Paper 20-28

Tips from the Hood: Challenging Problems and Tips from SAS-L

William W. Viergever, Viergever & Associates, Sacramento, CA

ABSTRACT

Ever have one of those days where you just can't find the solution to a problem? SAS-L is the electronic user group where you can get help for your SAS® problems from other SAS users.

However, even the experts of SAS-L sometimes need help or stumble over a question. This paper will cover some examples from SAS-L that challenged the experts in recent months as well as offered a few tips that you might not know. The author reads SAS-L many times a day and is a frequent SAS-L poster, although he admits that not all his posts are serious answers! <g>

Intended audience: basically – a good knowledge of Base SAS. This paper will cover several topics so attendees may find some examples more challenging than others.

INTRODUCTION

Someone once told me “the answer is the question” (and I hated it when he said that) and with respect to the SAS-L this is especially true. Often initial questions are posed that may be clear to the poster but are ambiguous to the many other SAS-Listas (lista – a member of a mailing list). Replies often iterate towards one or more correct answers or solutions

In addition, many times a post will touch on other, broader, issues and as such, will often elicit very detailed, almost academic, replies. Although some of these type replies may appear to be overkill, most of the Listas still enjoy these occasional pearls of wisdom and treasure them accordingly.

Keeping the above in mind, I’d like to now walk you through a few interesting “threads” (initial posts and relevant replies on a single subject) that caught my eye over the last year on the SAS-L. I’ll start w/ the original posts and then walk you through the various replies, providing my own “play-by-play” as to how the thread evolved.

Note: all posts can be found on-line at the SAS-L Archives: http://www.listserv.uga.edu/archives/sas-l.html

“FILL UP”

On a fine April Friday in 2002 someone started a thread with the subject “Fill Up”.

```
Hi people, I have a problem, I need to be
able to fill up.  I have a dataset but need
it to look like dataset2? If I could just
FILL UP the X Y I can easily retain the
parameters.

THANKS a lot.

Dataset1;

Parameter Value     X    Y
A         .99       .    .
AB        1         .    .
B         1.2       .    .
BB        0.2       .    .
C         0.3       .    .
CC        1.2       1    1
.         .98       .    .
.         .99       .    .
.         1.3       .    .
.         0.1       .    .
.         0.4       .    .
.         1.7       1    2

Dataset2;

Parameter Value     X    Y
A         .99  1    1
AB        1    1    1
B         1.2  1    1
BB        0.2  1    1
C         0.3  1    1
CC        1.2  1    1
A         .98  1    2
AB        .99  1    2
B         1.3  1    2
BB        0.1  1    2
C         0.4  1    2
CC        1.7  1    2
data one;
set two;
if mix(var1, var2) > 0 then do;
```

No, they weren’t having trouble at the gas station and although the suggestion was not tested and no output presented, the first reply ***seemed*** to provide a quick solution:

```
if X = . then X = 1;

for large numbers of missing variables
use an array:
array Nmbrs (*) _numeric_;
drop I;
do I = 1 to dim(Nmbrs);
if Nmbrs(I) = . then Nmbrs(I) = 1;end;
filling their carContinuation of body – after
source code.
```

For now the X’s will all be 1’s as the poster desired. However, the original poster came back with:

```
I Forgot to mention, that it isn't always
one, my X goes upto 32 and the Y 332. The
point was how to Fill up the dataset from the
bottom because my only reference to the XY
appears at the bottom.  Thanks.
```

Aha! - sighs the collective SAS-L, not just the X’s but the Y’s too. So another Lista proffers another suggestion:

```
This may not be very elegant (I'm not a very
elegant guy), but if DATASET1 is sorted on
PARAMETER (it appears that it is ascending
alphabetically), you could run PROC SORT by
DESCENDING PARAMETER.  Create a second
dataset, set DATASET1 (RENAME = (X = X1 Y =
Y2)), retain vars X and Y, and then let X =
X1 and Y = Y1.  Drop X1 and Y1 after setting
them equal to X and Y, and resort PARAMETER.

That should do it.
```

But does it?

