Paper 236-26

Some Added Value: Using SAS® for Table Look-Ups, Merges, and Data Selection

David Pingel, Lake Park Consulting Corp., South Milwaukee, WI

ABSTRACT

I do reporting in the sales, marketing and accounting areas for a manufacturing company. Many of the reports need to show various levels of the dealer organization and/or product level information. Dealers are defined under divisions, regions, marketing managers and business managers. Product is defined in one structure to provide summary information to upper management and in another structure to provide detail information to product managers.

Dealer and product information are used so often I set up jobs that produce formats for dealer and product every day. I produce the formats in every combination that appears useful and store the formats in a permanent library. This information is now available with just a few statements in a data step or a FORMAT statement in a PROC step.

Over time I found that formats can be used to select data and replace merges in some cases. I discovered that these techniques work particularly well when processing large data sets since no sorts are required.

These are some of the reasons I use formats often. I am sure you can come up with many more ideas and expand on the uses presented in this paper.

The topics discussed in this paper are appropriate for SAS version 6 and greater on mainframe, Unix or Windows platforms.

INTRODUCTION

Points I try to cover in this paper:

- How to create temporary or permanent formats using DATA step programming and PROC steps.
- Ways the WHERE statement and the PUT function are used with formats for data selection, table look-ups a merge replacement.
- Ideas for managing format catalogs.

EXAMPLE OF THE FORMAT

This paper will use only character formats coded using the VALUE statement. The sample format below converts a dealer code to a dealer name. If the format does not contain a dealer code requested the result will be the value of OTHER. In this format OTHER returns "No Name for This Dealer".

```
Proc Format;
  VALUE  $DLRNAME
        '0111'='ACME Hardware Store'
        '0214'='Union Sports Equipment'
                  ...
        '7630'='End Of The Line Books'
        other='No Name for This Dealer';
```

VALUE formats are made up of a value or range of values and a formatted value. Data on the left side of the equal sign are called the range and data to the right of the equal sign are called the formatted value. The range can be either a single item or a true range. The range must be unique or the format will fail while being built.

BUILDING FORMATS WITH DATA STEPS

The procedure guide or help panel for PROC FORMAT shows a list of variable names that PROC FORMAT needs to build formats from data. There are more but these are the only ones I have ever needed. These variable names are:

FMTNAME	name of the format
TYPE	'C'=character 'N'=numeric (also 'P'=picture 'I'=numeric informat 'J'=character informat)
START	the unique or starting range value
END	(optional) the ending range value
LABEL	the label returned from format or informat
HLO	(optional) 'H'=high, 'L'=low and 'O'=other allows selecting values for HIGH, LOW or OTHER

This example of building a format by reading data shows a mechanical way of building the $DLRNAME format coded above. This example will process a SAS data set called DLRDATA using the two variables DEALER and NAME.

```
DEALER         NAME
0111           ACME Hardware Store
0214           Union Sports Equipment
               (other records)
7630           End Of The Line Books
```

Remember ranges must be unique and not overlap or the format will fail to load. One way for the range to have only unique values is to sort the data in 'range' sequence using the NODUPKEY option. If this method proves appropriate for the data you are using it is probably best that the sort be done before the data step that builds the format data.

```
PROC SORT DATA=DLRDATA NODUPKEY;
  BY DEALER;
```

Look how the data step below uses 5 variable names from the list above to build the format. Two required variables are FMTNAME and TYPE. A RETAIN statement is used to give values to those variables. They have the same value on every record. FMTNAME provides the name of the format (DLRNAME) and TYPE states the format is a character format. Since character format names begin with a '$' the real name of the format is '$DLRNAME'. The program reads each observation in DLRDATA. The value of DEALER is moved to the range variable START and the value of NAME is moved to the formatted value variable LABEL. The observation is output.

