Paper 249-25

A Personal View of SAS-L as a Teaching Tool

Ian Whitlock, Westat, Rockville, MD

Abstract

SAS-L is an on-line SAS® Users Group with over 2000 members around the world, plus a huge number of silent or nearly silent listeners who follow the discussions via the associated news group, comp.soft-sys.sas.

After a short introduction and history of SAS-L I will consider questions such as

- Can you learn from asking questions?

  Or is there too much misinformation?

- Can you learn by giving answers?

  Or do you need to be an "expert"?

- Is SAS-L sufficiently recognized by training and support people?

  Or Should you add SAS-L to your learning kit?

I will then provide answers in terms of my participation on SAS-L using examples of interesting coding problems.

In summary, this presentation is for anyone into SAS who has not been aware of SAS-L and for all the SAS-Ler's who want more.

History

SAS-L began in 1986 with two list servers. One was located in Austria and the other at the University of Georgia. In the early days it was largely a group of systems people who supported SAS users.

It has grown to a list of approximately 2000 who receive mail from the one of the list servers supporting SAS-L and a group with an unknown size that participate via news group counterpart, comp.soft-sys.sas.

FAQ

FAQ stands for "frequently asked questions." and is commonly used in group lists. A FAQ has been maintained by Tim Berryhill in recent years and posted several times per year. In general these questions are about the list instead of about SAS. Perhaps the two most common questions are:

1. How do you join SAS-L?
2. How do you quit?

To join send the one line message:

SUBSCRIBE SAS-L Your Name

to one of the following list servers:

Marist Univ	listserv@vm.marist.edu
VaTech Univ	listserv@listserv.vt.edu
Univ of Vienna	listserv@akh-wien.ac.at
Univ of Ga	listserv@listserv.uga.edu

To leave SAS-L send the message

UNSUBSCRIBE SAS-L

to the same list server from which you joined. The system recognizes you by the e-mail address from which you joined, hence you must use the same address to quit. If this is not possible then the manager of the appropriate list server must be contacted.

To take a vacation without quitting, send the list server the message

SET SAS-L NOMAIL

You may receive each message separately or you may get a daily compilation of the day's messages by sending the list server the message

SET SAS-L DIGEST

All of the above are instructions about how to handle your membership in SAS-L. To send a message about SAS to the group, you send it to an address formed from the above addresses by replacing

listserv@

with

SAS-L@

As an alternate possibility, you may choose to view the messages via the news group

comp.soft-sys.sas

The advantage is that it doesn't fill your mailbox. The disadvantage is that messages in general are older. Hence answers are already posted to the questions you are currently reading. On the other hand it is common to see answers one and sometimes two weeks after they are posted.

Common SAS Questions

One popular question is - how do you read a comma delimited external file? Inevitably the advice comes back to use the DSD option on the INFILE statement and to catch up on the documentation in Technical Report P-222. Hence, SAS-L provides a tool for measuring the effect created when the SAS Institute published the basic reference manuals for version 6.06 and then made many

changes to the language in version 6.07 without updating the manuals in one place.

Another popular question is how to convert from a character variable to a numeric one or vice versa. The answer is simple, but requires putting together several pieces of information. Here is my favorite for converting a numeric ID to a character ID with leading zeros. The leading zero problem is not so often asked, but there are enough awkward answers whenever it is, that it is worth including in the problem.

```
data fixed ( drop = temp ) ;
   set problem (rename = (id=temp)) ;
   id = put ( temp , z8. ) ;
run ;
```

The old variable must be dropped and a renaming is necessary to give the replacement variable the old name. In addition, one must know about the PUT (or INPUT) function in order to provide a reasonable answer to this question. Now where should a beginner look to find the answer to this common problem? Difficult to say, that is why it is so often found on SAS-L.

Another common area for problems is moving between Excel and SAS or Access and SAS. Here the difficulties lie in the fact that the answer lies hidden in another SAS/product instead of being part of the base system.

Questions easily solved with a RETAIN statement and BY processing form another core group of "newbie" questions.