Another replies with:

```
If I understand the problem correctly, the
following (untested) code should
help:

Data dataset1 ;
  set dataset2 ;
  if (parm ne '')  ;
    do i = 1 to 2 ;
      x = 1 ;
      y = i ;
      output ;
 end ;
run ;
```

It appears that both of these replies were closer to a solution for, with the original poster's elaboration, in that they both recognized the need to grab **some** data from the **end** of the data and use that to populate the **earlier** data.

Unfortunately, of the two replies, the former only populates X/Y data for a given PARAMETER, (which is not what the poster was after) and the latter reply, although it technically replicates the desired X and Y data values and fills up PARAMTER (given the poster's example data), it results in incorrect values for VALUE.

Once again, our original poster, liking that PARAMTER is now getting filled, offers a tad more info:

```
I wanted to fill in the character values
also. My dataset is not sorted and is
much longer than the example I posted I have
hundreds of coordinates( so I can't
sort until the x y is filled)!!!!  I've only
just read the file in with the filename
statement … And my parameter names aren't
A.B.C etc there random mixed names, I only
used what I did so...
```

Now the plot thickens, for another pleas for some clarity:

```
What you want to fill up into the char
fields? The sequence of the non-missing
fields or what?

… but there has to be any kind of logic! If
they are randomly mixed, would you distribute
them also randomly to the empty fields? So
the rest of any record is also kind of random
- why do you read them? Just use the ranuni
function!

So do you mean, the sequence is the same as
A, B, C, D and is repeated as A, B, C, D and
the second A, B, C... has the same contents
(or should be filled up with the same
contents...)?
```

Now before the original poster had a chance to answer these questions, another reply hit the SAS-L:

```
*** fill from the bottom;

data d2 Parm(keep = Parameter);
  do k = nobs to 1 by -1;
  set d1(rename = (x=x2 y=y2)) nobs = nobs
point = k;
  if x2 ne . then x = x2;
  if y2 ne . then y = y2;

  output d2;
  if Parameter ne '' then output parm;
  end;
  stop;
  drop x2 y2;
run;


*** reverse the order to the original and
fill "parameter";

data d3;

  do k = nobs to 1 by -1;
  set d2 nobs = nobs point = k;

  j +  -1;
  if j le (0) then j = nparms ;
  set parm point = j nobs = nparms;


  output;
  end;
  stop;
run;
proc print data = d3;
run;
```

Now this looks interesting. The comments mention "filling from the bottom" and "reversing the order", yet I see no PROC SORTs in this code. Perhaps we should delve a wee bit deeper.

In data step D2 we see the novel use of a loop and two SET statement options: NOBS and POINT.

NOBS tells us (generally) how many OBS are in a dataset and is thus a number available at compile time. Thus the loop has us reading the source dataset, D1 in this example, backwards from the **last** OBS to the first.

It does this by using the POINT option and setting POINT equal to the index of the loop.

This being Advanced Tutorials, many of you will recognize that POINT specifies a temporary variable whose (numeric) value determines which observation is read; i.e., POINT= causes the SET statement to use direct access to read a SAS data set.

So by "looping" from NOBS back to OBS 1, POINT reads each OBS from NOBS back to one. D2 thus looks like:

Obs	parameter	value	x	y
1		1.7	1	2
2		0.4	1	2
3		0.1	1	2
4		1.3	1	2
5		.99	1	2
6		.98	1	2
7	CC	1.2	1	1
8	C	0.3	1	1
9	BB	0.2	1	1
10	B	1.2	1	1
11	AB	1	1	1
12	A	.99	1	1

Well this looks more like it – except for the missing values for PARAMETER in OBS 1 through 6. But wait, the respondent has a last step where they reverse the order **and** fills PARAMETER.

In this last step we again see the use of a loop and the POINT option – but now we see two SET statements.

The data step again begins by creating a loop and, using the POINT option, reading the input dataset in backwards which clearly will reverse the OBS back to their original order.

Very slick! However, another novel trick here is the second dataset being read (also backwards!).

