

New Tricks for an Old Tool: Using Custom Formats for Data Validation and Program Efficiency

S. David Riba, JADE Tech, Inc., Clearwater, FL

ABSTRACT

PROC FORMAT is one of the old standards among SAS® Procedures, yet many programmers often fail to understand how custom formats can be used to improve their SAS programs.

This paper will examine some ways user defined custom formats can be used for data validation, for developing more flexible SAS program code, and for other programming efficiency techniques. It will illustrate how the SAS programmer can avoid hard-coding conditional logic chains in a program, how data can be grouped for analysis, how the appearance of report output can be enhanced, and how SAS data can be recoded for more efficient storage. The paper will also review recent enhancements to PROC FORMAT such as Control data sets, format reports, and the ability to define and use format libraries.

INTRODUCTION

```
IF ( VAR1 EQ 'CONDITION1' )
  THEN VAR2 = 'RESULT1';
ELSE IF (VAR1 EQ 'CONDITION2' )
  THEN VAR2 = 'RESULT2';
ELSE IF (VAR1 EQ 'CONDITION3' )
  THEN VAR2 = 'RESULT3';
ELSE VAR2 = 'RESULT4';
```

Does the above SAS code look familiar? It should. Many SAS programmers, and most beginning SAS programmers, use code like this to test and assign values of variables based on some form of condition. Many SAS programs have long, complicated, chains of conditional logic. Often the same logic is repeated more than once in the same SAS job in different steps. Maintenance of these logic chains becomes a source of problems. Any change to the logic (additional tests, revisions to the logic, etc.) results in code changes that must be revised everywhere in the source code.

How many times have you written a SAS program where you have included a DATA step solely to select observations to include in a SAS procedure?

There is a tool in the toolkit of every SAS user to help manage these problems. The tool is PROC FORMAT, which is part of Base SAS®. Have you ever considered using PROC FORMAT for more than just nice looking output? Most people have not, even though there are many uses for this procedure

The FORMAT statement is used to assign a format, either SAS-supplied or user-defined, to a SAS variable. Once a format is assigned to a variable, the formatted value of the variable will be used whenever it is appropriate. Typically, these format definitions are used to enhance the appearance of their printed reports.

PROC FORMAT is used to define custom formats, which has obvious benefits when creating reports. What is less obvious is that, combined with the PUT and INPUT functions, custom formats can be used for data validation and modification.

The PUT and INPUT functions operate similar to the PUT and INPUT statements:

```
TESTDATE = INPUT('012693',MMDDYY6. );
```

converts a character string (012693) into a SAS date value and assigns it to variable TESTDATE.

A basic rule of thumb is to use the PUT function to apply a format to a variable, and the INPUT function to apply an informat to a variable. Both functions take a value, either a variable, a number, or a text string, and "write" the value with the format or informat specified by the second argument.

So how can the PUT and INPUT functions combined with PROC FORMAT simplify your SAS code and improve your program's maintainability?

ASSIGNING VALUES WITH FORMATS

The example that started this paper could have been written as a single statement using a custom format. Assuming your SAS program has defined a custom format (MY_FMT.), this example could have been rewritten as:

```
VAR2 = PUT(VAR1,MY_FMT.);
```

This is considerably less code, and easier to maintain. Any additional tests or any changes in values can be handled with one change to the custom format MY_FMT.

To create a custom format, you need to use PROC FORMAT with either a VALUE, PICTURE, or INVALUE statement. The VALUE statement defines a character or numeric format, PICTURE defines the appearance of numeric variables, and INVALUE defines input formats.

For example:

```
PROC FORMAT ;
  VALUE sexfmt      1 = 'MALE'
                   2 = 'FEMALE' ;
```

The VALUE statement defines a numeric format called SEXFMT. When using this format variables with a value of 1 will be given the label MALE, while those with a value of 2 will be given the label FEMALE.

The following example defines a character format called \$GENDER, which assigns the letter M the value of MALE and F the value of FEMALE:

```
PROC FORMAT ;
  VALUE $gender      'M' = 'MALE'
                   'F' = 'FEMALE' ;
```

Where it is not otherwise specified in each of the examples in this paper, you can assume that a format was first created using either the VALUE or INVALUE statement with PROC FORMAT.