When all the input observations are processed the END option causes the variable EOF to be set. When end of file is detected 'O' is moved to HLO. The 'O' indicates that all the OTHER ranges not defined by the input data will be assigned the formatted value of 'No Name for This Dealer'. In the program below the variable names from the list above are shown as bold.

```
DATA DLRFMT(KEEP=FMTNAME TYPE START LABEL HLO);
  RETAIN FMTNAME 'DLRNAME'  TYPE 'C';
  SET DLRDATA END=EOF;
  START=DEALER;
  LABEL=NAME;
  OUTPUT;
  IF EOF;   /* Process only on last record */
  HLO='O';
  LABEL='NO NAME FOR THIS DEALER';
  OUTPUT;
RUN
```

Data set DLRFMT looks like this:

```
FMTNAME TYPE START  LABEL                      HLO
DLRNAME  C    0111   ACME Hardware Store
DLRNAME  C   0214   Union Sports Equipment
DLRNAME  C              (other records)
DLRNAME  C   7630   End Of The Line Books
DLRNAME  C   7630   NO NAME FOR THIS DEALER    O
```

(Note that the value of START on the last record is the same as the previous record but the HLO variable set to 'O' causes it to be ignored)

The CNTLIN= option is used by PROC FORMAT to point to the data set needed to build the format.

```
PROC FORMAT CNTLIN=DLRFMT;
RUN;
```

The message below indicates that the format was built successfully.

```
NOTE: Format $DLRNAME has been output.
```

This format is stored in a SAS catalog named WORK.FORMATS. Formats by default are output in the WORK data library. WORK is normally a temporary data library so the format will be deleted at the end of the program.

The LIBRARY= option is used to tell PROC FORMAT the name of the data library to output the format. Assuming a SAS data library named PERM is allocated, the code below will put the format $DLRNAME in a catalog named PERM.FORMATS.

```
PROC FORMAT CNTLIN=DLRFMT LIBRARY=PERM;
RUN;
```

The following message is placed in the log to show where the format is written.

```
THE FORMAT $DLRNAME HAS BEEN WRITTEN TO
PERM.FORMATS
```

A SAS program will not use the formats stored in the PERM.FORMATS catalog unless told to. An options statement with the FMTSEARCH= option defines the libraries to search when looking for formats. The library names must be listed in parenthesis.

```
OPTIONS FMTSEARCH=(PERM WORK);
```

The OPTIONS statement tells SAS to resolve formats by searching the libraries PERM and WORK. If the $DLRNAME format were found in both libraries the program would use the one in PERM because the statement told it to look there first.

THE PUT FUNCTION

The PUT function is the key to using formats in DATA steps and WHERE statements. The form of this function is

```
        PUT(variable,fmtname.)
```

An example of how this function works using the $DLRNAME format is:

```
        FLD1=PUT(DEALER,$DLRNAME.);
```

If the value of DEALER is '0111' then the value of FLD1 is "ACME Hardware Store". Notice that the format name begins with $ because it is a character format and ends with a '.' just the way a format is represented in a FORMAT statement.

DATA SELECTION EXAMPLES

Data selection is a "YES" or "NO" situation. This example uses a list of serial numbers to be selected from a different data set. The serial numbers in data set SERDATA are used to create a format named $SERFMT. This format is used in a data step or proc step to do the selection.

```
PROC SORT DATA=SERDATA NODUPKEY;
  BY SERIAL;
```

Note that the label produced is either 'YES' or 'NO'.

```
DATA FMT(KEEP=FMTNAME TYPE START LABEL HLO);
  RETAIN FMTNAME 'SERFMT' TYPE 'C' LABEL 'YES';
  SET SERDATA END=EOF;
  START=SERIAL;
  OUTPUT;
  IF EOF;
  START=' ';
  HLO='O';
  LABEL='NO';
  OUTPUT;
  RUN;

PROC FORMAT CNTLIN=FMT;
  RUN;
```

DATA SELECTION USING A DATA STEP

The data step below reads the SAS data set named MASTER.WARRANTY. This data set contains a variable named SERIAL. The new data set SELECTED will contain all observations with a serial number found in the format $SERFMT. Notice how the PUT function is used in the WHERE statement. The result of the PUT function is either 'YES' or 'NO'. This WHERE statement will select only the observations that resolve to 'YES'.

```
DATA SELECTED;
  SET  MASTER.WARRANTY;
  WHERE PUT(SERIAL,$SERFMT.)='YES';
  (other statements??)
```

DATA SELECTION USING PROC STEPS

PROC steps can use the same WHERE statement with a PUT function to select data. The result of the WHERE statement is dependant on the purpose of the proc.