In their first step when they "filled from the bottom" they also outputted a second dataset called PARM, which was a unique set of non-missing PARAMETER values, albeit in reverse order:

```
Obs    parameter

 1        CC
 2        C
 3        BB
 4        B
 5        AB
 6        A
```

Well as Professor Higgins might say, "by George, I think we have it!" for now, as we reverse-read each of the reversed OBS with the "filled up" X's and Y's, we also reverse-read in the repeating, non-missing, values for PARAMETER.

Here is what each iteration of this last loop looks like:

```
parameter=A value=.99 x=1 y=1 k=12 j=6
parameter=AB value=1 x=1 y=1 k=11 j=5
parameter=B value=1.2 x=1 y=1 k=10 j=4
parameter=BB value=0.2 x=1 y=1 k=9 j=3
parameter=C value=0.3 x=1 y=1 k=8 j=2
parameter=CC value=1.2 x=1 y=1 k=7 j=1
parameter=A value=.98 x=1 y=2 k=6 j=6
parameter=AB value=.99 x=1 y=2 k=5 j=5
parameter=B value=1.3 x=1 y=2 k=4 j=4
parameter=BB value=0.1 x=1 y=2 k=3 j=3
parameter=C value=0.4 x=1 y=2 k=2 j=2
parameter=CC value=1.7 x=1 y=2 k=1 j=1
```

Another SAS-Lista also replied with a "loop & POINT" example, but before it hit the SAS-L the original poster had replied with:

```
I have my answer, thanks to all who reply'd I
know my interpretation was a bit
flaky but …
```

Which I think was an understatement <g>

Regardless, via this example, we see a typical scenario, played out daily, on the SAS-L: a (perhaps ambiguous) question is posed, eliciting (hopefully) numerous replies, slowly iterating towards a solution/answer.

In this example we were introduced to the concept of "looping" to read data, and the use of the POINT option to effect direct access reads of SAS datasets. Not bad for our first lesson, eh?

"HOWTO: FIND LAST FRIDAY OF MONTH"

Suffice it to say, after a tough week, the above subject line caught my eye. So, thinking this would be an easy one, I opened the post:

```
Does anyone have a tip of how to find the
last Friday of a month?

The last day of the month can be found with
the INTNX function:
lastDay=intnx('month',today(),0,'E');
This gives the SAS date value of the last day
of the current month.

Anyone have a suggestion?
```

Recognizing that the above INTNX function would return a SAS date of the last day of the current month, I glibly replied:

```
Do a WEEKDAY on this and go from there?
```

Which was similar to a few other replies that basically required code to IF/THEN your way from a given **last** day of month backwards to the Friday prior:

```
data _null_;
lastDay=intnx('month',today(),0,'E');
put lastday ddmmyy10.;
day=weekday(lastday);
put day=;
if day=6 then friday=lastday;
else if day=7 then friday=lastday-1;
else if day<6 then friday=lastday-(day+1);
put friday ddmmyy10.;
run;
```

Brute force, yes – but it did answer the question. However, as the SAS-L is wont to do, someone else posted:

```
Will this do it?

Use the date of the first of the following
month and apply this.

lastfri=intnx('week.6',date-1,0);
```

Clean, simple …Dot 6?? But before I could post a reply someone else wrote:

```
Can anyone tell me where the dot notation on
the interval is documented?
```

And of course, the SAS-L responded:

```
It's in SAS Language Reference:
Concepts; Part 1: SAS System Concepts;
Dates, Times, and Intervals
```

Not being familiar with this notation I went to the above reference and found that:

```
The form of an interval is

name<multiple><.starting-point>
```

where

```
multiple
creates a multiple of the interval. Multiple
can be any positive number. The default is 1.
For example, YEAR2 indicates a two-year
interval.

.starting-point
is the starting point of the interval. By
default, the starting point is 1. A value
greater than 1 shifts the start to a later
point within the interval. The unit for
shifting depends on the interval, as shown in
the following table. For example, YEAR.3
specifies a yearly period from the first of
March through the end of February of the
following year.
```

Thus, for a given date, we take the last day of that month, and then figure a week interval for that last day, where the interval begins with the 6th day –i.e., Friday.

```
lastfri =
intnx('week.6',intnx('month',dt,0,'E'),0);
```

