Paper 022-2010

Using Advanced Features of User-defined Formats and Informats

Ron Cody, Camp Verde, Texas

INTRODUCTION

Formats can do more than just make your output more readable. This paper discusses several of the more advanced features of formats, such as using formats to create new variables, using informats to filter data errors, reading combinations of character and numeric values in one step (using enhanced numeric informats), using formats for table look-up, embedding formats within other formats, multi-label formats, control data sets, and creating format names on the fly in a DATA Step to allow multi-way table lookups.

USING FORMATS WITH A PUT FUNCTION TO CREATE NEW VARIABLES

There are times when you would like to create a new variable consisting of formatted values. Program 1 demonstrates how this is done.

Program 1 Using a format and a PUT function to create a new variable

```
proc format;
   value agefmt  0 - <20  = '< 20'
                20 - <40  = '20 to 39'
                40 - <60  = '40 to 59'
                60 - high = '60+';
run;

data survey;
   set learn.survey;
   AgeGroup = put(Age,agefmt.);
run;
```

The key to this program is the PUT function. This function take the value of its first argument, formats this value with the format listed as the second argument, and returns the formatted value. Remember that the PUT function always returns a character value. In this example, AgeGroup is a character variable with a length of eight bytes (the length of the longest formatted value). Here is a listing of data set SURVEY:

```
Listing of SURVEY

ID   Gender  Age  Salary  Ques1  Ques2  Ques3  Ques4  Ques5  AgeGroup

001    M      23   28000    1      2      1      2      3    20 to 39
002    F      55   76123    4      5      2      1      1    40 to 59
003    M      38   36500    2      2      2      2      1    20 to 39
004    F      67  128000    5      3      2      2      4    60+
005    M      22   23060    3      3      3      4      2    20 to 39
006    M      63   90000    2      3      5      4      3    60+
007    F      45   76100    5      3      4      3      3    40 to 59
```

CREATING USER-DEFINED INFORMATS

Although you are probably familiar with using PROC FORMAT to create user-written formats, you may not have used this procedure to create your own informats. You can use user-written informats to alter values as they are read from a raw data file, or with the INPUT function to perform a table look-up. Let's look at an example:

Your raw data consists of ID's and letter grades (A+, A, A-, etc.). You want to convert these letter grades into numbers according to a predefined table. The program below creates an informat called CONVERT and uses that informat to read the letter grades.

Program 2 Demonstrating a user-written INFORMAT

```
proc format;
   invalue convert 'A+' = 100
                    'A'  = 96
                    'A-' = 92
                    'B+' = 88
                    'B'  = 84
                    'B-' = 80
                    'C+' = 76
                    'C'  = 72
                    'F'  = 65;
run;

data grades;
   input @1 ID         $3.
         @4 Grade convert2.;
datalines;
001A-
002B+
003F
004C+
005A
;
```

You use an INVALUE statement to create an informat. The rules are similar to the ones you use in creating formats. One tiny difference is that informat names can only be 31 characters in length (including the $ if it will be used to read character data). For those with curious minds, SAS® uses one byte in the format catalog (the @ sign) to differentiate between formats and informats (if you look at the entries in your catalog with the FMTLIB option or PROC CATALOG, you will see the @ signs added to the informat names).

Following the keyword INVALUE, you type the name of the informat you want to create. Just as with formats, you must start the name with a dollar sign if it is to be used to create character data. In Program 2 above, even though you are reading character values (A+, A, A-, etc.) the resulting variable (Grade) will be a numeric variable. If you used the name $CONVERT instead of CONVERT in this program, the variable Grade would be character.

Next, you specify ranges, and equal sign and labels. Again, the same rules as formats.

