate the correct p-value.

SAS developers made our lives easier with the newly (SAS 9.2 onwards) provided option, WARN = OUTPUT, along with the CHISQ option in the TABLES statement. Using this option gives us a new variable WARN_PCHI (label = Chi-Square Warning) with values of 0 and 1 (Table 9). If this value is 0, then consider the Pearson Chi-square p-value; if it is 1, then consider the Fisher's exact p-value.

Obs	Origin	WARN_PCHI (Chi-Square warning)	P_PCHI (P-value for Chi-Square)	XP_PCHI (Exact P-value for Chi-Square)	XP2_FISH (Fisher's Exact Test P-value (2-Tail))
1	Asia	**0**	**2.942908E-13**	**8.8589E-15**	**8.8589E-15**
2	Europe	**1**	**0.0005168136**	**.007600266**	**.007600266**
3	USA	**1**	**5.764249E-15**	**3.2587E-13**	**3.2587E-13**

Table 9

SORTING CHARACTER INTEGERS

SORTSEQ = UCA (NUMERIC_COLLATION=ON)

This option allows integers expressed as text in a character string, to be ordered numerically. We can use this option to sort values such as "1", "2", "3", etc. numerically *without* creating a corresponding numeric variable or using a 'Z.' format. For example, in SDTM datasets generation we need to sort by a sponsor defined identifier variable such as MHSPID ($) in domain MH (Medical History).

```
proc sort data = mh SORTSEQ = UCA (NUMERIC_COLLATION=ON);
  by usubjid mhspid;
run;
```

MERGE/JOIN ALTERNATE

There are two main ways to merge two different datasets in order to get required variables into one dataset. One way is the data step merge, which requires sorting the datasets before merging. The other way is PROC SQL, which does not require prior sorting. Both methods have advantages and disadvantages and programmers choose one method over the other based on their preferences and the data.

A third way exists as well, which can be very handy for selecting a few variables from another data set. This method requires no sorting of datasets, but does require multiple passes. One can argue that this technique is inefficient for very large datasets, or takes up more system resources than either of the other two methods. Consuming system resources is not a concern for today's computers with huge memory capacity and multiple processors, for most clinical trial domains. One example is to acquire study reference start date into the current dataset, in order to calculate the study day.

Syntax:

```
data xf1;
  set xf;
  /* Get study reference start date */
  length string1 $200;
  string1 = catx('',
                 '(where = (usubjid =',
                 quote(strip(usubjid)),
                 ')');
  dsid = open('sdtm.dm' || string1);
  vnum = varnum(dsid,'RFSTDTC');
  rc = fetch(dsid);
  rfstdtc = getvarc(dsid,vnum);
  rc = close(dsid);
  if length(rfstdtc) ge 10 then surdt = input(rfstdtc,yymmdd10.);
  if n(xfdt, surdt) eq 2 then xfdy = xfdt - surdt;
  if (xfdy ge 0) then xfdy = xfdy + 1;
run;
```

CONCLUSION

Using PROC FORMAT in different ways enhances the efficiency of SAS programs. Building our own functions makes it easy to verify and modify our code, and easily accommodates changing specifications. We can reduce the dependence on an individual's memory or institutional memory to raise the flags against the ever-changing data or environment by using the ERROR/WARNING messages. We get the required results in the easiest of ways, by using the new options (WARN = OUTPUT, NUMERIC COLLATION = ON) and the obscure functions (PUTN). By writing robust code with all these procedures, functions, options and the merge equivalent technique, we not only reduce the number of lines of code, but also assure accuracy of results in an environment where the data and specifications are ever-changing.

The opinions expressed in this paper are the opinions of the author and should not be interpreted as the opinion or position of Omeros Corporation.

REFERENCES

Rick Langston. 2012. "Author of Proc Format" personal communication and follow-up from SGF

Pusarla, Sunil Kumar. 2011. "Fishing for the Correct P-Value: Automating the Derivation of Chi-square or Fisher's Exact Test P-Value" PNWSUG

RECOMMENDED READING

- Base SAS® Procedures Guide
- SAS® Language Reference Dictionary (9.2 or later)

CONTACT INFORMATION

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

Name: Sunil Kumar Pusarla and Paul Hamilton
Enterprise: Omeros Corporation
Address: 201 Elliott Avenue West
City, State ZIP: Seattle, WA 98119
Work Phone: 206-676-5000
E-mail: skumar@omeros.com, phamilton@omeros.com