Paper PO06

Building Dynamic Informats and Formats

Michael Zhang, Merck & Co., Inc, West Point, PA

ABSTRACT
Using the FORMAT procedure to define informats and formats is a common task in SAS programming. This paper reviews two existing ways of defining dynamic informats and formats before introducing a new and more flexible approach to utilizing the procedure.

INTRODUCTION
Informats and formats give the SAS System information about data that is to be read or written. It is safe to believe that every SAS programmer has experiences using PROC FORMAT to build formats through hardcoding. While it is straightforward to use INVALUE and VALUE statements and hardcoded ranges and informatted/formatted values to define informats and formats, one can easily make a typing error in hardcoding when one is entering hundreds of ranges and informatted/formatted values. Besides, informats and formats defined this way are not robust against possible changes in the number of ranges and informatted/formatted values. In some cases there is no alternative to dynamic definition of informats/informats.

By dynamic definition of informats/formats we mean that the ranges and informatted/formatted values are obtained at run time.

Example 1:
In programming menu-driven parameter selection programs for vaccine projects, users typically require that the parameter selection interface offers the capability of selecting, renaming, combining, and reordering treatment groups. Since treatment information varies from protocol to protocol within one study and the program accepts renamed treatment labels entered by a user via an interface window, it is not possible to define formats for treatment group through hardcoding.

Example 2:
In programming medical history reports, there can be a large number of different medical terms (dictionary terms or broader terms) within each System Organ Class (SOC). For example, there are almost 200 different medical terms within SOC ‘Infections and Infestations’ for one of the vaccine studies involving more than ten thousand subjects. Although it is not absolutely necessary to build formats for SOC and medical terms, it definitely facilitates programming that generates output report with medical terms grouped under corresponding SOCs. Again it is not possible to define formats by hardcoding as the number of different medical terms within each SOC varies from protocol to protocol.

BUILDING DYNAMIC INFORMATS/FORMATS

Approach 1 – PROC FORMAT with the CNTLIN=SAS-data-set

This approach creates a format or informat from information stored in a SAS data set called input control data set. One specifies an input control data set with the CNTLIN= option in the PROC FORMAT statement. The FORMAT procedure uses the data in the input control data set to construct informats and formats. Thus, one can create informats and formats without writing INVALUE, VALUE, or PICTURE statements.
An input control data set can be a codebook file that matches variable values to character strings; it can also be the output control data set produced from a previous FORMAT procedure execution with the CNTLOUT= option.

An input control data set must at least contain the three variables – FMTNAME, START, and LABEL, where FMTNAME is character variable whose value is the name of informat or format, START is character variable that gives the range’s starting value, and LABEL is character variable whose value is the informatted or formatted value.

Each observation of the input control data set must have a valid SAS informat or format name value for the variable FMTNAME or else the observation will be ignored with an error message output to log. For current SAS Version 8.2, the naming conventions of SAS formats and informats are summarized in Table 1:

Table 1 – SAS Formats Naming

<table>
<thead>
<tr>
<th>Informats</th>
<th>Numeric</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formats</td>
<td>up to 7 characters, not ending in a number</td>
<td>up to 6 characters, not ending in a number</td>
</tr>
<tr>
<td>Formats</td>
<td>up to 8 characters, not ending in a number</td>
<td>up to 7 characters, not ending in a number</td>
</tr>
</tbody>
</table>

Note that the length of informat name is one less than that of format name. This is because SAS internally prefixes an at sign (@) to the informat name when storing the informat.

SAS also offers character variable TYPE in the input control data set to indicate the type of format. Possible meaningful values of TYPE are:

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

Although the SAS manual states that a TYPE variable value is indispensable for creating a character format, a numeric informat, a character informat, or a PICTURE statement format, we find that there are alternatives to using TYPE variable in all cases except for the picture format. The alternatives are to let FMTNAME values have a leading at sign (@) or dollar sign ($) or combination of these two signs. This is illustrated as follows:

<table>
<thead>
<tr>
<th>Value of FMTNAME</th>
<th>Format Constructed</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmt1_</td>
<td>numeric format - fmt1_</td>
</tr>
<tr>
<td>@fmt3_</td>
<td>numeric informat - fmt3_</td>
</tr>
<tr>
<td>@$fmt4_</td>
<td>character informat - $fmt4_</td>
</tr>
</tbody>
</table>

One can create more than one informat/format from an input control data set if the observations for each informat/format are grouped together.
Now, suppose data set DRILL contains variables FMTNAME, START, END, LABEL, and TYPE with the following observations:

