

To FORMAT or MERGE That Is The Question

Caroline Bahler, AIC, Morrisville NC
Stephen Cios, ISSC, Durham NC

Introduction

One of the strengths of the SAS® system is that it provides many different options for joining data values from many different data structures. For the purpose of this paper, the term data structure will include flat or VSAM files (accessed using INFILE and INPUT STATEMENTS), database tables (DB2®, Oracle®, Rbase® etc. accessed using PROC SQL and ACCESS data views), and SAS data sets. Data structure joins can occur within the DATA step utilizing the MERGE statement, formats created using PROC FORMAT, or the SET statement with the KEY = option, or within PROC SQL. The types of joining methods available within the SAS system include match-merge using multiple SAS data sets (MERGE statement and SET statement with KEY = option), one to one matching using a "look-up" table (formats), one to one merge using multiple SAS data sets (MERGE statement). The objectives of this paper are to discuss each joining method and to compare and contrast those methods.

Formats

What are formats?

SAS has a facility (PROC FORMATS) that allows a user to define their own permanent or temporary formats'. A temporary format is defined and stored within the WORK directory, used within the program stream and deleted at the end of the program. A permanent format is defined and stored within a permanent SAS catalog for use in any program. PROC FORMAT allows the user to define a set of matching or "look-up" values and their corresponding labels or descriptions. These "look-up" values can represent a single key variable or multiple key variable values. In addition, the label can consist of single or multiple values'.

Example 1

A simple format could be structured as follows:

Look-up	Label
1	Red
2	Green
3	Blue

where both the look-up and label consist of a single values.

Example 2

A complex format could be structured as follows:

Look-up	Label
1	Red Cardinal
2	Green Parrot
3	Blue Indigo Bunting

where the look-up is a single value and the label consists of feather color and bird name;

or

Look-up	Label
1A	Red Cardinal
2B	Green Parrot
3C	Blue Indigo Bunting

where the lookup contains two (2) key values and the label consists of feather color and bird name. In all cases a "look-up" table is being created that can be used to transfer information into subsequent data sets.

Types of user defined formats - PROC FORMAT allows the creation of the following types of formats:

Informats -	Numeric Character
Formats -	Numeric Character Picture*

*A picture format is a template for printing numbers by defining leading zeros, decimals and comma punctuation, fill characters, prefixes etc.'.

Format creation -

User defined formats can be created by entering the values for the look-up and label values into PROC FORMAT or by reading an appropriately configured SAS data set.

PROC FORMAT - To create a format using PROC FORMAT the format name, look-up or key values and label values are entered using the following statements:

- INVALUE - creates both character and numeric informats
- VALUE - creates both character and numeric formats
- PICTURE - creates picture formats

To create a format containing the information from Example 1 the following statements would be used:

Example 3

A. The format created below is a numeric informat with the "look-up" values representing a specific color.

```
PROC FORMAT;
  INVALUE BIRD 1 = 'Red'
                2 = 'Green'
                3 = 'Blue' ;
```

B. The same format could be created as a numeric format by using a VALUE statement instead of an INVALUE statement. A picture format is created within example B and a range of values are labeled using a population description. The NOEDIT option is used to force PROC FORMAT to treat the label as a character value and print all information within the label as they appear'.

```
PROC FORMAT;
  PICTURE POP
    1-3='Population less than 10,000' (noedit)
    4-6='Population = 10,000' (noedit)
    7-9='Population greater than 10,000' (noedit);
```

CNTLIN Data sets - Formats can also be created from an appropriately configured SAS data set that is accessed by PROC FORMAT using the CNTLIN= option. The CNTLIN= option of PROC FORMAT reads in a SAS data set that has been configured so that it can be the input for a format. The SAS data set can contain data that was originally part of another SAS data set or from a database table through an ACCESS view or PROC SQL Pass-Through. To configure a SAS data set for input into PROC FORMAT the following variables need to be created:

Variable	Definition
START	consists of a single variable or concatenated variables that will be used as "look-up" variable values. These "look-up" values must be unique.
LABEL	consists of a single variable or concatenated variables that contain values that will be either displayed or added to the data set when the format is used.
FMTNAME	name of format.
TYPE	specifies the type of format being created and is defined by the data values within the START variable.

Other optional variables are available to further define the different format types.