The LAG function is responsible for a lot of questions, in part because people think it gives them information about the previously read observation. In fact, it merely returns the value of the last call. This means that you are liable to get unexpected results when you make the call in the consequent to an IF statement. Questions about looking ahead to the next record are also common. I am usually surprised at how rarely one sees the simple solution:

```
data out ;
  merge in
        in (obs = 2
            keep = ...
            rename = ( ... )
           ) ;
  /* no by statement */
. . .
run ;
```

Questions about identifying data sets with 0 observations abound. This is due, in part, to the fact that relatively few people realize that the DATA step is data driven and thus usually stops on some form of input statement when there is no more data to be read. Consequently the best place for the test for end of file is before the SET statement.

A Month of Quickies from SAS-L

Here are some of the problems that I saved from SAS-L in the month of December 1999. They, of course, present a rather prejudiced SAS-L view, mine. The names involved are a accident of the month that I picked. However they are common responders and worth looking for. On the other hand, many very good people have been left out because I did not happen to save their messages. It does not mean that there wasn't a lot more to learn from SAS-L in December.

The subjects chosen mainly involve code because this is what interests me most. Many questions, of course, do not involve code. For example,

> What driver do you use for <printer, graphic, etc.>?
> Why isn't my export data set recognized?
> How do you get reports into Word?

How do you skip a line after each record using PROC REPORT? The master, Ray Pass gave a simple answer - add an OBS variable to the data set and then use

```
proc report nowd data=test2;
    column ...other vars... linenum;
    define linenum / order noprint;
    break after linenum / skip;
run;
```

One could add a simple COMPUTE AFTER to do any kind of processing after each line. For example, skipping multiple lines.

How do you get a count of the rows for each page in a PROC TABULATE? In the original question, a crosstab giving hours was needed by employee and phase for each week. Here one wanted a count of distinct employees for each week.

```
PROC TABULATE DATA=TIMEREC;
  CLASS EMPLOYEE WEEK PHASE;
  VAR HOURS;
  TABLE WEEK,
        EMPLOYEE ALL,
        SUM*HOURS=' '*(PHASE ALL);
RUN;
```

Ya Huang, a relatively recent, SAS-L expert responded creatively. Modify the value of WEEK in an SQL step to include the required information.

```
proc sql;
  create table timerec (drop=week
      rename=(week_nem=week)) as
```

```
  select *,
         week||" (N="
       ||put(count(distinct employee)
             ,2.)||")" as week_nem
  from timerec
  group by week
  ;
quit ;
```

Jack Shoemaker was working with version 8 producing an HTML page. He wanted to flow a variable using the HTML command "
" as in

```
yourcolumn = 'Dancer'
   ||'<br>'||'Prancer'
   ||'<br>'||'Blitzen';
```

The problem was that ODS was quoting the HTML commands. In this case, the SAS Institute Tech Support came to the rescue, but Jack let us all in on the secret. The answer was to override the default ODS behavior with a DEFINE statement.

```
define yourcolumn/style(COLUMN) =
   {ProtectSpecialChars=FALSE};
```

David Ward noted that memory was eaten up when one did not delete SCL lists after use. The Australian expert, Mark Bodt reposted an old SAS-L tip of his. The utility macro man, Richard DeVenezia responded with his solution:

```
%macro DELLIST (aList, recurse=N);
  if &aList>0 and listlen(&aList)>=0
  then &aList =
          dellist(&aList,&recurse);
%mend;
...
a=makelist();
b=makelist();
c=makelist();
...
%dellist (a)
%dellist (b)
%dellist (c)
```

Then the English expert, Andrew Ratcliff added a note on invoking the application to check for missing deletions.

```
==> AFA C=lib.cat.prog.scl
       AUTOTERM=VERBOSE
```

Now any persisting classes are noted at the end of execution.

Do you need the login name under Windows 95?

From the Netherlands, Lex Jansen provided a pointer to method using a WINAPI and the MODULEN routine.

http://www.pwcons.com/Tips/base/macro.html#login

How do you use the SQL pass through facility and ODBC to connect to Excel? The West Coast expert, Pete Lund responded quickly with

1. In Excel name the range of cells you're interested in (i.e., MyData)
2. In the ODBC administrator set up a data source name pointing to the spreadsheet (i.e., TheSheet)
3. In SAS, use SQL to reference the data: the spreadsheet will be the "database" and the named range will be the "table."

