Formats: An Advanced Tutorial

Richard D. Langston, SAS Institute Inc.

ABSTRACT

This tutorial discusses several advanced features of format processing that can be of interest even to knowledgable users of the SAS System®. Items covered include: using formats as labels, cntlout and cntlin data sets, gaining efficiency with the the NOTSORTED option, using a format as an alternative to sorting large data sets, and comparing V5 and V6 formats.

INTRODUCTION

The FORMAT procedure is a powerful tool for building customized formats to handle advanced processing needs. In this tutorial, we discuss some of the newer features of the FORMAT procedure that may not be known even to advanced users.

USING FORMATS AS LABELS

A new feature in PROC FORMAT was introduced in Release 6.07 of the SAS System on Mainframes. (The feature appeared for the first time on UNIX platforms in Release 6.09). This features allows for you to specify a format name as the label of another format. This in turn allows you to logically chain formats together.

A format-as-label is denoted by an unquoted format name in square brackets (or (| |) as a replacement for left and right brackets). For example:

```
proc format;
     value cats   1- 50=[fruit20.]
                 51-100=[veg20.]
                101-150=[dairy20.]
                151-200=[meat20.];
     value fruit  1='banana'
                  2='apple'
                  3='pear';
     value veg   51='broccoli'
                 52='lima beans'
                 53='peas';
     value dairy 101='skim milk'
                 102='cheese'
                 103='yogurt';
     value meat  151='ground beef'
                 152='pork chops'
                 153='sirloin';
     run;

data _null_;
     input foodcode @@;
     put foodcode=cats.; cards;
1 53 103 152
run;
```

In the above example, the CATS format is used to handle food categories. In order to allow for separate maintenance for the different food groups, each food group has its own format, and that format name is given as the label. When the DATA step is run, the following lines will be produced:

```
FOODCODE=banana
FOODCODE=peas
FOODCODE=yogurt
FOODCODE=pork chops
```

When the value 1 is to be formatted, it is matched in the first range, whose label is the format FRUIT. The FRUIT format is then invoked with the value 1, which will then be formatted as 'banana'. It would have been acceptable for the FRUIT format to have continued subdividing with more format labels, such as TROPICAL and DOMESTIC as more specific formats.

It is important that your formats do not loop back to a previously referenced format. The formatting subsystem does not permit this, in order to avoid endless looping.

THE CNTLIN= AND CNTLOUT= DATA SET

The CNTLIN= data set can be specified as an input data set to PROC FORMAT. It is an alternative method of providing format definitions to PROC FORMAT.

The complete list of variables that PROC FORMAT uses in the CNTLIN= data set is as follows:

```
FMTNAME     Char   Format name
START       Char*  Starting value for format
END         Char*  Ending value for format
LABEL       Char*  Format value label
MIN         Num    Minimum length
MAX         Num    Maximum length
DEFAULT     Num    Default length
LENGTH      Num    Format length
FUZZ        Num    Fuzz value
PREFIX      Char   Prefix characters
MULT        Num    Multiplier
FILL        Char   Fill character
NOEDIT      Num    Is picture string noedit?
TYPE        Char   Type of format
SEXCL       Char   Start exclusion
EEXCL       Char   End exclusion
HLO         Char   Additional information

* can be numeric in certain cases
```

Note that you can have any other variables that you want, but they will simply be ignored by PROC FORMAT.

Note also that you do not have to provide all of the above variables. The three variables that must be provided are FMTNAME, START, and LABEL.

PROC FORMAT can also be used to create a CNTLOUT= data set that corresponds to the format catalog being processed. You can use this output data set to create a codebook, to postprocess for reading back into PROC FORMAT at a later point, to port to another platform, or for any other reason that you may deem useful.

THE NOTSORTED OPTION

Also new in Release 6.07 for PROC FORMAT is the NOTSORTED option. Before explaining this option, a bit of in-depth discussion on PROC FORMAT logic is in order.

When you specify a group of ranges in a PROC FORMAT step (either in the VALUE/INVALUE/PICTURE statements or via the CNTLIN= data set), those ranges are sorted in memory so that when they are finally written to a permanent format they are written in sorted order. This allows for the format to be subsequently loaded in sorted order so that more efficient binary

tree searching routines can be used to find matching ranges.