To create a format containing the information from Example 2 data using the CNTLIN= option the following statements can be used:

Example 4

Data file EXAMPLE:

```
1 A Red      Cardinal
2 B Green    Parrot
3 C Blue     Indigo Bunting
9 X Blue     Blue Jay
```

```
DATA FORMAT(KEEP=START LABEL TYPE
  FMTNAME);
  INFILE EXAMPLE;
  INPUT COLOR $ 1 TYPE $ 3 COLORNME $ 5-12
        BIRD $ 13-27;
  START=COLOR||TYPE;
  LABEL=COLORNME||BIRD;
  FMTNAME='BIRD';
  TYPE='C';
  CARDS;
  RUN;
```

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

In the preceding example the bolded variable definitions are required to produce a data set that can be used by PROC FORMAT to create a format. The start and label variables were created by concatenating two (2) variables together, with the format type being defined as a character format.

Once the format is created the information encompassed within the format definition is stored within a compiled entity that has a binary structure. The values that are part of a format definition are stored using a variable structure so that each individual value gets established at the length of the last non-blank position within the label value. For example, the label values contained within Example 4 will range in length from fourteen (14) to twenty-two (22) positions. Retrieval of this information is through the use of binary searches'.

Note: SAS does not consider the values stored within a format as data. However, data values defined by a data set or within PROC FORMAT statements are part of the format definition'.

Programmatic Usage of Formats

Programmatic usage of a format depends upon the type of format created and the needs within a given program. Formats can be used to either display information when a data set is printed or to create a new variable.

Example 5 -

Using data sets from Table 1

A. Format Creation with Default

```
DATA FORMAT;
  SET TEST3 END=EOF;
  HLO=' ';
  RSP5A=PUT(RSP5,Z8.2);❶
  RSP6A=PUT(RSP6,Z8.2);
  START=VAR1||VAR2;
  LABEL=RSP5A||RSP6A;
  FMTNAME='TESTA';
  TYPE='C';
  OUTPUT FORMAT;
  IF EOF THEN DO;
    HLO='O';
    START='Other';
    LABEL=' ';
    OUTPUT FORMAT;
  END;
  RUN;
```

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

```
DATA ALL;
  SET TEST1;
  RSP=PUT(VAR1||VAR2,$TESTA.);
  RSP5=INPUT(SUBSTR(RSP,1,8),8.2);❷
  RSP6=INPUT(SUBSTR(RSP,9),8.2);
  RUN;
```

VAR1	VAR2	RSP1	RSP2	RSP5	RSP6
1	A	10	3.2	10.7	13.2
2	A	13	3.5	13.2	13.5
2	B	12	2.9	11.9	23.1
1	B	11	3.1	12.3	12.9
2	C	15	3.0	9.8	12.8
2	C	9	2.8	9.8	12.8
2	D	11	3.6	10.1	9.4
1	E	10	3.4	.	.
1	F	12	3.3	.	.

Using a format to join data from data set TEST1 with TEST3 results in the above data set. This data set does not contain all of the values within both data sets.

In Example 5A, by defining the HLO variables as 'O' and adding that observation to the end of the data set used to create the \$TESTA format a default value for the format was created.

In the above example multiple formats were used as follows:

- ❶ The first instance is in the creation of the TESTA format where numeric variables were converted to character variables.

- ❷ In the second instance once \$TESTA was used the new variable was used to create the two (2) response variables (5 and 6) which were converted to numeric variables.

B. Format Creation without Default

Format created without HLO variable defined as in Example 5A.

```
DATA FORMAT;
  SET TEST2;
  HLO=' ';
  START=VAR1||VAR2;
  LABEL=RSP3||RSP4;
  FMTNAME='TESTB';
  TYPE='C';
```

```
PROC FORMAT  CNTLIN=FORMAT;
```

```
DATA ALL;
  SET TEST1;
  RSP=PUT(VAR1||VAR2,$TESTB.);
  RSP3=INPUT(SUBSTR(RSP,1,8),8.2);
  RSP4=INPUT(SUBSTR(RSP,9),8.2);
  RUN;
```

VAR1	VAR2	RSP1	RSP2	RSP3	RSP4
1	A	10	3.2	100	23.2
2	A	13	3.5	120	22.9
2	B	12	2.9	2B	.
1	B	11	3.1	110	23.1
2	C	15	3.0	90	22.8
2	C	9	2.8	90	22.8
2	D	11	3.6	150	23.0
1	E	10	3.4	110	23.6
1	F	12	3.3	1F	.