You use your informat just as you would a built-in SAS informat. Notice that you can add a width to it to specify how many columns of data to read. Here is a listing of data set GRADES:

```
Listing of GRADES

ID     Grade

001      92
002      88
003      65
004      76
005      96
```

It is important to remember that the original character data values are not part of this data set. If you wanted both the original letter grade and its corresponding numeric value, you could read the letter grade as a character variable and use an INPUT function (with the CONVERT informat) to create the numeric grade.

There are some useful options that you can use when you create your informats. UPCASE and JUST are two such options. UPCASE, as the name implies, will convert the data values to uppercase before checking on the informat ranges. JUST will left-align character values. Let's use these two options to enhance Program 2.

Program 3 Demonstrating INFORMAT options UPCASE and JUST

```
proc format;
   invalue convert(upcase just)
           'A+' = 100
           'A'  = 96
           'A-' = 92
           'B+' = 88
           'B'  = 84
           'B-' = 80
           'C+' = 76
           'C'  = 72
           'F'  = 65
         other  =  .;
run;

data grades;
   input @1 ID          $3.
         @4 Grade convert2.;
datalines;
001A-
002b+
003F
004c+
005 A
006X
;

title "Listing of GRADES";
proc print data=grades noobs;
run;
```

Notice that the raw data values being read contain upper- and lowercase values and the value for student 005 contains a leading blank. We also added a student with an invalid grade (X). To prevent error messages in the SAS Log, the informat category OTHER was added so that invalid values would be converted to numeric missing values. The listing below shows that all the data values were read correctly:

```
Listing of GRADES

ID     Grade

001      92
002      88
003      65
004      76
005      96
006       .
```

ENHANCED NUMERIC INFORMAT – READING CHARACTER AND NUMERIC DATA IN ONE STEP

There is a little-know feature in user-defined informats—the ability to create an informat that reads both character and numeric data and creates a numeric variable. The examples that follow show ways that you can take advantage of this feature.

For this first example, you have temperature readings on hospital patients. Since many of the patients have a "normal" temperature (98.6 degrees Fahrenheit), it is convenient to record normal temperatures by entering an 'N' as a data value. Some sample lines of data are:

101 N 97.3 n N 104.5

Notice that some of the N's are in uppercase and others in lowercase. Before we look at the elegant enhanced informat solution, let's look at a tradition approach:

Program 4 A traditional approach to reading a combination of character and numeric data

```
data temperatures;
   input Dummy $ @@;
   if upcase(Dummy) = 'N' then Temp = 98.6;
   else Temp = input(Dummy,8.);
   drop dummy;
datalines;
101 N 97.3 n N 104.5
;
```

Each data value is read as character data. A check is made to see if this value is equal to an upper- or lowercase 'N'. If so, Temp is set to 98.6—if not, the INPUT function performs the character to numeric conversion. This works fine, but compulsive programmers (who me?) are always looking for a more elegant solution. The program below uses an enhanced numeric informat:

Program 5 Using an enhanced numeric informat to read a combination of character and numeric data

```
proc format;
   invalue readtemp(upcase)
                96 - 106 = _same_
                'N'      = 98.6
                other    = .;
run;
data temperatures;
   input Temp : readtemp5. @@;
datalines;
101 N 97.3 n N 67 104.5
;
```

The UPCASE option converts any character to uppercase. The keyword _SAME_ in this informat leaves any numeric values in the range 96 to 106 unchanged. Values of 'N' are converted to the numeric value of 98.6 and any values that are not in the range 96 to 106 or equal to 'N' are set to a numeric missing value. The technique of using the keyword _SAME_ for valid values and setting other values to missing is one way to screen unwanted extreme values as they are being read. A listing of data set temperatures is shown next:

```
Listing of Data Set TEMPERATURES

 Temp

101.0
 98.6
 97.3
 98.6
 98.6
   .
104.5
```

The next program reads a combination of letter and number grades. The grades A-F are converted to number grades as follows: 'A' = 95, 'B' = 85, 'C' = 75, 'F' = 65. Number grades are not changed. Here is the program:

Program 6 Another example of an enhanced numeric informat

```
proc format;
   invalue readgrade(upcase)
      'A' = 95
      'B' = 85
      'C' = 75
      'F' = 65
      other = _same_;
run;

data school;
   input Grade : readgrade3. @@;
datalines;
97 99 A C 72 f b
;
```

Here the values A-F are converted to the appropriate numeric value and all other values are left unchanged. Here is a listing of the data set:

```
Listing of SCHOOL

Grade

  97
  99
  95
  75
  72
  65
  85
```

USING FORMATS (AND INFORMATS) TO PERFORM A TABLE LOOK-UP

Table look-up is a process where one or more data values are used to retrieve a value for another variable. For example, given an item number from a catalog, you might want to retrieve a product name and a price.

SAS provides you with a variety of ways to perform a table look-up, such as a data set MERGE, a key value from an index, an array, formats, and informats. Since SAS formats are stored in memory, they are fast and efficient. Program 7, uses a item number (ItemNumber) to look up a product name and its price. Here is the program:

Program 7 Using formats and informats to perform a table look-up

```
proc format;
   value namelookup
     122 = 'Salt'
     188 = 'Sugar'
     101 = 'Cereal'
     755 = 'Eggs'
   other = ' ';
   invalue pricelookup
     'Salt'   = 3.76
     'Sugar'  = 4.99
     'Cereal' = 5.97
     'Eggs'   = 2.65
      other   = .;
run;

data grocery;
   input ItemNumber @@;
   Name = put(ItemNumber,namelookup.);
   Price = input(Name,pricelookup.);
datalines;
101 755 122 188 999 755
;
```

A user-defined format is used, along with a PUT function, to retrieve a product name from the ItemNumber. An informat, relating product names and prices is used with an INPUT function to retrieve the price. The reason an INFORMAT was used for the price look-up was that the INPUT function can return a numeric value. Remember that the PUT function always returns a character value. (If you used a format, you would have to perform a character to numeric conversion to obtain a numeric value for the price.)

Both the format and informat use the keyword OTHER to assign a missing value to an invalid item code. A listing of data set GROCERY follows:

```
Listing of GROCERY

 Item
Number     Name     Price

  101     Cereal     5.97
  755     Eggs       2.65
  122     Salt       3.76
  188     Sugar      4.99
  999                 .
  755     Eggs       2.65
```

USING A SAS DATA SET TO CREATE A FORMAT

The example in the last section used only a small number of item codes. If you have a large number of values, writing a format statement by hand is tedious. Luckily, SAS allows you to use a specially constructed SAS data set to create a SAS format. The PROC FORMAT option CNTLIN= (control in) allows you to name the SAS data set (or data view) that you want to use.

Control input data sets require a minimum of three variables named FMTNAME, START, and LABEL. FMTNAME is a character variable that specifies the format or informat name; START is either a single value to be formatted or, if END is also used, the beginning of a range. You should include a dollar sign as the first character of FMTNAME if you are creating a character format. Optionally, you can include a variable called TYPE that is either a 'C' for character formats or 'N' for numeric formats.

If you have data values and format labels already stored in a SAS data set, you can use that data set as input to a DATA step (with variables renamed as needed) to create your control data set.

This sounds complicated, but an example should make it clear. You have a permanent SAS data set (CODES) that contains ICD-9 code values (International Classification of Diseases, version 9) and descriptions. You want to create a format that provides labels for each of the ICD-9 codes. So that you can try this yourself, the program below creates this permanent SAS data set CODES.