You can create custom formats that are either permanent or temporary. To create a permanent format, use the LIBRARY = option.

```
LIBNAME LIBRARY '_____' ;

PROC FORMAT LIBRARY = library ;
```

This will store any formats you create in the library defined with the LIBRARY libref. They can then be used in subsequent SAS sessions and by other SAS users who have access to that library. Permanently stored formats are stored in your library in SAS catalogs. These catalogs have a second-level SAS name of FORMATS. If you check your WORK library, you will find a FORMATS catalog automatically created for you anytime you define a temporary format.

With permanent formats, the same logic can be used in a number of programs with only a single format table to maintain. An example of this might be a Chart of Accounts, where every account number has a descriptive name. Most accounting programs duplicate the logic of assigning names to each account in every program (and then require revisions to every program each time there is a new account or revised account name). With a permanent format, the individual programs never have to be modified.

A statement such as

```
DESCRIP = PUT(ACCT_NO, ACCT.) ;
```

will always assign the current value for each account number to a variable called DESCRIP, eliminating all of those IF... THEN ... logic chains.

The same principle applies to numeric calculations. How many times have you written code like:

```
IF(VAR EQ 1)
  THEN FACTOR = 90 ;
ELSE IF(VAR EQ 2)
  THEN FACTOR = 80 ;
ELSE ...
```

A format table would provide the same results, be easier to maintain, and make your program code more readable.

Compare the above code with the simpler

```
FACTOR = PUT( VAR, FACTFMT. ) ;
```

DATA VALIDATION WITH FORMATS

Another use of custom formats is for data validation. Testing variables with multiple valid options usually requires some form of conditional logic such as:

```
IF(ANSWER EQ 1) | (ANSWER EQ 2)
                | (ANSWER EQ 3)
  THEN RESULT = 'RIGHT' ;
ELSE RESULT = 'WRONG' ;
```

Consider a PROC FORMAT statement such as:

```
PROC FORMAT ;
  VALUE TEST1 1,2,3 = 'RIGHT'
          OTHER = 'WRONG' ;
```

Now the above program code could be written

```
RESULT = PUT( ANSWER, TEST1. );
```

which assigns the value of RIGHT or WRONG to the variable RESULT, depending on the value of the variable ANSWER.

Suppose you needed to determine if the value of a variable was one of a group of acceptable values, when the list of acceptable values might change on a regular basis. Coding the values in your SAS program means that you will need to modify your program every time the list of values changes. Typically, you might have the test of acceptable values in multiple DATA steps, requiring that each DATA step has to be recoded.

Using a custom format, the list of acceptable values can be kept in a format table. One change to the format table makes the current list of acceptable values available throughout your SAS program with no further programming required.

Instead of:

```
IF (ITEM IN('A105', 'B110', 'B250',
            'C301', 'C425') THEN DO ;
```

which requires that your SAS program be modified every time the item list changes, consider:

```
PROC FORMAT ;
    VALUE $PAYITM 'A105' 'B110'
                  'B250' 'C301' 'C425' = '1'
                  other = '0'
RUN ;

DATA ;

    IF (PUT(ITEM,$PAYITM.) EQ '1') ;
```

In this example, only the format needs to be changed to make the validation effective throughout the SAS program. No other coding is required.

SUBSETTING DATA WITH FORMATS

Since this subsetting IF is really a TRUE - FALSE test, the statement could be simplified as:

```
IF (PUT(ITEM,$PAYITM.)) ;
```

Custom formats are ideal for TRUE - FALSE tests. All IF tests in SAS resolve into a simple TRUE or FALSE result (1 or 0). For example, a test such as IF (VALUE EQ 1) resolves to one of only two

results - a 1 (TRUE) if the variable VALUE equals the number 1, or a 0 (FALSE) for any other result.

Suppose you have a large number of values to test for, and only want to process a selected subset. This example is a business Chart of Accounts where only Balance Sheet accounts are to be included in a report.

The usual way of handling this situation is a DATA step with conditional logic something like this:

```
DATA ;

    IF (ACCT EQ '123') | (ACCT EQ '125') |
        (ACCT EQ '126') | (ACCT EQ '190') |
        (ACCT EQ '210') ...
```

ad infinitum.