<table>
<thead>
<tr>
<th>FMTNAME</th>
<th>START</th>
<th>END</th>
<th>LABEL</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGE1_</td>
<td>1</td>
<td>1</td>
<td>Young</td>
<td></td>
</tr>
<tr>
<td>AGE1_</td>
<td>2</td>
<td>2</td>
<td>Old</td>
<td></td>
</tr>
<tr>
<td>AGE2_</td>
<td>1</td>
<td>1</td>
<td>Young</td>
<td>C</td>
</tr>
<tr>
<td>AGE2_</td>
<td>2</td>
<td>2</td>
<td>Old</td>
<td>C</td>
</tr>
<tr>
<td>$AGE3_</td>
<td>1</td>
<td>1</td>
<td>Young</td>
<td></td>
</tr>
<tr>
<td>$AGE3_</td>
<td>2</td>
<td>2</td>
<td>Old</td>
<td></td>
</tr>
<tr>
<td>YN1_</td>
<td>Yes</td>
<td>Yes</td>
<td>1</td>
<td>I</td>
</tr>
<tr>
<td>YN1_</td>
<td>No</td>
<td>No</td>
<td>2</td>
<td>I</td>
</tr>
<tr>
<td>@YN2_</td>
<td>Yes</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>@YN2_</td>
<td>No</td>
<td>No</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>YN3_</td>
<td>Yes</td>
<td>Yes</td>
<td>1</td>
<td>J</td>
</tr>
<tr>
<td>YN3_</td>
<td>No</td>
<td>No</td>
<td>2</td>
<td>J</td>
</tr>
<tr>
<td>@YN4_</td>
<td>Yes</td>
<td>Yes</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>@YN4_</td>
<td>No</td>
<td>No</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>DAYS</td>
<td>01</td>
<td>31</td>
<td>00</td>
<td>P</td>
</tr>
</tbody>
</table>

Then \texttt{proc format cntlin=DRILL; run;} is equivalent to

\begin{verbatim}
proc format;
  value age1_ 1='Young'
                2='Old';
  value $age2_ '1'='Young'
                '2'='old';
  value $age3_ '1'='Young'
                '2'='old';
  invalue yn1_ 'Yes'=1
                'No'=2;
  invalue yn2_ 'Yes'=1
                'No'=2;
  invalue $yn3_ 'Yes'=1
                'No'=2;
  invalue $yn4_ 'Yes'=1
                'No'=2;
  picture days 01-31='00';
run;
\end{verbatim}

One can use a VALUE, INVALUE, or PICTURE statement in the same PROC FORMAT step with the CNTLIN= option. Keep in mind, though, that the control data set information is processed before the information given in the statements. As a result, if the VALUE, INVALUE, or PICTURE statement is creating the same informat or format that the CNTLIN= option is creating, the VALUE, INVALUE, or PICTURE statement creates the informat or format and the CNTLIN= data set is not used. One can, however, create an informat or format with VALUE, INVALUE, or PICTURE and create a different informat or format with CNTLIN= in the same PROC FORMAT step. For example:

\begin{verbatim}
proc format cntlin=DRILL;
  value gender 1='Male'
                2='Female';
run;
\end{verbatim}
One drawback of this approach is that the lengths of the variables START, END, and LABEL in the input control dataset must be determined in advance to accommodate the longest ranges and informatted/formatted values.

**Approach 2 – Writing SAS code that is later used in the program**

This approach employs PUT statement to generate SAS code in an external file and then uses %INCLUDE to bring the generated code into the program. In the following example, an existing SAS data set (ACCTINFO) contains account codes and corresponding customer names. There is one observation per account number. The observations in the data set are as follows:

<table>
<thead>
<tr>
<th>ACCTNUM</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>5008074</td>
<td>John Smith</td>
</tr>
<tr>
<td>5008075</td>
<td>Bill Jones</td>
</tr>
<tr>
<td>5009766</td>
<td>Benjamin Estes</td>
</tr>
</tbody>
</table>

Using the data in this data set, we want to create a format that converts an account code into a customer name. The code can be set up as the following:

```sas
filename tmpfmt 'external_file';
data _null_;
set acctinfo end=eof;
file tmpfmt;
if _n_=1 then do;
    put @1 "proc format;" ;
    put @8 "value account";
end;
put @8 acctnum @20 "= ' name '';"
if eof then do;
    put @8 ";
    put @1 "run;";
end;
run;
%include tmpfmt;
```

The dynamically created code produces:

```sas
proc format;
  value account
      5008074 = 'John Smith '
      5008075 = 'Bill Jones '
      5009766 = 'Benjamin Estes '
    .
    .
    .
; run;
```

The drawback to this approach is the difficulty in generating SAS code in one external file capable of defining multiple informats/formats.
Approach 3 – Using macro to output a string of ranges and informat/formatted values

This approach uses information in a SAS data set to create a series of macro variables. These macro variables are then used in a macro to construct a string of ranges and informat/formatted values needed in PROC FORMAT.