Program 8 Creating a test data set that will be used to make a CNTLIN data set

```
data learn.codes;
   input ICD9 : $5. Description & $21.;
datalines;
020 Plague
022 Antrax
390 Rheumatic fever
410 Myocardial infarction
493 Asthma
540 Appendicitis
;
```

You want to use this data set to create a control input data set as follows:

Program 9 Creating a CNTLIN data set from an existing SAS data set

```
data control;
   set learn.codes(rename=
                   (ICD9 = Start
                    Description = Label));
   retain Fmtname '$ICDFMT'
          Type 'C';
run;

title "Demonstrating a Input Control Data Set";
proc format cntlin=control fmtlib;
run;
```

The RENAME= data set option is used to rename ICD9 to START and Description to LABEL. A RETAIN statement is used to set FMTNAME to '$ICDFMT' and TYPE equal to 'C'. Using a RETAIN statement is more efficient than an assignment statement, since these values are set at compile time—an assignment statement executes for each iteration of the DATA step. The CNTLIN= option names this data set and the FMTLIB option generates a table showing the ranges and format labels. Below are both a listing of the CONTROL data set and the output from PROC FORMAT:

```
Listing of CONTROL

Start    Label                    Fmtname    Type

 020     Plague                   $ICDFMT     C
 022     Antrax                   $ICDFMT     C
 390     Rheumatic fever          $ICDFMT     C
 410     Myocardial infarction    $ICDFMT     C
 493     Asthma                   $ICDFMT     C
 540     Appendicitis             $ICDFMT     C
```

```
Demonstrating a Input Control Data Set

----------------------------------------------------------------------------
|       FORMAT NAME: $ICDFMT  LENGTH:   21   NUMBER OF VALUES:    6        |
|   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  21  FUZZ:        0   |
|--------------------------------------------------------------------------|
|START           |END             |LABEL  (VER. V7|V8   21NOV2005:10:14:05)|
|----------------+----------------+----------------------------------------|
|020             |020             |Plague                                  |
|022             |022             |Antrax                                  |
|390             |390             |Rheumatic fever                         |
|410             |410             |Myocardial infarction                   |
|493             |493             |Asthma                                  |
|540             |540             |Appendicitis                            |
----------------------------------------------------------------------------
```

Instead of creating a SAS data set to be used with CNTLIN, you could elect to create a data view instead. A data view can be thought of as a virtual data set. Depending on your application, using a data view may be more efficient than using a SAS data set. The only change to

Program 9 would be to replace the DATA statement with the following:

```
data control / view = control;
```

The program below shows how you might use this format.

Program 10 Using the CNTLIN= created data set

```
data disease;
   input ICD9 : $5. @@;
datalines;
020 410 500 493
;
title "Listing of DISEASE";
proc print data=disease noobs;
   format ICD9 $ICDFMT.;
run;
```

Here is the output:

```
Listing of DISEASE

ICD9

Plague
Myocardial infarction
500
Asthma
```

Notice that there is no format for the ICD-9 code of 500. If you were writing your own PROC FORMAT statements, you could use the keyword OTHER to provide a label for non-matching codes. If you want to accomplish this using a CNTLIN data set, you need a way to assign a value to OTHER. CNTLIN data sets use the variable HLO (stands for High, Low, Other) to indicate that you want to use one of these keywords and not an explicit range value.

If you want to assign the label, 'Not Found' for all ICD-9 codes not in the list, you proceed as follows:

Program 11 Adding an OTHER category to your format

```
data control / view=control;
   set learn.codes(rename=
                   (ICD9 = Start
                   Description = Label))
                   end = last;
   retain Fmtname '$ICDFMT'
          Type 'C';
   output;
   if last then do;
      Start = ' ';
      Hlo = 'o';
      Label = 'Not Found';
   end;
   output;
run;
```

Program 11 also demonstrates how to use a data view instead of a SAS data set. You want to add one observation at the bottom of your CONTROL data set with the value of HLO equal to 'o' (OTHER) and LABEL equal to 'Not Found'. You can do this by using the END= data set option to test for the end of the input data set. The variable Last will be true when the last observation in data set CODES has been read. HLO and LABEL are both assigned a value and a single observation is added to the end of the data set. Here is a listing of data set CONTROL:

```
Listing of CONTROL

     Obs     Start     Label                        Fmtname     Type     Hlo

       1      020      Plague                       $ICDFMT      C
       2      022      Anthrax                      $ICDFMT      C
       3      390      Rheumatic fever              $ICDFMT      C
       4      410      Myocardial infarction        $ICDFMT      C
       5      493      Asthma                       $ICDFMT      C
       6      540      Appendicitis                 $ICDFMT      C
       7               Not Found                    $ICDFMT      C        o
```