Although I knew about the INTNX function, I wasn't aware of the nuances of the interval with it's multiple and starting point parameters. To me, this is what make the SAS-L so valuable. Routine reading of the SAS-L is almost like you're enrolled in a SAS "Continuing Ed" class. In fact, as a follow-up to this thread, a SAS-L regular, referring to the documentation on intervals, wrote:

```
After looking at it, I first thought you
could do the same thing with
months. You cannot.  They gave an example
"MONTH2.2" so I tried "MONTH1.3",
no good.  Ah-ha, new system.  So I tried

905  data _null_ ;
906     dt = "1nov2002"d ;
907     y = intnx ("month.3" , dt , 0 ) ;
908     x1 = intnx ( "month3.1" , dt, 0 ) ;
909     x2 = intnx ( "month3.2" , dt, 0 ) ;
910     x3 = intnx ( "month3.3" , dt, 0 ) ;
911     format _all_ date9. ;
912  run ;

NOTE: Invalid argument to function INTNX at
line 907 column 8.
dt=01NOV2002 y=. x1=01OCT2002 x2=01NOV2002
x3=01SEP2002 _ERROR_=1 _N_=1
NOTE: Mathematical operations could not be
performed at the following places. The
results of the operations have been set to
missing values.
Each place is given by: (Number of times) at
(Line):(Column).
      1 at 907:8
NOTE: DATA statement used:
      real time           0.04 seconds

Can anyone explain in English what this
means?  (I am not interested in the mistake
[I broke the rules to indicate the month
rules are different from the week rules and
to get some output cheaply].  It is the
"good" values that I want to understand.)  At
first I thought "MONTHm.n" might mean the nth
month of a sequence of m months beginning
some time.  X1 and X2 appear
to be consistent with this theory.

Here if my final attempt at understanding
before asking for help.

1071  data _null_ ;
1072     dt = "1jan2002"d ;
1073     do until ( dt > "1dec2002"d ) ;
1074        x1 = intnx ( "month5.1" , dt, 0 )
;
1075        z1 = intnx ( "month5.1" , dt, 1 )
;
1076        x2 = intnx ( "month5.2" , dt, 0 )
;
1077        x3 = intnx ( "month5.3" , dt, 0 )
;
1078        x4 = intnx ( "month5.4" , dt, 0 )
;
1079        x5 = intnx ( "month5.5" , dt, 0 )
;
1080        y1 = intnx ( "month3.1" , dt, 0 )
;
1081        z2 = intnx ( "month3.1" , dt, 0 )
;
1082        y2 = intnx ( "month3.2" , dt, 1 )
;
1083        y3 = intnx ( "month3.3" , dt, 0 )
;
1084        put dt= / (x1 z1 x2-x5) (=) / (y1
z2 y2-y3) (=) ;
1085        dt = intnx ( "month" , dt , 1) ;
1086     end ;
1087     format _all_ date9. ;
1088  run ;
```

One SAS-Lista responded and succinctly described the "month3.x" interpretations:

```
MONTH3.1 means a 3-month interval beginning
with the default month of Jan. Thus the
intervals will be Jan - Mar, Apr -Jun, Jul -
Sep, Oct - Dec. In your first example, dt is
1NOV2002 which falls within the Oct - Dec
interval. The result in x1 is the first day
in the interval, 1OCT2002. For x2, the
function uses another 3-month interval, but
the interval begins in Feb. The intervals are
Feb - Apr, May - Jul, Aug - Oct, Nov - Jan
(of following year). dt falls within the Nov
- Jan interval, so x2 contains the first day
in that interval, 1NOV2002. For x3, dt is
within the interval of Sep - Nov, so the
result of the function is 1SEP2002
```