Contrast that with the following:

```
PROC FORMAT;
    VALUE BALSHEET
        '123','125','126' = '1'
        '190' = '1'
        '210' - '230' = '1'
        '300' - '399' = '1'
        OTHER = '0';
```

```
DATA ;

    IF (PUT(ACCT, BALSHEET.)) ;
```

The result is that only accounts with a formatted value of 1 are included, since they evaluate as TRUE, while all other accounts evaluate as FALSE and are not included. Even though this example uses a subsetting IF, this technique is useful anytime you need to process multiple logical conditional tests.

Note the use of the ranges 210 - 230 and 300 - 399. This will include all values between these bounds. Any variable with a value between 210 and 230 or 300 and 399 will be included.

The WHERE statement can be used in either DATA or PROC steps. Combined with user defined formats, your data can easily be subset in a procedure without the need to create an extra data step to select observations. The elimination of a DATA step and creation of an additional dataset every time you wanted to select observations for a Procedure can result in savings of both I/O and DASD.

For example, you might have a SAS dataset with observations containing data identified by County. If you wanted to select observations from the dataset for a particular District (comprising multiple Counties), the typical SAS program would first require either a variable with the District number and/or a DATA step to subset the data:

```
IF (county eq '12' | county eq '27' |
    county eq '83') ;
```

Unfortunately, every time you needed to select a different District, you would need to change the values of the counties to select.

Contrast this with a format that assigns a District to each County number. Using a custom format (\$DISTR.), you can eliminate the DATA step and invoke a Procedure with a WHERE statement such as:

```
PROC PRINT ;
  WHERE (put(county,$distr.) eq '7');
```

Only those observations which had a value for County that formatted to '7' (i.e. District 7 in this example) would be included in the Procedure. All other observations would be excluded. To change your program to select District 1, for example, would require only a single character change in the WHERE statement.

You can also use custom formats to group information for summarization. Instead of using a DATA step to accumulate totals to a summary level for a procedure like TABULATE, consider:

```
PROC FORMAT ;
  VALUE sumgrp
    low   - 10000 = 'Below 10,000'
    10001 - 20000 = '10,001 - 20,000'
    20001 - 30000 = '20,001 - 30,000'
    30001 - high  = 'Above 30,000'
RUN ;

PROC TABULATE ;
  FORMAT annualpy sumgrp. ;
```

This will create a table of the data summarized to the four groups defined, with each group labeled according to the format definition. All observations will be included, and the ANNUALPY values will be summarized into one of four categories. A DATA step to summarize information prior to the PROC TABULATE is totally unnecessary in this case.

USER DEFINED INFORMATS

Informats defined with the PROC FORMAT INVALUE statement can be useful to prevalidate INPUT data from an external data source.

Like the VALUE and PICTURE statements, the INVALUE statement is used by PROC FORMAT to define a custom format. The INVALUE defines an *informat*, which is used to convert the value of a character variable into a new value, either character or numeric. The maximum length of the label for both VALUE and INVALUE statements is now 200 characters.

You can use informats to convert:

- a character number to a character string ('2' to 'TWO')
- a character string to a number ('TWO' to 2)
- a character string to a different character string ('DEPT' to 'DEPARTMENT')

It is important to be aware that user-defined informats read only character data. They can create numeric values, but they must start with character data.

How many times have you needed to make assignments based on your input data?

For example:

```
DATA HISTORY ;
  INPUT DEPT $3. .... ;
  IF (DEPT IN('100','102','110'))
    THEN GROUP = 'ADMIN' ;
  ELSE IF (DEPT IN('600','610'))
    THEN GROUP = 'SALES' ;
  ELSE IF (DEPT IN('700','701','705'))
    THEN GROUP = 'MFG' ;
```

Contrast the above (which probably looks very familiar), with a custom informat:

```
PROC FORMAT ;
  INVALUE $DEPT '100'-'199' = 'ADMIN'
                '600'-'699' = 'SALES'
                '700'-'799' = 'MFG' ;
RUN ;
DATA HISTORY ;
  LENGTH DEPT $ 6 ;
  INPUT DEPT $DEPT3. ....
```

In a single step, the INPUT statement reads in a value and converts the value based on the informat. Instead of changing the conditional logic in your SAS program, it is easier to maintain a Format Table. The use of Format Tables simplifies program maintenance and provides more flexibility when changing your code.