There are at least two circumstances in which you do not want format processing to create a binary tree when the format is loaded. One situation occurs when you want to arrange your ranges in a prescribed order that indicates the likelihood of the occurrence of the range. That is, the earlier in the list of ranges it appears, the more likely a match will be found in the range based on the characteristics of the data. For example, suppose your format has the following ranges:

```
18-34='young adult'
40-64='middle age'
0-17='child'
65-high='senior'
```

and the data involves ages of parents of young children. You may decide that the above order ranks the likelihood of occurrence of range matching. If you use the NOTSORTED option, you may find that the range matching will be faster. Of course, with such a small number of ranges, there won't be much of a performance difference, but differences could be more substantial on a larger scale.

The other circumstance in which NOTSORTED is necessary is when you are porting an informat or character format from an EBCDIC to an ASCII platform (or vice versa). The reason this process is special is because the collating sequence for EBCDIC and ASCII are different. For example, consider the strings '123', 'abc', and 'XYZ'. On an EBCDIC system, the sorted order for those strings is 'abc', 'XYZ', and '123'. On an ASCII system, the sorted order is '123', 'XYZ', and 'abc'. If the format from this PROC FORMAT step is saved, the order of the ranges stored will be different between an ASCII and EBCDIC system:

```
proc format;
     value $afmt '123'='string 1'
                 'abc'='string 2'
                 'XYZ'='string 3';
     run;
```

The fact that the order is different is not really a problem until you try to transport the format (via PROC CPORT and CIMPORT) to another platform with a different collating sequence. What PROC CIMPORT does to prevent problems is to automatically mark such imported formats as NOTSORTED.

FORMATS VS. SORTING

Suppose you have a situation in which you want to add special codes to your data set based on values in the observations. Suppose the key variable is ACCOUNT and the code variables are TYPE, CREDIT, and BRANCH:

```
data acctinfo;
     input account accttype $ credit branch $;
     cards;
11111  P 500 mall
22222  P 300 downtown
33333  B 1000 lake
44444  P 700  mall
run;

data transact;
     input account amount; cards;
11111 150
33333 -200
44444 50
11111 230
run;
```

To add the variables TYPE, CREDIT, and BRANCH to the TRANSACT data set, you can do the following:

```
proc sort data=transact; by account;
data transact;
     merge transact(in=want) acctinfo;
     by account;
     run;
```

This method is fine except for at least the following reasons: 1) if TRANSACT is a large data set, you incur additional overhead for the sort; 2) if you need to preserve the order of the TRANSACT data set, you will have to sort back again, possibly creating a new variable to indicate the sort order.

Another approach is to use formats instead:

```
data cntlin; set acctinfo;
     fmtname='ACCTINFO';
     start=account;
     label=trim(accttype)||' ,'
           ||trim(left(put(credit,best12.)))
           ||' ,'||branch;
     run;
proc format cntlin=cntlin;
     run;
data transact;
     input account amount;
     acctinfo=put(account,acctinfo.);
     branch = scan(acctinfo,2,',');
     cards;
<same data as above>
run;
proc print;
```

Of course, the ACCTINFO format could instead be broken up into several different formats, but it's probably just as simple to have all the values appear in one range. You can use the SCAN function to extract any portion of the label. Note that when the LABEL variable was created, a blank is added before each comma. This allows for a blank value to be extracted when a missing value is encountered. Note that if no blank appeared between the commas, the SCAN function would not recognize the token count. For example:

```
X = SCAN('abc,,xyz',2,',');
Y = SCAN('abc, ,xyz',2,',');
```

X will be set to 'xyz' but Y will be set to blanks.

The final PROC PRINT from the above example produces the following output:

OBS	ACCOUNT	AMOUNT	ACCTINFO	BRAN
1	11111	150	P ,500 ,mall	mal
2	33333	-200	B ,1000 ,lake	lak
3	44444	50	P ,700 ,mall	mal
4	11111	230	P ,500 ,mall	mal

VERSION 5 VS. VERSION 6 FORMATS

In Version 5 of the SAS System for IBM mainframes, formats generated by PROC FORMAT were stored as small executable programs. In Version 6 of the SAS System for all platforms, formats generated by PROC FORMAT are stored as members of a format catalog. This allows for simpler porting, simpler reporting

on format contents, and the avoidance of code generation issues.