This seemed fine given that are exactly 4 3-month intervals in a year. But what then is the "month5.x" interpretation? Another esteemed Lista responded with:

```
I guess 5 month intervals are in blocks from
Jan60, so.....

15JAN02 is in the "month5" interval from
SEP2001-JAN2002 (using simple "month
arithmetic"
   Jan2002=( 2002*12+1) =24025
   Jan1960=( 1960*12+1) =23521
   dif = 504
   mod(dif, 5) = 4
So Jan2002 is the final month of the month5
interval starting Sep2001

The .2 offset implies intervals starting as
at Feb1960 Using the month arithmetic from
Feb1960 results in a dif=503 and mod(503, 5)
= 3, so the month5.2 interval covering
Jan2002 runs from Oct2001  (= Jan2002 -3
months )
```

Of course! - the infamous "January 1960" rears its head once again <g>. And sure enough, the above mentioned OnLineDocs explains this clearly under the sub-topics "Multiunit" and "Shifted" Intervals. In fact, it even describes why "MONTH1.3" didn't work:

```
In addition, you cannot shift an interval by
itself. For example, you cannot shift the
interval MONTH because the shifting subperiod
for MONTH is one month and MONTH contains
only one monthly subperiod. However, you can
shift multi-unit intervals by the subperiod.
For example, MONTH2.2 specifies bimonthly
periods starting on the first day of the
second month.
```

This has been an example of those SAS-L questions where, although they may appear elementary, end up triggering an

exchange that exposes one to some of the nuances of SAS – for both novices and wizards/gurus/mavens alike.

"RENAME MACRO.CODE"

Posts involving the renaming of large numbers of variables appear frequently on the SAS-L. What caught my eye on this one was that the poster was asking for macro code to accomplish this. Macro code is often the answer to posts such as this; not necessarily the question.

```
I've written some code to rename 250
variables in my dataset. The code works but
it is extremely inefficient. Too much
iteration. Would appreciate your advice.
Thanks.

* to produce sample data ;
%let class=alpha ;
data oldset_&class ;
 do i=1 to 250 ;
  val = int(ranuni(1)*100) ;
 output ;
 end ;
 drop i ;
run ;
proc transpose data=oldset_&class
out=newset_&class(drop=newname)
name=newname prefix=Var  ;  run ;

* can the following code be made more
efficient? too much iteration.
can a call symput statement be used to dump
the entire 'chngname' string
into macro rename1 ? ;
%macro rename1(chgname=chgname,
class=&class);
options mprint mlogic ;
data newset&class ;
 set newset&class ;
 rename &chgname ;
 label &chgname ;
run ;
%mEnd rename1 ;

%macro rename2(class=alpha) ;
options mprint mlogic ;
proc contents data=newset_&class
out=list_&class (keep=name) noprint ; run ;
data name_&class ;
 set list_&class ;
 if name not in ('Var2','Var100') then
chgname =
trim(name)||"="||trim(name)||"_&class" ; *
Var2 & Var100 excluded ;
 call execute
('%rename1(chgname='||chgname||",class=_&clas
s)" ) ; * rename2
calls rename1 ;
run ;
%mEnd rename2 ;

%rename2(class=alpha)
```

Let's see now; we have a nested macro to do the renames, with one macro doing a PROC CONTENTS to get the variable names, and then a CALL EXECUTE of the other macro which uses a data step to apply the renames.

But the nature of the CALL EXECUTE is such that it's calling macro one repeatedly, once for every variable to be renamed. The poster was correct, this is inefficient code.

Another SAS-Lista spotted this promptly offered a little admonishment:

```
Horror of horrors in macro!  Repetition
should be wielded like a scalpel not a sledge
hammer.  See the archives and search for
"rename" any two month period will probably
be enough.

In short, use PROC SQL with the dictionary
files to the names of the variables and make
the macro variables.  then use PROC DATASETS
to apply the renames and labels.  No data at
all should be read! (Unless you need a copy
of the data and then only once.)
```

To which the original poster came back with a revised rename1 macro:

```
%macro rename1(chgname=chgname, class=&class,
name2=name2);
proc datasets lib=work nodetails nolist ;
modify newset&class ;
rename &chgname ;
label &name2=&name2 ;
run ;
quit ;
%mEnd rename1 ;
```

Now, however, even though they were now using PROC DATASETS (vs. reading the entire dataset), because of all the CALL EXECUTEs, they were still doing it as many times as there were variables to rename.