The following query will create a SAS dataset called SAScopy that is a copy of the spreadsheet range. The first row of values will be treated as the variable names.

```
proc sql;
   connect to odbc(dsn='TheSheet');
   create table SAScopy as
      select *
      from connection to odbc
        (select * from MyData)
   ;
quit;
```

The point here is that I did not ask how to solve these problems. I learned by simply reading the answers to questions that I found interesting.

Others would choose a completely different set of responses as valuable for the same time-period. In general I tend to ignore a lot of the statistics and graphics information along with answers to questions that I know how to answer.

Some Quickies to SAS-L

Here are a few of my recent public responses to SAS-L questions.

Is there a function to count the words in a string? In the last round of this question in December 1999, Paul Dorfman, the efficiency expert, responded with a clever function style macro pieced together from past discussions on SAS-L.

```
%macro nwords(s);
    (compress(&s) ne ' ') *
    (length(left(compbl(&s)))-
     length(compress(&s))+1)
%mend nwords;
```

The idea here is to count the spaces between words and add one. (Other separators are not considered to produce words. For example "A*B" is one word, not two. For some applications it will matter, but for many it will not.) The function COMPBL is used to reduce the number of spaces between words to one, and the function COMPRESS is needed to remove all the spaces.

Now it is a simple matter of subtracting lengths. The first factor is needed to handle the special case of no words. In SAS, all strings have at least one byte, so the LENGTH function returns 1 when applied to a blank string. Since the macro just consists of function calls it can be used as function. For example,

```
nwords = %nwords(string);
```

The same question has also been asked for macro variables. Here the question of efficiency is rather important because the common answer, "Use %QSCAN in a loop and count the iterations of the loop", is rather slow. I responded with a solution based on the above work for SAS Version 8.

```
%macro nwords(s);
  %local crect onesp nosp ;
  %if %length(&s) = 0 %then 0 ;
  %else
  %do ;
     %let crect = %eval(
  %qsysfunc(compress(&s)) ne %str( )
     ) ;
     %let onesp =
            %qsysfunc(trim(
            %qsysfunc(left(
            %qsysfunc(compbl(&s))))))
     ;
     %let nosp =
           %qsysfunc(compress(&s)) ;
     %eval(&crect
           *(%length(&onesp)
              -%length(&nosp)
            )+1
          )
  %end ;
%mend nwords;
```

It is very fast and efficient. For example, on a string of 540 3-byte words with some extra leading and trailing spaces my macro executed in 0.11 seconds. The standard loop, scan-and-count-word method took 6.05 seconds on the same machine. That represents an time improvement factor of 55!

Unfortunately the method does not work under SAS 6.12 because the macro facility cannot handle long tokens. Note that the DATA step functions can handle 32K strings when applied in the macro facility using the macro function %SYSFUNC, but each word must be less than 200 bytes. The COMPRESS function makes a single token, so it cannot be used as it is in the above solution. The answer is to leave the blanks and remove the rest, since the macro function %LENGTH does count blanks in a string of blanks. Here is the macro that I posted for version 6.12 (Changes from the SAS-L version: The macro variable, STD, was introduced here to make the reading easier in this format, and the omission of a digit was corrected.)

```
%macro nwords(s);
  %local onesp len std ;
  %let std =
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_ ;
  %if &s = %str() %then 0 ;
  %else
  %do ;
    %let onesp =
      %qsysfunc(trim(
      %qsysfunc(left(
      %qsysfunc(compbl(&s))))))  ;
    %let len =
     %qsysfunc(compress(&onesp,&std))
    ;
    %if &len = &onesp %then 0 ;
    %else %eval(%length(&len)+1) ;
  %end ;
%mend nwords;
```

In this version, other separators become a problem. For example, "A*B", now gets counted as two words since the compressed form is "*" which has a length of one and one is added to this length to obtain the number of words. In practice, the word counting problem is applied to a space separated sequence of SAS variable names; hence the different behavior should not be a problem. For comma separated lists, one could translate commas into spaces, and then apply the above macro.