The last observation in this data set contains the instructions for setting OTHER equal to 'Not Found'. Although we decided to set START to a missing value in this program, It was not actually necessary since the keyword OTHER and not an actual range is being used.

When this new format is used in Program 10, the ICD-9 code of 500 is now labeled as 'Not Found'. Below are both the output from PROC FORMAT showing that the OTHER category was added to the format and a listing of the DISEASE data set:

```
        FORMAT NAME: $ICDFMT  LENGTH:   21   NUMBER OF VALUES:    7
    MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  21  FUZZ:        0

START             END               LABEL  (VER. V7|V8   24MAR2008:15:20:21)

020               020               Plague
022               022               Anthrax
390               390               Rheumatic fever
410               410               Myocardial infarction
493               493               Asthma
540               540               Appendicitis
**OTHER**         **OTHER**         Not Found
```

```
Listing of DISEASE

ICD9

Plague
Myocardial infarction
Not Found
Asthma
```

UPDATING AND MAINTAINING YOUR FORMATS: CNTLOUT

Suppose you want to add some ICD-9 codes to your existing $ICDFMT format. The easiest way to accomplish this is to first use the CNTLOUT= option of PROC FORMAT to create a data set containing all the formatting information. You can then use this data set in a DATA step to add new codes or modify existing ones.

As an example, suppose you want to add two new ICD-9 codes to your $ICDFMT format, 427.5 (Bronchitis) and 466 (Cardiac arrest). The following program will do the trick:

Program 12 Updating an existing format using a CNTLOUT data set

```
proc format cntlout=control_out;
   select $ICDFMT;
run;

data new_control;
   length Label $ 21;
   set control_out end=Last;
   output;
   if Last then do;
      Hlo = ' ';
      Start = '427.5';
      End = Start;
      Label = 'Cardiac Arrest';
      output;
      Start = '466';
      End = Start;
      Label = 'Bronchitis';
      output;
   end;
run;

proc format cntlin=new_control;
   select $ICDFMT;
run;
```

You first run PROC FORMAT with the CNTLOUT= option, creating an output data set containing all of the information necessary to recreate the format. A SELECT statement allows you to choose which format you want to use to create this data set. Here is a listing of this CNTLOUT data set:

```
Listing of CONTROL_OUT

FMTNAME START     END       LABEL                 MIN MAX DEFAULT LENGTH FUZZ

ICDFMT  020       020       Plague                 1   40    21     21     0
ICDFMT  022       022       Antrax                 1   40    21     21     0
ICDFMT  390       390       Rheumatic fever        1   40    21     21     0
ICDFMT  410       410       Myocardial infarction  1   40    21     21     0
ICDFMT  493       493       Asthma                 1   40    21     21     0
ICDFMT  **OTHER** **OTHER** Not Found              1   40    21     21     0

PREFIX MULT FILL NOEDIT TYPE SEXCL EEXCL HLO DECSEP DIG3SEP DATATYPE LANGUAGE

         0          0    C     N     N
         0          0    C     N     N
         0          0    C     N     N
         0          0    C     N     N
         0          0    C     N     N
         0          0    C     N     N    O
```

Notice that there are additional variables that PROC FORMAT uses when creating a format You do not need to concern yourself with these variables—SAS will use them as needed. The next step is to add observations to the end of this data set, containing the new codes and their labels. You can do this in a DATA step, by first reading all of the existing format information and then adding the new ones. (Alternatively, you could create a data set of the new codes and use PROC APPEND to add these observations to the end of the existing control data set.) A LENGTH statement is used to ensure that the storage length for Label is sufficient for any new labels you want to create.