The WHERE statement with PROC FSEDIT determines which observations are displayed on the terminal.

```
PROC FSEDIT DATA=MASTER.WARRANTY;
  WHERE PUT(SERIAL,$SERFMT.)='YES';
  RUN;
```

The WHERE statement with PROC PRINT determines which observations are included in the report.

```
PROC FSEDIT DATA=MASTER.WARRANTY;
  WHERE PUT(SERIAL,$SERFMT.)='YES';
  RUN;
```

The WHERE statement with PROC SORT determines which observations are included in the output data set. If the OUT= option is not used the MASTER.WARRANTY data set will contain only the observations containing serial numbers found in $SERFMT.

```
PROC SORT DATA=MASTER.WARRANTY OUT=SELECT;
  BY var1 varx;
  WHERE PUT(SERIAL,$SERFMT.)='YES';
  RUN;
```

RANGE VALUES USING MULTIPLE VARIABLES

In the examples so far the range value has been made up of a single variable. Sometimes data selection requires a combination of variables. Creating a range of concatenated values can solve that problem.

This example uses a data set named SELECTN which contains the variables BRANCH and PRODUCT. These codes will be used to create a format to select combinations of branch codes and product codes from MASTER.WARRANTY.

The length of BRANCH is $4 and the length of PRODUCT is $8. The program to prepare the format might look like this:

```
PROC SORT DATA=SELECTN NODUPKEYS;
  BY BRANCH PRODUCT;

DATA SELFMT;
```

```
  RETAIN FMTNAME 'SELFMT' TYPE 'C' LABEL 'YES';
  SET SELECTN END=EOF;
  START=BRANCH||PRODUCT;
  OUTPUT;
  IF EOF;
  HLO='O';
  LABEL='NO';
  OUTPUT;

PROC FORMAT CNTLIN=SELFMT;
  RUN;
```

The format will now be used to select data from MASTER.WARRANTY. The data step might look something like this. (Perhaps the sort could be used instead).

```
DATA TOREPORT;
  SET MASTER.WARRANTY;
  WHERE PUT(BRANCH||PRODUCT,$SELFMT.)='YES';
  (other statements??)
  RUN;
```

There are potential problems using concatenated variables. If the variable length of BRANCH in MASTER.WARRANTY is something other than $4 the WHERE statement will not work. There are ways to make sure the variable length is $4. One of those is to put a LENGTH statement after the DATA statement and before the SET statement:

```
DATA TOREPORT;
  LENGTH BRANCH $4;
  SET MASTER.WARRANTY;
    …;
```

Another way is to use the SUBSTR function within the PUT function:

```
WHERE PUT(SUBSTR(BRANCH,1,4)||PRODUCT,$SELFMT.);
```

TABLE LOOK-UP

The table look-up can be illustrated by looking at the $DLRNAME format created earlier. The PUT function will use the DEALER code and the $DLRNAME format to put the dealer name in the field DLRNAME.

```
DATA … ;
  SET  … ;
 DLRNAME=PUT(DEALER,$DLRNAME.);
```

TABLE LOOK-UP USING MULTIPLE FORMATTED VALUE VARIABLES

I receive many requests to prepare reports where the dealer code, dealer name, city and state are shown on the report. The $DLRNAME format produces the name. Formats called $DLRCITY and $DLRST can be created. $DLRCITY can associate dealer code with the dealer's city while $DLRST can associate the state with the dealer code. These three formats will pass the information to satisfy the request.

Instead of accessing three formats it may be worth while to establish one format to satisfy the request. This example will concatenate the dealer name, city and state and associate that with the dealer code. The building of the format could look like this:

```
PROC SORT DATA=DLRDATA NODUPKEY;
  BY DEALER;

DATA DLRFMT(KEEP=FMTNAME TYPE START LABEL HLO);
  LENGTH NAME CITY $20  STATE $2;
  RETAIN FMTNAME 'DLRINFO'  TYPE 'C';
  SET DLRDATA END=EOF;
  START=DEALER;
  LABEL=NAME||CITY||STATE;
  OUTPUT;
  IF EOF;
  HLO='O';
  LABEL='NO DEALER INFO       ????';
  OUTPUT;
```

```
PROC FORMAT CNTLIN=DLRFMT;
  RUN;
```

When building the format data set the program assures the length of NAME, CITY and STATE with the LENGTH statement. The three variables are then concatenated together to form the LABEL.