Another SAS-Lista spotted this and offered the code for the above mentioned PROC SQL and PROC DATASETS approach

```
To avoid all those CALL EXECUTES you could

%let class=alpha;

%macro rename (libname=work, dsn=,
class=alpha);
proc sql;
select trim(name) into :nm1 through :nm9999
   from sashelp.vcolumn
   where libname eq "%upcase(&libname)"
   and upcase(memname) eq "%upcase(&dsn)"
   and upcase(name) not in ('VAR2', 'VAR5')
   ;
%let n_nm = &sqlobs;

proc datasets lib=&libname nodetails nolist;
modify &dsn;
rename
%do i = 1 %to &n_nm;
   &&nm&i = &&nm&i.._&class
%end;
;
%mend;

%rename(dsn=newset_&class,class=alpha);
run;
```

This appeared to end this thread. The original poster had received both code for the recommended solution, as well a some discussion on why it was preferred, i.e., avoiding reading any data, and avoiding the repetition due to all the CALL EXECUTEs.

Yet, almost as a challenge, the SAS-L offered an alternative approach explicitly utilizing CALL EXECUTE:

```
%let class=alpha ;
data _null_ ;
   set new_&class ( obs = 1 ) ;
   array __nums (*) _numeric_ ;
   call execute ( "proc datasets lib = work ;
modify new_&class ; rename " );
   do __i = 1 to dim ( __nums ) ;
     call execute(trim(vname(__nums[__i]))||
"="||trim(vname(__nums[__i]))||"_&class");
   end ;

   call execute ( "; label " );
   do __i = 1 to dim ( __nums ) ;
      call execute ( trim(vname(__nums[__i]))
|| _&class="||quote(trim(vname(__nums[__i])))
);
   end ;
   call execute ( ";run;quit;" ) ;
run ;
```

In this scenario, we do read just the first OBS of data but that is used to generate an array of the variables to be renamed.

From there, the code uses a null data step and begins “writing” a PROC DATASETS via a CALL EXECUTE, and continues, looping through the array, using the VNAME function to grab the variable names and constructing the actual rename (and label) lines which via more CALL EXECUTES are added to the developing PROC DATASETS. A final CALL EXECUTE runs the PROC DATASETS.

I found this piece of code to be pretty slick; a nifty alternative to the more common PROC SQL / PROC DATASETS approach. Bottom-line, another arrow for my SAS-quiver.

CONCLUSION

The SAS-L is a very cool place. The Archives contain a trove of SAS knowledge, and the List proper, a community of fellow users always available for help and advice. The examples given herein cover just a few of the various topics routinely discussed on the List; ones that caught my eye this last year.

I hope the examples discussed today did not bore you. Writing this paper proved somewhat a challenge. As I was reviewing various posts of interest, I found I had to skip some because many of them either evolved into very complex threads that didn’t afford brief summarization (or I didn’t understand?), or there were others that contained so much code proper that either they were either self-explanatory or could have been SUGI papers by themselves. Also, there many parts of SAS I do not use regularly, and hence tend to skip posts with those subjects.

I’ve been coding SAS since 1976 and tend to get set in my ways. I think that’s why I like the SAS-L so much: I continue to learn new things about SAS from it; almost daily.

Won’t you give it a whirl, too?

REFERENCES

As mentioned above, all examples in this paper have been culled from the SAS-L and can be found at:
http://www.listserv.uga.edu/archives/sas-l.html
The above (quoted) section headings above are from the SUBJECT lines for these 2002 SAS-L posts; search accordingly.

ACKNOWLEDGMENTS

First off, to Joe, the List owner – for if not Joe, no List. Secondly, to the misc. Masters, Wizards, Gurus, and even Mavens – for if not them, no “interesting” List. Thirdly, to the masses, for with out us, who would the Masters, Wizards, Gurus, Mavens expound to?

And lastly, to Deb Cassidy for the idea & invite, and her patience in putting up with this harried first-time writer.

CONTACT INFORMATION

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

William W. Viergever
Viergever & Associates
2920 Arden Way Ste N
Sacramento, CA 95825
V: 916-483-8398
F: 916-486-1488
wwvierg@attglobal.net