The result of not having a default value within a format is illustrated above. RSP3 has the concatenated value of VAR1 and VAR2 as it's value in two (2) cases. SAS will issue error messages when this occurs since RSP3 is a numeric variable. There are some cases that defaulting to the value of the "matching" variable is appropriate. However the user needs to evaluate each situation where a format is used and determine whether a default is required.

What is the "cost" of using a format?

A format only matches information with data values occurring on the "base" data set. If information occurs within the format that does not occur within the "base" data set then that information it will not be transmitted to the new data set being created. Therefore, if information within a format is critical and corresponding "look-up" values do not reside on the "base" data set then a format should not be used.

Resource consumption is another cost of using a format. A format's resource utilization can be divided into three (3) sections. The first section consists of the format creation where memory, CPU and work space are utilized. The second section consists of format usage. When using formats the entire format is loaded into memory this will increase the amount of memory required for the program. The memory usage is directly related to the size of the format being loaded². The third section consists of organization and accessing permanent formats. Using multiple libraries to store permanent formats requires the use of the FMTSEARCH option. This increases the memory cost since buffers must be loaded for these format libraries³. To reduce this cost consider the use of a central format library where the libref LIBRARY can be assigned. This will automatically cause SAS to search the format catalogs within WORK, LIBRARY and FORMAT librefs for formats¹.

Merge

What is a merge?

A merge is a DATA STEP statement that combines two (2) or more SAS data sets using the following strategies - one-to-one merging or match-merging. One-to-one merging combines all of the SAS data sets listed within the MERGE STATEMENT using a common variable and creates a data set that is as large as the largest data set within the merge list⁴. The restrictions for using a one-to-one merge are that this type of merge does not match data values of common variables instead it simply places each data set side by side and it replaces data values of the common variables of all other data sets with the data values of the last data set within the merge list. Using this type of merge each common variable value is super-imposed by the values within the last data set in the merge. Thereby causing data values which should have been retained to be left out (see Example 6A).

Match-merging combines all of the data sets listed by the data values within the common variables. A BY statement lists the common variables to merge upon and all data sets must be sorted by the common variables in the same sort order. An exception to the pre-sorting required data sets would be that are indexed by the BY variable(s) to be used in the merge⁵. All data values within each data set listed is kept unless specified by the programmer, therefore duplicate data values of the common variables are protected.

Programmatic usage of Merge

Programmatic usage of the MERGE statement depends upon the type of merge being performed. The following example can be used to perform a one-to-one merge and a match-merge:

Example 6

Using the data sets from Table 1 -

A) One-to-one merge -

```
DATA TEST;
  MERGE TEST1 TEST2
  TEST3;
  RUN;
```

Results of Merge:

VAR1	VAR2	RSP1	RSP2	RSP3	RSP4	RSP5	RSP6
1	A	10	3.2	100	23.2	10.7	13.2
2	A	13	3.5	130	23.5	13.2	13.5
1	B	12	2.9	120	22.9	12.3	12.9
2	B	11	3.1	110	23.1	11.9	23.1
1	C	15	3.0	150	23.0	15.3	13.0
2	C	9	2.8	90	22.8	9.8	12.8
1	D	11	3.6	110	23.6	11.1	13.6
2	D	10	3.4	100	23.4	10.1	9.4
2	F	12	3.3	120	23.3	12.4	23.3

Example 6A results in a merge where all data values of the common variables are from data set TEST3 and the data sets are placed physically side-by-side starting at observation 1 within each data set. There is no matching of data values within the common variables and the resulting length of the new data set is equal to the longest data set within the merge list. In addition, information was lost because the values of VAR1 and VAR2 within data sets TEST1, TEST2 and TEST3 were not identical and in the same order.

B) Match-merge -

```
DATA TEST;
  MERGE TEST1 TEST2
  TEST3;
  BY VAR1 VAR2;
  RUN;
```

Results of Merge:

VAR1	VAR2	RSP1	RSP2	RSP3	RSP4	RSP5	RSP6
1	A	10	3.2	100	23.2	10.7	13.2
1	B	11	3.1	110	23.1	12.3	12.9
1	C	.	.	.	.	15.3	13.0
1	D	.	.	.	.	11.1	13.6
1	E	10	3.4	100	23.4	.	.
1	F	12	3.3	.	.	.	.
1	G	.	.	120	23.3	.	.
2	A	13	3.5	120	22.9	13.2	13.5
2	B	12	2.9	130	23.5	11.9	23.1
2	C	15	3.0	90	22.8	9.8	12.8
2	C	9	2.8	90	22.8	9.8	12.8
2	D	11	3.6	150	23.0	10.1	9.4
2	E	.	.	110	23.6	.	.
2	F	.	.	.	.	12.4	23.3