However, IBM mainframe formats from Version 5 can still be used under Version 6. They are referenced in the usual fashion and are searched for via the SASLIB DD statement or within the STEPLIB concatenation.

These formats can be converted to Version 6 representation by using the V5TOV6 procedure.

The Version 5 formats are actually small assembler program stubs followed by the range data and then by the label data. There is a different stub for numeric formats, numeric informats, character formats, character informats, and picture formats. There is a limit of 16 characters for any character range value, and 40 characters for any label. The stub uses a binary search technique to find matching ranges. All labels are stored with the same length, so the length of the longest label will be used as the length for all labels, using blank padding when necessary. All ranges have a start and end value stored, regardless of whether the start and end values are actually the same.

By comparison, Version 6 formats consist of tables of ranges and labels stored in a catalog. There is no executable code stored with the format, and the common code used to traverse the tables resides within the SAS Supervisor. The limit on ranges and labels is 200 characters instead of 16 or 40. Ranges and labels are stored such that only the necessary number of characters are saved. If a start and end value for a range is the same value, only a single value is stored.

Consider the storage differences for the following:

```
proc format;
     value 1='yes' 2='no'
           3='possibly yes' 4='possibly no';
     run;
```

Version 5 storage:

```
<assembler code for numeric format>
(ranges)
1-1: label 1
2-2: label 2
3-3: label 3
4-4: label 4
(labels)
(1) 'yes         '
(2) 'no          '
(3) 'possibly yes'
(4) 'possibly no '
```

Version 6 storage:

```
(header info)
(ranges)
1: label 1
2: label 2
3: label 3
4: label 4
(labels)
(1) 'yes'
(2) 'no'
(3) 'possibly yes'
(4) 'possibly no'
```

As new features have been added to PROC FORMAT, these features are not being added to the capability of generating Version-5-style formats. Larger label strings, larger range strings, the FMTSEARCH= option, FMTLIB output, and the NOTSORTED option are examples of extensions that do not apply to Version-5-style formats. You should consider the execution of Version-5-style formats to be a migration aid as you convert these formats to Version-6-style.

FMTSEARCH= OPTION

The FMTSEARCH= option allows you to search one or more format catalogs to find a specific format. Each format is searched for separately. This allows you to separate your formats into logical groups. Consider the following example:

```
libname employee '/dept/employee';
libname accting '/dept/accting';
libname software '/dept/mis/software';
options fmtsearch=(work.formats library.formats
                   employee.formats accting.formats
                   software.formats);
```

Assume that the following formats reside in the libraries:

```
employee.formats: empnum, deptname
accting.formats:  acctcode, deptname, info
software.formats: vendor, softcode
```

Note that since DEPTNAME appears in both employee.formats and accting.formats, the format in employee.formats will be picked up first because employee.formats appears before accting.formats in the FMTSEARCH= list.

Because work.formats comes before the other format catalogs in the FMTSEARCH= list, you can create a version of a format in that catalog that will override the one in another catalog in the list. For example, an updated version of ACCTCODE could be placed in work.formats. With the FMTSEARCH= option given above, the ACCTCODE format will be found in work.formats before it would be found in accting.formats.

The catalogs listed in the FMTSEARCH= option do not have to exist at the time a format is being searched for. For example, suppose employee.formats does not yet exist, and you have the following SAS code that follows the OPTIONS FMTSEARCH= from above:

```
data _null_;
     x=1; put x vendor.;
     run;
proc format lib=employee.formats;
     value yesno 1='yes' 2='no';
     run;
data _null_;
     x=1; put x=yesno.;
     run;
```

In the example above, the first DATA step will look for the VENDOR foramt in work.formats, library.formats, employee.formats, and accting.formats before finding it in software.formats. There will be no error message because employee.format does not yet exist.

Note how the format catalog can be referenced with two-level names via the LIB= option. If the catalog does not exist when being written to by PROC FORMAT, it will be created.

SUMMARY

The FORMAT procedure provides many mechanisms to allow for the reformatting of data and output. The features described herein

allow for you to make better use of the FORMAT procedure within a production environment by providing features to avoid sorting and to allow for searching among compartmentalized catalogs.

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