A beginner asked, how could an array be used to improve the program?

```
DATA CLASSES (KEEP = HOUR
        CLASSA CLASSB CLASSC CLASSE
        CLASSF CLASSG CLASSH CLASSM
        CLASSN CLASSP CLASSR CLASST
        CLASSU CLASSV CLASSY TOTAL);
  SET JOBS;
  JHRSTR = HOUR(TIMEPART(JSTRTIME));
  JHREND = HOUR(TIMEPART(JENDTIME));
  DO HOUR = JHRSTR TO JHREND;
   IF JOBCLASS = 'A' THEN CLASSA = 1;
   IF JOBCLASS = 'B' THEN CLASSB = 1;
   IF JOBCLASS = 'C' THEN CLASSC = 1;
   IF JOBCLASS = 'E' THEN CLASSE = 1;
   IF JOBCLASS = 'F' THEN CLASSF = 1;
   IF JOBCLASS = 'G' THEN CLASSG = 1;
   IF JOBCLASS = 'H' THEN CLASSH = 1;
```

```
    IF JOBCLASS = 'M' THEN CLASSM = 1;
    IF JOBCLASS = 'N' THEN CLASSN = 1;
    IF JOBCLASS = 'P' THEN CLASSP = 1;
    IF JOBCLASS = 'R' THEN CLASSR = 1;
    IF JOBCLASS = 'T' THEN CLASST = 1;
    IF JOBCLASS = 'U' THEN CLASSU = 1;
    IF JOBCLASS = 'V' THEN CLASSV = 1;
    IF JOBCLASS = 'Y' THEN CLASSY = 1;
    TOTAL = 1;
    OUTPUT;
  END;
run ;
```

I reacted in part because anyone who has enough instinct to realize that this wall-paper style of code is bad should be helped. In testing the code I soon found out that jobs that run past midnight might not pass the big DO-loop test.

Then, in the original question, I saw the rest of the code. The purpose of outputting a record for each hour was so that a PROC MEANS could sum up the 1's in each job class, so I changed the purpose of the DATA step to output the final counts and left it to PROC PRINT to do the total.

```
proc format ;
  invalue jobx
    "A" = 1  "B" = 2  "C" = 3
    "E" = 4  "F" = 5  "G" = 6
    "H" = 7  "M" = 8  "N" = 9
    "P" = 10 "R" = 11 "T" = 12
    "U" = 13 "V" = 14 "Y" = 15
  ;
run ;

DATA CLASSES (KEEP = jclass count ) ;
  length jclass $ 5 ;
  array jobc (15)   _temporary_ ;
  array letter (15) $ 1 _temporary_
    ( "A" "B" "C" "E" "F" "G" "H" "M"
      "N" "P" "R" "T" "U" "V" "Y" ) ;

  if eof then
  do ;
    do j = 1 to dim ( jobc ) ;
       jclass = letter ( j ) ;
       count = jobc ( j ) ;
       output ;
    end ;
  end ;

  SET JOBS(keep = jstrt jend jobclass

          ) end = eof ;

  DO jt = jstrt to jend+3600 by 3600;
    hour =  HOUR(TIMEPART(jt)) ;
    jobc (input(jobclass, jobx1.))+1;
  END;

PROC PRINT data = classes NOOBS ;
    sum count ;
run ;
```

Paul Dorfman, ever thinking about efficiency, objected to the use of the INFORMAT as too time-consuming. He suggested calculating the index from the position of the letter in the value of a string variable.

```
jobs = "ABCEFGHMNPRTUVY" ;
```

For real speed he then increased the array size to 256 and used the RANK function to locate the element of interest. Paul uses this trick over and over - don't minimize the size of the array, include dummy places to speed up the look up of the index. Minimizing the space for the array only results in more time to calculate the index.

The point in both of these cases is that the final results are often a collaborative effort better than any one individual would have suggested.