Consider, again, creating the ACCOUNT format from the data set ACCTINFO discussed in Approach 2. Approach 3 consists of three steps:

1) Create a series of paired macro variables

```
data _null_;  
  set acctinfo end=eof;  
  call symput('acctno'||compress(put(_n_,best12.)),compress(put(acctnum,best12.)));  
  call symput('name'||compress(put(_n_,best12.)),trim(left(name)));  
  if eof then call symput('tot_no',compress(put(_n_,best12.)));  
run;
```

2) Create a macro that outputs the string of ranges and formatted values

```
%macro acct_fmt;  
  %local i;  
  %do i=1 %to &tot_no;  
    %str(&&acctno&i = "&&name&i")  
  %end;  
%mend acct_fmt;
```

3) Call the macro in PROC FORMAT

```
proc format;  
  value account %acct_fmt  
;  
run;
```

The core of Approach 3 is to generate the string of ranges and informat/formatted values and it offers flexible way of doing so. For example, if you want to bold the formatted values in RTF table, all you need to do is modify step 2) as follows:

```
%macro acct_fmt;  
  %local i;  
  %do i=1 %to &tot_no;  
    %str(&&acctno&i="\b &&name&i\b")  
  %end;  
%mend acct_fmt;
```

Advantages of Approach 3 over Approach 1 and 2:

- Approach 3 does not involve creating a control input data set, nor does it need to create an external SAS code;
- Approach 3 is more general and more flexible;
• With Approach 3, you can define all informats and formats more easily under one PROC FORMAT in your program;

• The series of macro variables created in step 1) of Approach 3 can be used to construct both informats and formats.

In the following example, we show how combining treatment groups is implemented in the programming through an interface. Suppose a study involves four (4) treatments. The user wishes to combine the four treatments into two groups and name the two combined treatment groups interactively on the screen. First, we create a macro %TRT_WIN that has the following functions:

• Creates a window to display the four treatment descriptions.

• Accepts user’s selections to combine treatments and name the two combined treatment groups.

• Creates a dataset that reflects user’s selections.

Next, macro %TRT_WIN is called in the program and a dataset TRTGRPS is created. The observations of the dataset TRTGRPS are as follows:

<table>
<thead>
<tr>
<th>TRTDSC</th>
<th>TRTLBL</th>
<th>TRTNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Treatment1</td>
<td>Active</td>
<td>1</td>
</tr>
<tr>
<td>Treatment2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Treatment3</td>
<td>Placebo</td>
<td>2</td>
</tr>
<tr>
<td>Treatment4</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

The dataset TRTGRPS reflects that Treatment 1 and Treatment 2 were combined with new label ‘Active’ and Treatment 3 and Treatment 4 were combined with new label ‘Placebo’. The values of variable TRTNUM dictate the order of combined treatment groups to be displayed in the output table.

This dataset allows us to create an informat and a format. The informat will map each original treatment to combined treatment group index (TRTNUM). The format will translate TRTNUM into combined treatment group label (TRTLBL). The programming can be outlined as follows:

**Step 1:** Create two groups of paired macro variables, one for informat definition and the other format definition, using Data _null_ similar to the example given above.

**Step 2:** Create two macros, %in_fmt and %fmt, that output the strings of ranges and informatted/formatted values, using the two groups of paired macro variables, respectively.

**Step 3:** Call the two macros in PROC FORMAT to define in_trt. informat and out_trt. format:

```plaintext
proc format;
  value in_trt %in_fmt ;
  value out_trt %fmt ;
run;
```
Additionally, we may need to provide footnotes in the output table revealing what treatments each combined treatment group contains. This can also be accomplished starting from dataset TRTGRPS.

First, we define macro variable LBL_NO to be the number of combined treatment groups.

Next, we generate macro variables LABELi and NEW_GRPi, where i indexes combined treatment group, i = 1 to &LBL_NO. LABELi describes label for combined treatment group i. NEW_GRPi stores original treatments information for combined treatment group i.

Finally, we build macro \%footnotes as follows:

\%macro footnotes;
   %local i;
   %do i=1 %to &lbl_no;
      footnote&i "Treatment Label: &&label&i - Treatment Groups(s): &&new_grp&i";
   %end;
%mend footnotes;

In our case, invoking macro \%footnotes causes SAS to execute two footnote statements:

footnote1 "Treatment Label: Active - Treatment Group(s): 'Treatment1', 'Treatment2'";
footnote2 "Treatment Label: Placebo - Treatment Group(s): 'Treatment3', 'Treatment4'";

CONCLUSION
In this paper we have introduced a novel approach to building dynamic informats/formats and compared it to two other existing approaches. Through the examples, we have shown that the three-step new approach is more flexible and of more practical use.

REFERENCES


CONTACT INFORMATION
Your comments and suggestions are valued and encouraged. Contact the author at:

   Michael Zhang
   Merck & Co., Inc.
   UN-A102
   785 Jolly Rd
   Blue Bell, PA 19422
   (484)344-7651
   (484)344-7105(Fax)
   michael_zhang@merck.com