Example 6 B results in a merge where all data values of the common variables are matched with one another. All data values of the common variables across data sets were retained within the created data set. Missing data values occurred because data sets TEST1, TEST2 and TEST3 did not contain the same data values for the common variables. A variation on the same code uses

the IN= option with each MERGE statement data set. This allows the programmer to default the response variables to 0 that do not have common variable data values.

```
DATA TEST;
MERGE TEST1(IN=A) TEST2(IN=B) TEST3(IN=3);
BY VAR1 VAR2;
IF NOT A THEN DO;
  RSP1=0;
  RSP2=0;
  END;
IF NOT B THEN DO;
  RSP3=0;
  RSP4=0;
  END;
IF NOT C THEN DO;
  RSP5=0;
  RSP6=0;
  END;
RUN;
```

Results of Merge:

VAR1	VAR2	RSP1	RSP2	RSP3	RSP4	RSP5	RSP6
1	A	10	3.2	100	23.2	10.7	13.2
1	B	11	3.1	110	23.1	12.3	12.9
1	C	0	0.0	0	0.0	15.3	13.0
1	D	0	0.0	0	0.0	11.1	13.6
1	E	10	3.4	100	23.4	0.0	0.0
1	F	12	3.3	0	0.0	0.0	0.0
1	G	0	0.0	120	23.3	0.0	0.0
2	A	13	3.5	120	22.9	13.2	13.5
2	B	12	2.9	130	23.5	11.9	23.1
2	C	15	3.0	90	22.8	9.8	12.8
2	C	9	2.8	90	22.8	9.8	12.8
2	D	11	3.6	150	23.0	10.1	9.4
2	E	0	0.0	110	23.6	0.0	0.0
2	F	0	0.0	0	0.0	12.4	23.3

What is the "cost" of using a MERGE statement?

Resource utilization by a MERGE statement is affected by the size of the data sets involved and if those datasets are indexed within the by variable(s). The two (2) resources impacted by the MERGE statement are DASD and CPU. SAS work space availability can be critical, especially with large data sets. Work space is needed for sorting the data sets involved and the data set resulting from the merge.

Conclusions

Comparison of a format to a match-merge using the MERGE statement results in the following conclusions (Table 2):

- Number of data sets joined at one time - a format can join two (2) data sets at once using a common "look-up" or key value. The MERGE and SET statements allow two (2) or more data sets to be joined.
- Data retention - a format result in a new data set that contains only those data values found within the "base" data set. In contrast, the match-merge retains all data values from all data sets merged unless specifically indicated by the use of an IF or WHERE statement.

In conclusion, be sure to benchmark and establish usage according to resources available within your location.

References

1. SAS Institute Inc. (1990), SAS Procedures Guide, Version 6, Third Edition. Cary, NC: SAS Institute Inc., 275-312.
2. Personal Communication from SAS.
3. SAS Institute Inc. (1991), SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07. Cary, NC: SAS Institute Inc., 91, 207-217.
4. SAS Institute Inc. (1990), SAS Language: Reference, Version 6, First Edition. Cary, NC: SAS Institute Inc., 147-155.

SAS is a registered trademark or trademark 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.

Table 1. Example Data Sets.

TEST1

VAR1	VAR2	RSP1	RSP2
1	A	10	3.2
2	A	13	3.5
2	B	12	2.9
1	B	11	3.1
2	C	15	3.0
2	C	9	2.8
2	D	11	3.6
1	E	10	3.4
1	F	12	3.3

TEST2

VAR1	VAR2	RSP3	RSP4
1	A	100	23.2
2	B	130	23.5
2	A	120	22.9
1	B	110	23.1
2	D	150	23.0
2	C	90	22.8
2	E	110	23.6
1	E	100	23.4
1	G	120	23.3

TEST3

VAR1	VAR2	RSP5	RSP6
1	A	10.7	13.2
2	A	13.2	13.5
1	B	12.3	12.9
2	B	11.9	23.1
1	C	15.3	13.0
2	C	9.8	12.8
1	D	11.1	13.6
2	D	10.1	9.4
2	F	12.4	23.3

Table 2. Comparison of Formats and Merges

Usage	Format	Merge
Programmatic Usage	DATA step	DATA step
Data Retention	Base data set	All data sets
Number of Data Sets	Two (2)	Multiple
Resource Consumption	Memory	DASD, CPU, EXCP