The data step below shows that the variable DLRINFO will contain a character string of data after the PUT function is performed. The SUBSTR function is used to convert the string of data into the variables DNAME, DCITY and DSTATE.

```
DATA …;
  SET …;
  DLRINFO=PUT(DEALER,$DLRINFO.);
  DNAME=SUBSTR(DLRINFO,1,20);
  DCITY=SUBSTR(DLRINFO,21,20);
  DSTATE=SUBSTR(DLRINFO,41);
  …
```

MERGE REPLACEMENT

Merge replace is similar to the table-lookup. The difference is that a merge has the ability to associate several pieces of data by matching several pieces of data.

DATA1 contains information that needs to be added to DATA2. These are the variables found in DATA1:

```
  STATE        $2     Alphabetic state abbrev.
  COUNTY       $3     Numeric county code
  STNAME       $20    State name
  CNTYNAME     $20    County name
  STFIPS       $2     State FIPS code
```

Data set DATA2 contains the variables STFIPS and COUNTY. If doing a merge these are the variables that would be used to associate the data.

This procedure can be used in place of a merge.

```
PROC SORT DATA=DATA1 NODUPKEY;
  BY STFIPS COUNTY;

DATA FMTDS;
  RETAIN FMTNAME 'STCNTY'  TYPE 'C';
  SET DATA1 END=EOF;
  START=STFIPS||COUNTY;
  LABEL=STATE||STNAME||CNTYNAME;
  OUTPUT;
  IF EOF;
  HLO='O';
  LABEL='??????               ???';
  OUTPUT;

PROC FORMAT CNTLIN=FMTDS;
```

The program that uses this format will probably contain code similar to the bold face statements:

```
DATA … ;
  SET DATA2;
        …  ;
  STINFO=PUT(STFIPS||COUNTY,$STCNTY.);
  STATE=SUBSTR(STINFO,1,2);
  STNAME=SUBSTR(STINFO,3,20);
  CNTYNAME=SUBSTR(STINFO,23);
        …  ;
```

DATA2 did not have to be sorted in any particular sequence to make this "merge" work. This technique is especially useful for large data sets.

FORMAT CATALOGS

PROC FORMAT has additional options that may prove useful when working with format catalogs.

- FMTLIB option prints the contents of the specified library.
- CNTLOUT= option specifies the name of an output data set that can be produced. This data set contains all the variables necessary to

reload the format(s).

- LIBRARY= option tells PROC FORMAT the library to access to perform the requested operation.
- INCLUDE statement tells PROC FORMAT the format names to include in an operation.
- EXCLUDE statement tells PROC FORMAT the format names not to include in an operation.

Examples:

```
PROC FORMAT    FMTLIB;
```
Will print the detail of all the formats in the catalog WORK.FORMATS.

```
PROC FORMAT  LIBRARY=PERM  FMTLIB;
```
Will print the detail of all the formats in the catalog PERM.FORMATS.

```
PROC FORMAT LIBRARY=PERM FMTLIB;
  INCLUDE $DLRNAME;
```
Will print the detail of the format $DLRNAME from PERM.FORMATS.

```
PROC FORMAT FMTLIB;
  EXCLUDE $DLRNAME;
```
Will print the detail of all the formats in the catalog WORK.FORMATS except $DLRNAME..

```
PROC FORMAT  LIBRARY=PERM  CNTLOUT=ALLFMTS;
```
Will put the values of all the formats found in PERM.FORMATS into the data set WORK.ALLFMTS.

A format can be modified and reloaded with the following code.

```
PROC FORMAT  CNTLOUT=MAINTFMT;
  INCLUDE $DLRFMT;
  RUN;

PROC FSEDIT DATA=MAINTFMT;
  RUN;

PROC FORMAT CNTLIN=MAINTFMT;
  RUN;
```

CONCLUSION

I have found many applications for using formats. I hope you have picked up an idea or two from this paper. Assuring that variable lengths are what you expect will solve the biggest problem with concatenated ranges or formatted values. Good luck and let me know if you come up with any good ideas using formats. Also let me know if you have problems. I will try to help.

REFERENCES

SAS Institute Inc. (1990) SAS® Procedures Guide, Version 6, Third Edition, Cary, NC:

CONTACT INFORMATION

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

David H. Pingel
Lake Park Consulting Corp.
902 18th Ave.
South Milwaukee, WI 53172
Dpingel@WI.RR.COM