There are several options that you can use to control the actions of the INVALUE statement. These options include:

DEFAULT = n specify the default width for the informat.
If no default width is specified, the length of the longest informatted value will be used

JUST left justify all input strings before they are compared

UPCASE convert all input strings to uppercase before they are compared

With the following example, all values would be left justified before the format \$DEPT. was applied. Thus, '100' and ' 100' would both be assigned the value of 'ADMIN'.

```
PROC FORMAT ;
  INVALUE $DEPT (JUST)
    '100'-'199' = 'ADMIN'
    '600'-'699' = 'SALES'
    '700'-'799' = 'MFG ' ;
RUN ;

DATA ;
  INPUT @5 ACCT @5 DEPTNAME $DEPT3. ;
```

Note that in the above example the same field was read and assigned to two different variables. ACCT was unformatted and contains the value read, while DEPTNAME was formatted with the value of the \$DEPT. informat.

There are also a number of special values that can be used with PROC FORMAT. These special values include LOW, HIGH, OTHER, _SAME_ and _ERROR_.

Both LOW and HIGH are used to define ranges of formatted values, a bottom range and a top range. OTHER is used to define how to format those values that are not otherwise defined. _SAME_ and _ERROR_ can be used to define the formatted data

values to store as a result of the format operation. _SAME_ defines that no change is to occur to the formatted value. The _ERROR_ value indicates invalid data, and is very useful in validating input data.

For example, you can pre-validate survey questionnaire data with this method:

```
INVALUE $answer (UPCASE) 'Y', 'N' = _same_
                                OTHER = . ;
```

This will accept any value that is 'Y', 'y', 'N', or 'n', but it will force any other response to missing. The _ERROR_ value would have forced an error condition for every observation where the variable did not have a value of 'Y', 'y', 'N', or 'n'.

In addition to the missing value . (single period), there are 27 additional missing values that can be assigned with PROC FORMAT. They are designated as .A through .Z and also ._ (period followed by an underscore). Thus, it is possible to assign different missing values to a variable based on a range of valid and invalid values. For example, to determine if blood pressure readings are out of range on the high or low side, you might use PROC FORMAT like this:

```
PROC FORMAT ;
  INVALUE sbp      low - <40   = .L
                    40 - 300   = _same_
                    301 - high = .H ;
  INVALUE dbp      low - <10   = .L
                    10 - 150   = _same_
                    151 - high = .H ;
  VALUE  range     .H = 'High'
                    .L = 'Low'
                    .  = 'Missing' ;
RUN ;

DATA ;
  FORMAT sbp dbp range. ;
  INPUT  @1 id    $3.
         @4 sbp   sbp3.
         @7 dbp   dbp3. ;

CARDS ;
001160090
002310220
003020008
004   080
005150070
;
RUN ;
```

Valid values are unchanged, because they are read with an informat value of _same_. Invalid values are assigned either Low or High depending on the range.

User defined informats can also be used to group records as the data is being read. For example, an informat called AREACD can be used to assign a value for State based on the telephone area code:

```
INVALUE $AREACD '305','407','813','904'
                = 'FLORIDA'
                '404','706','912'
                = 'GEORGIA'
                OTHER = 'UNKNOWN' ;
```

With this informat it is no longer necessary to read in and save the State of a record, since it can be derived from the telephone area code. The informat can also be used to validate that the state and area code data is consistent.

Another example which validates that data is within a specific list of valid values, but that forces an error condition for any invalid value:

```
INVALUE $PARTNO (UPCASE)
  'A100'-'A999' 'B100'-'B999'
  'C100'-'C999' = SAME
                OTHER = _ERROR ;
```

You can also use informats to validate SAS dates:

```
INVALUE chekdate
  '01JAN1990'd-'31DEC1995'd = [mmddyy6.]
                OTHER = . ;
```

Note the use of the brackets to define the SAS informat to use. Any valid SAS informat can be used in this manner as long as the informat is enclosed in either square brackets - [date9.] - or with parentheses and vertical bars - (| date9. |). This feature is also available for SAS VALUE. This informat will convert any valid SAS date between 01/01/90 and 12/31/95 to an MMDDYY6. informat, while any other value will be forced to a missing value.