When Craig Lake asked a SAS Version 6.12 question, "How do you read character strings longer than 200 bytes?", I wrote back read it into an array of character variables. He then posted the additional information that it was a comma separated file that he wanted to read. This makes it an interesting problem. A formatted read is needed to read the value into an array, and list input with the DSD option is needed to handle the comma separated data. I dashed off a two step process - one to obtain the beginning and end of the problem variable on each record, and one to then read the problem field. Here is the code, assuming 10 instead of 200 is the maximum character length to simplify testing.

```
data w ( drop = skip ) ;
   rec + 1 ;
   infile cards col = c dsd ;
   input skip $ skip $ @ ;
   begcol = c + 1;
   input skip $ @ ;
   endcol = c - 2 ;
cards ;
1111,222,"1234567890this is a
comment",4,5
1,2,"another very  loooong
comment",9999,333
;
```

List input is used with the DSD and COL= options to determine the starting position of the long field; 3 in this case. Armed with this information, one can now perform formatted input for the field using a trailing "@" to hold the line while reading in a loop. Essentially a one-to-one merge is performed since each record of the SAS data set corresponds to one record of the external file.

```
data w2 ;
   array com (10) $ 10 ;

   set w ;
```

```
   infile cards col = c ;
   input @begcol @ ;

   do i = 1 to dim (com) while (c <
endcol);
      len = min ( 10 , endcol - c ) ;
      put len= ;
      input com(i) $varying. len @ ;
   end ;
cards ;
1111,222,"1234567890this is a
comment",4,5
1,2,"another very  looooong
comment",9999,333
;
```

Independently, David Ward assumed that the problem field was last. Thus he could determine the length from the starting position and the record length. He then gave code to do the job in one step. After seeing David's solution I realized how easy it is to combine my two step process into one.

```
data w2 ;
   array com (10) $ 10 ;

   infile cards col = c dsd ;
   input skip $ skip $ @ ;
   begcol = c + 1;
   input skip $ @ ;
   endcol = c - 2 ;

   input @begcol @ ;

   do i = 1 to dim (com)
         while (c < endcol);
      len = min ( 10 , endcol - c ) ;
      put len= ;
      input com(i) $varying. len @ ;
   end ;
cards ;
1111,222,"1234567890this is a
comment",4,5
1,2,"another very  looooong
comment",9999,333
;
```

The lesson here is that two solutions are better than one, because it gives you more ideas to play with. That is an important theme on SAS-L. It is very instructive to see the variety of ways different people choose to answer the same question. Far more variety is given on SAS-L than anyone can expect from a single author.

I have not only had a chance to see problems very different from those in my work; I have also had the chance to study solutions that I would never have dreamed of, thus enhancing my programming ability. No matter what your level of programming ability, there is much to learn from attempting to answer SAS-L questions.

A Big Lesson

All of the previous topics have presented little things that could be learned from SAS-L, in part, because that is nature of most of what is on SAS-L. Now I would like to turn to a topic too big for this paper. For the Christmas of 1998 Paul Dorfman sent to SAS-L a series of solutions to problems involving the hashing technique. Paul works on "The Big Iron" as Tim Berryhill would say. He specializes in time and CPU efficient solutions to problems involving millions of records. For example, the question might be:

> I have a file of 40 million records from which I have to extract a small set of about 100 thousand. I cannot afford the time to sort the large file in order to merge. What can I do?

In the old days, the early 90's, the experts would usually suggest a format or possibly a binary search. Occasionally someone would mention hashing, but I don't remember anyone ever spelling it out. Well Paul did in all its gory details with executable code. Moreover, he has done it several times over the past year, so that anyone who wants to find examples has a ready source in SAS-L.

For me it was an opportunity to spend several hours getting intimately aquainted with this important technique. The sequence is a jewel even by SAS-L standards in my opinion.

David Pidder had another reaction and wrote a rather bitter message about people who do not know how to use SAS. Soon after that someone asked about how to handle a frequency problem where the number of distinct values was very large, say a million. David took the opportunity to say that PROC SUMMARY was the answer and one shouldn't let any self-appointed experts say to use a DATA step.

Too much for me, I generated a set with the properties described using hundred-thousand different values and showed how an array could produce the counts in 12 seconds. I then tried the suggested PROC SUMMARY, but killed the job after 5 minutes.

Karsten Self, a rather erudite responder to questions involving UNIX, took the more humorous approach and made all members of SAS-L Self-appointed SAS experts.

Control Information - Code or Data?

William W. Viergever had a labor intensive problem. Each quarter he has to prepare numerous Excel files summarizing information about the top 30 CPT-4 codes (proc codes). His system was to run a program to find the top 30 codes for each report since they changed each quarter. He then used this information to manually modify a program that prepared the Excel files. The number of Excel files based on this system had grown to his breaking point, so he asked for advice from SAS-L.

To make matters worse, his code had simply put the relevant proc codes at the top of each column. The client now wanted nice labels with descriptive information instead of codes.

Here is the second program, written after the top Proc Codes have been determined.

```
/* map top 30 Proc Codes (and E&M's)
   to variables */
data tmp01;
  set &dsn1..&file1;
  *Map Top 30 Proc Codes(and E&M's);
  if proccode = "E&M's" then
     r_e_ms  = net;
  else if proccode = '95165'  then
     r_95165 = net;
  else if proccode = '95004'  then
     r_95004 = net;
  . . .
  else
     resid = net;
run;

/* First level summary */
proc summary data=tmp01 missing nway;
  class hpcode provid membid;
  id spec name;
  var net
      r_e_ms
      r_95165
      r_95004
      . . .
      resid;
  output
      out=print1a(drop=_type_ age)
      sum=
      mean(age)=mage;
run;

/* summary of the summary */
proc summary data=print1a
     (rename=(_freq_=lines mage=age))
     missing nway;
  class hpcode provid;
  id spec name;
  var net
      r_e_ms
      r_95165
      r_95004
      . . .
      resid
      lines
      age;
  output
    out=print1b(drop=_type_
                rename=(_freq_=elig))
    sum=
      mean(age)=mage;
run;

/* Create TAB delimited file */
data _null_ ;
  set print1b;
  file raw1;
  if _n_=1 then put  /* Headers */
     . . .
     'Net'           '09'x
     '99-Codes/E&M' '09'x
     '#1:  95165'   '09'x
     '#2:  95004'   '09'x
     . . .
     'All Others' ;
  put  /* variables */
     . . .
     net             '09'x
     r_e_ms          '09'x
     r_95165         '09'x
     r_95004         '09'x
     . . .
     resid;
run;
```

Here we have the classic problem, run one program to determine results enabling one to write the desired program. Peter Lund and Paul Dorfman quickly replied with answers showing how to generate the above program. One used a time-honored method - a DATA _NULL_ step with PUT statements to write the code. The other achieved the same end with more macro code and an array of macro variables to hold the names of the Proc Code variables. Neither, solution addressed the client's problem of more descriptive information than just the codes.

I was intrigued by William's comment:

> To para-phrase Ian, I think it's one of these jobs where one mixes data with code (?).

and his offer of a bottle of fine wine.

He had sat in the front row of my talk, "Code or Data?", given in the advanced tutorial section at last year's SUGI, Well his problem didn't suit the particular example that I had presented, but he was right that it would yield to an interchange of code and data.

From this point of view, using variable names (code) like R_95165 is wrong. Proc Codes are data and should stay data. The variable names should be fixed, V1, V2, ... This was a large part of William's problem.

An informat mapping Proc Codes into the array V1, V2, etc. was missing. Armed with this informat I suggested:

```
/* map top 30 Proc Codes (and E&M's)
   to variables */
```

```
data tmp01 ;
  set &dsn1..&file1 ;
  array v (0:&n) ;
  v ( input ( proccode , vfmt. ) ) =
net ;
run ;

/* First level summary */
proc summary data=tmp01 missing nway;
  class hpcode provid membid;
  id spec name;
  var net v0 - v&n resid age;
  output out=print1a(drop=_type_ age)
         sum= mean(age)=mage;
run;

/* summary of the summary */
proc summary data=print1a
               (rename=(_freq_=lines
mage=age))
         missing nway;
   class hpcode provid;
   id spec name;
   var net v0 - v&n resid lines age;
   output
       out=print1b(drop=_type_

rename=(_freq_=elig))
       sum=
       mean(age)=mage;
run;

/* Create TAB delimited file */
data _null_;
  set print1b;
  array v (*) v: ;
  file raw1;
  if _n_=1 then
  do ;
    put
      'Health Plan'      '09'x
      'Specialty'        '09'x
      'Prov ID'          '09'x
      'Prov Name'        '09'x
      'Mean Age'         '09'x
      'Number of Users'  '09'x
      'Number of Claims' '09'x
      'Net'              '09'x
    @ ;
    do i = lbound (v) to hbound (v) ;
       call label ( v (i), label ) ;
       put label         '09'x @ ;
    end ;
    put ;
  end ;

  put hpcode            '09'x
      spec              '09'x
      provid            '09'x
      name              '09'x
      mage              '09'x
      elig              '09'x
      lines             '09'x
      net               '09'x
    @ ;
    do i = lbound (v) to hbound (v) ;
       put v ( i )       '09'x @ ;
    end ;
    put ;
run;
```