These are some examples of what you can do with custom user-defined informats in your SAS code. User-defined informats can help make your SAS programs more flexible, more efficient, and easier to maintain.

FORMAT LIBRARIES

One problem with PROC FORMAT has been that formats were limited to only a single library location. This meant that all permanent formats used in a SAS program had to be stored in the same location. In Release 6 of the SAS System there is now the

capability to search through multiple locations until a particular format is found. With this option, it is now possible to establish several different levels of custom formats -- personal, departmental, and company-wide formats. For example, a financial chart of accounts might be organization-wide, while the balance sheet example discussed above might be a personal format.

Use the FMTSEARCH option to define multiple format locations. FMTSEARCH specifies which libraries and catalogs are to be searched, and the order to look for a custom format. For example, if you have custom formats stored in the FINANCE catalog of a company library as well as your personal library, you would code it like this:

```
LIBNAME ORG ' ' ;
LIBNAME LIBRARY ' ' ;

OPTIONS FMTSEARCH =
  (ORG.FINANCE ORG.DEPT LIBRARY) ;
```

This would define the search path for any format as the FINANCE catalog of the ORG library, followed by the DEPT catalog of the ORG library, followed by the permanent library specified in the second LIBNAME statement.

CNTLIN and CNTLOUT DATA SETS

So far, we have discussed creating and maintaining user-defined formats and informats by using the VALUE and INVALUE statements and PROC FORMAT. In Release 6 of the SAS System, there is now the ability to use SAS datasets as input to PROC FORMAT. You can create a SAS dataset which contains the values to be formatted, and use that to create the PROC FORMAT VALUE and INVALUE statements. This SAS dataset is called a CNTLIN data set. CNTLIN data sets are regular SAS data sets, but they have several required variables.

These variables are:

FMTNAME - a character variable containing the name of the format to create from that observation

TYPE - a character variable that contains the format type. Valid values for TYPE are:

- P picture
- C character format
- I numeric informat
- J character informat

START - the starting (or only) range value for an observation

END -(optional) the ending range value for an observation

LABEL - the character string to use as the label for this range

HLO - (optional) special range information.

blank	No range defined
L	ending range is LOW
H	ending range is HIGH
O	range is OTHER
N	no ranges defined
R	ROUND option on PICTURE format
F	formatted value or Invalue
I	numeric informat range
S	NOTSORTED option in effect

For the accounting example mentioned above, a SAS data set might be extracted from existing data. The data set would contain observations with values such as the following:

FMTNAME	TYPE	START	LABEL
\$ACCT	C	'105'	'Cash on Hand'
\$ACCT	C	'106'	'Cash in Bank'
\$ACCT	C	'110'	'Accts Receivable'

There may be other variables in the data set, but these variables can now be used as input to PROC FORMAT. The syntax to use a CNTLIN data set with PROC FORMAT is as follows:

```
PROC FORMAT CNTLIN=dsname fmtlib page ;
RUN;
```

Note that there is no VALUE, INVALUE, or PICTURE statement used if you use a CNTLIN data set. The CNTLIN data set would create a custom format for EVERY format name defined in the FMTNAME variable. The values of the START, END, and LABEL variables would be the same as the code needed to define the VALUE or INVALUE statement manually.

In the example above, there are two new options to PROC FORMAT listed. FMTLIB prints a report of the contents of a format catalog. PAGE specifies that each format or informat is printed on a separate page. With the FMTLIB option, it is now possible to document the contents of SAS format libraries.

The syntax to create a CNTLOUT data set with PROC FORMAT is as follows:

```
PROC FORMAT CNTLOUT=dsname ;
RUN;
```

This will contain a SAS dataset (dsname) that contains all the information about every format in that library. The variables in the dataset include FMTNAME, START, END, LABEL, and TYPE. These are the same variables that were used in the CNTLIN dataset to create the format.

The CNTLOUT dataset contains one observation for every formatted value start and end range. In the example on this page, the CNTLOUT dataset would have three observations. If you choose, this CNTLOUT dataset can be modified within your SAS program the same as any other dataset, and then used as an input CNTLIN dataset to recreate any formats that need to be changed.

In addition, there are two other new options to PROC FORMAT. These options are NOREPLACE and NOTSORTED. The NOREPLACE option prevents existing formats and informats with the same name from being replaced every time you run the SAS job. The NOTSORTED option stores the format with the ranges in the original order and performs format searches sequentially in that order. If you have certain format values that are used more frequently, you might be able to achieve some program execution efficiencies if you specify them first in the format and store the format with the NOTSORTED option.

USING FORMATS FOR PROGRAM EFFICIENCY

> Typically, SAS programmers store all the information about each observation in their datasets. For example, in the Chart of Accounts example it is natural to store the account description for each account. In a mailing list application, it would be natural to store the City and State information in addition to the Zip Code. However, since a pre-defined format can always be used to derive the information about the value of a variable, it does not make sense to create and store this information in your SAS dataset. Suppose the additional information was character data with a length of 50 characters. This means that SAS needs to store 50 additional characters for EVERY observation in the dataset. Yet the

information can always be derived by applying the format whenever it is necessary. In addition, the result is always the CURRENT value, which is particularly useful where data changes regularly.

- Considerable DASD savings can be achieved by storing only coded variables. For example, using the format \$GENDER., you only need to store the single character 'M' or 'F' instead of MALE or FEMALE.
- Maintenance of SAS programs is simplified through the use of custom formats. If you need to change the values of an assignment or logical condition (as in the examples above), it is much simpler to change a single instance in the format than every instance in your program.
- Custom formats can also simplify the maintenance of DATA step reports. Typically, all text is either stored in a variable or hard-coded in the Data step. Changes to values often requires a programming change to the step. Often this information can be accessed via custom formats, thus eliminating the need to modify your program every time the information changes.

CONCLUSION

This paper examined some of the uses that custom user-defined formats have for data validation, program efficiency, and ease of maintainability.

Through the use of PROC FORMAT, the SAS programmer has the ability to define a wide variety of custom formats and informats. These formats and informats can be used:

- to pre-validate and modify input data
- assign values to variables
- assign and use special missing values
- conditionally validate data based on changing criteria
- subset data
- improve program efficiency
- reduce data storage requirements
- improve the flexibility of SAS programs

Recent enhancements to PROC FORMAT such as the addition of CNTLIN and CNTLOUT data sets have simplified the process of creating and maintaining custom formats and informats.

Custom formats and informats developed with PROC FORMAT are an under-utilized tool that is already part of the basic toolkit of every SAS programmer. This paper has demonstrated some of the techniques to make this tool truly useful for all SAS programmers.

REFERENCES

Levine, Howard (1992). Using and Understanding SAS Formats. *Proceedings of the Seventeenth Annual SAS Users Group International Conference*. SAS Institute, Inc., Cary, NC. pp. 274-280.

Cody, Ron (1994). Some Clever Things to Do with User Defined INFORMATS. *Proceedings of the Nineteenth Annual SAS Users Group International Conference*. SAS Institute, Inc., Cary, NC. pp. 1325-1330.

Katsanis, Gary (1994). Table Lookups in the SAS Data Step. *Proceedings of the Nineteenth Annual SAS Users Group International Conference*. SAS Institute, Inc., Cary, NC. pp. 1364-1370.

SAS Institute, Inc., *SAS® Procedures Guide, Version 6, Third Edition* Cary, NC. SAS Institute Inc., 1990. pp 275-312.

SAS Institute, Inc., *SAS® Technical Report P-222, Changes and Enhancements to Base SAS® Software, Release 6.07*, Cary, NC. SAS Institute Inc., 1991. pp 207 -217

ACKNOWLEDGEMENTS

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

The author would like to thank Andrew Kuligowski for his suggestions and assistance in the preparation of this paper.

AUTHOR

The author may be contacted at:

S. David Riba
JADE Tech, Inc.
P O Box 4517
Clearwater, FL 34618
(813) 726-6099
INTERNET: 0006495817@mcimail.com