How did the proper variable labels get in the summary data set PRINT1B? How is the informat, VFMT created?

Here is the plan.

1. Build table PROCCODE of all possible Proc Codes and their labels.
2. Step to get top 30 Proc Codes as WANTED to merge with PROCCODE.
3. Producing FMTDATA for making an invalue format VFMT.
4. PROC SQL step to get labels into a macro variable.
5. PROC DATASETS to make labels.

The first step is a one-time affair to organize the missing control information. In part, I think it was the failure to provide this information that led to adhoc nature of the original program.

The first part of Step 2 has not been shown, since the original program included messy details, is quite adequate to obtain this information, and is not essential to understanding the problem. Assume that TOP30 holds the required codes and has the variables PROCCODE and RANK. To make the informat, one could use:

```
data fmtdata
     ( keep = proccode rank label
              type hlo fmtname
       rename = ( proccode = start )
     ) ;
  length proccode $ 6 label $ 30 ;
  retain type "I"
         fmtname "VFMT"
         hlo " " ;

  if eof then
  do ;
    hlo = "O" ;
    rank = 30 ;
    proccode = "" ;
    label = "Remainder"
    output ;
  end ;

  merge
```

```
    proccode (in = p
              keep = proccode label )
       top30 (in = w
              keep = proccode rank )
    end = eof
  ;
  by proccode ;
  if w ;
  if not p then
    error "Unexpected " proccode= ;
  output ;
run ;

proc format cntlin = fmtdata
   ( keep=start rank type hlo fmtname
     rename = ( rank = label ) ) ;
run ;
```

The code for step 4 is a simple SQL step tomake a macro variable for the LABEL statement.

```
proc sql ;
  select "v"||left(put(rank,2.))
            ||"='"||trim(label)||"'"
      into :lablist separated by " "
    from fmtdata
  ;
quit ;
```

It is then used in step 5 by PROC DATASETS to add the label information.

```
proc datasets lib = work ;
       modify print1b ;
          label &lablist ;
    quit ;
```

Conclusion:

SAS-L does provide a valuable for teaching SAS and getting SAS advice. This paper cannot begin to show the enormous wealth of topics and advice that SAS-L provides - first because I have restricted it to my interests and knowledge, and second because many respondents give their advice in private.

This paper has provided a very positive view o SAS-L, in part, because I have been very selective in the messages considered here. In general, you also must be prepared to be selective to get the most out of SAS-L. In particular, you must be able to recognize good solutions from bad ones.

However, SAS-L is amazingly self-correcting when messages are made public. Far too often the person asking for advice takes it in private and goes on his merry way. If one summarized what was learned and thanked the respondents, then one would often find that the best plans now become available because the group sees when poor advice is given.

I started by asking, can one learn by giving answers. I have not given evidence in this paper one way or another, but I have observed many individuals grow stronger with their participation in SAS-L.

Is SAS-L sufficiently recognized by your training and support people? I didn't really mean to answer this question, I really meant you to judge based on what you have learned in reading the problems and solutions that i have selected from SAS-L discussions.

The author may be contacted by mail at

Ian Whitlock
Westat
1650 Research Boulevard
Rockville, MD 20850

or by e-mail

whitloi1@westat.com

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.