Program 12 uses the data set option END= to determine when you have read the last observation from data set CONTROL_OUT. At this point, the statements in the DO group execute, outputting two observations to the end of the new data set. It is important to set the variable HLO (High, Low, Other) to missing and to set the variable End equal to the Start values. Since these variables exist in CONTROL_OUT data set and are read with a SET statement, they are automatically retained. Therefore, if you do not assign values to these variables, they will retain the values they had in the last observation in data set CONTROL_OUT.

Finally, you use the CNTLIN= option to recreate the $ICDFMT format. Instead of the FMTLIB option, a SELECT statement is used to identify which format you want to list. When you use a SELECT statement with PROC FORMAT, it is not necessary to also include the FMTLIB option. Here is the output from PROC FORMAT, showing that the format now contains the two additional codes:

```
Listing of CONTROL_OUT

 ------------------------------------------------------------------------------
 |       FORMAT NAME: $ICDFMT   LENGTH:   21   NUMBER OF VALUES:    8          |
 |   MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH  21  FUZZ:        0      |
 |----------------------------------------------------------------------------|
 |START           |END             |LABEL  (VER. V7|V8   22NOV2005:10:23:49)|
 |----------------+----------------+------------------------------------------|
 |020             |020             |Plague                                    |
 |022             |022             |Antrax                                    |
 |390             |390             |Rheumatic fever                           |
 |410             |410             |Myocardial infarction                     |
 |427.5           |427.5           |Cardiac Arrest                            |
 |466             |466             |Bronchitis                                |
 |493             |493             |Asthma                                    |
 |**OTHER**       |**OTHER**       |Not Found                                 |
 ------------------------------------------------------------------------------
```

USING FORMATS WITHIN FORMATS: NESTING FORMATS

When you define format or informat labels, you can also include the name of a SAS or user-written format or informat, rather than a text string in place of a label. Here is an example:

You want to read dates from July 15, 2005 to December 31, 2006 using the mmddyy10. informat. Dates before July 15, 2005 should be formatted as "Not Open" and dates after December 31, 206 should be formatted as "Too Late." You can use nested formats as follows to accomplish this task:

Program 13 Demonstrating nested formats

```
proc format;
   value registration low - <'15Jul2005'd = 'Not Open'
              '15Jul2005'd - '31Dec2006'd = [mmddyy10.]
              '01Jan2007'd - high         = 'Too Late';
run;
```

The format mmddyy10. is placed in square brackets where you normally place a format label. Program 14 uses this format to process registration dates and produce a list of dates and names.

Program 14 Creating a test data set

```
data conference;
   input @1  Name $10.
         @11 Date mmddyy10.;
   format Date registration.;
datalines;
Smith     10/21/2005
Jones     06/13/2005
Harris    01/03/2007
Arnold    09/12/2005
;
```

A listing of data set CONFERENCE is shown below:

```
Listing of CONFERENCE

 Name        Date

Smith     10/21/2005
Jones     Not Open
Harris    Too Late
Arnold    09/12/2005
```

The next example uses nested user-written informats. In this example, you are given data on benzene exposure. Some of the values are actual benzene levels in parts per million, other values are the year a worker was exposed. For each of the years from 1946 to 1952, benzene levels were tabulated. For years outside this range, the actual benzene levels were reported. Some sample data, consisting of either a year or a benzene level is shown below:

File: c:\books\learning\benzene.txt

```
001 90
002 1950
003 217
004 1952
005 177
```

All of the benzene levels are much less than any of the year values so there is no confusion whether to read a value as a year or a benzene level. Creating a data set of ID's and benzene levels is greatly simplified by nesting informats, like this:

Program 15 Using the nested format in a DATA step

```
proc format;
   invalue yearexp 1946 = 250
                   1947 = 244
                   1948 = 240
                   1949 = 200
                   1950 = 188
                   1951 = 150
                   1952 = 100;
   invalue exp low - <1946   = [7.1]
               1946 - 1952   = [yearexp.]
               1952< - high = [7.1];
run;

data benzene;
   infile 'c:\books\learning\benzene.txt';
   input ID Exposure : exp4.;
run;
```

The first informat, YEAREXP substitutes a benzene level for each of the years from 1946 to 1952. The informat EXP uses a SAS 7.1 informat for values less than 1946 or greater than 1952. Otherwise, the YEAREXP informat is used.

This problem could have also been solved using formats instead of informats. To use formats, you would need to read a raw data value and use a PUT function to obtain a formatted value. Since the result of a

PUT function is a character value, you would also need to use an INPUT function to obtain a benzene level as a numeric value. Creating informats makes the process much simpler.

MULTILABEL FORMATS

Under normal circumstances, you get an error message if any of your format ranges overlap. However, you can create a format with overlapping ranges if you use the multilabel option on the VALUE statement. Certain multilabel enabled procedures can then use the multilabel format to produce tables showing all of the format ranges. Here is an example:

You want to see the variable Age (from the SURVEY data set) broken down two ways—one, in 20 year intervals; the other by a split at 50 years old. You first create a multilabel format like this:

Program 16 Creating a MULTILABEL format

```
proc format;
   value agegroup (multilabel)
    0 - <20   = '0 to <20'
    20 - <40  = '20 to <40'
    40 - <60  = '40 to <60'
    60 - <80  = '60 to <80'
    80 - high = '80 +'

    0 - <50   = 'Less than 50'
    50 - high = '> or = to 50';
run;
```

Without the multilabel option, PROC FORMAT will issue an error message about overlapping ranges and fail to create the format.

You can use this format in PROC MEANS, PROC SUMMARY, and PROC TABULATE. Here is a PROC MEANS example:

Program 17 Using a MULTILABEL format with PROC MEANS

```
title "Demonstrating a Multilabel Format";
title2 "PROC MEANS Example";
proc means data=learn.survey;
   class Age / MLF;
   var Salary;
   format Age agegroup.;
run;
```

It is important to use the MLF (multilabel format) option on the class statement if you want to use this feature of the format. Here is the output:

```
Demonstrating a Multilabel Format
PROC MEANS Example

The MEANS Procedure

                        Analysis Variable : Salary

                N
Age           Obs   N        Mean      Std Dev      Minimum      Maximum
-------------------------------------------------------------------------
20 to <40       3   3    29186.67      6798.13     23060.00     36500.00

40 to <60       2   2    76111.50   16.2634560     76100.00     76123.00

60 to <80       2   2   109000.00     26870.06     90000.00    128000.00

> or = to 50    3   3    98041.00     26857.01     76123.00    128000.00

Less than 50    4   4    40915.00     24104.46     23060.00     76100.00
-------------------------------------------------------------------------
```

Inspection of this output shows average Salary broken down by Age in two ways: In 20 year intervals and by a split at 50 years old.

You can use multilabel formats with PROC TABULATE to create interesting tables. Here is a program to compute frequencies in a table of Age by Gender:

Program 18 Using a MULTILABEL format with PROC TABULATE

```
title "Demonstrating a Multilabel Format";
title2 "PROC TABULATE Example";
proc tabulate data=learn.survey;
   class Age Gender / MLF;
   table Age ,
         Gender;
   format Age agegroup.;
run;
quit;
```

This produces the following output:

```
Demonstrating a Multilabel Format
PROC TABULATE Example

                         Gender
                     F             M
                     N             N
Age
20 to <40                 .          3.00
40 to <60              2.00             .
60 to <80              1.00          1.00
> or = to 50           2.00          1.00
Less than 50           1.00          3.00
```

Notice that certain age groups are not shown in the table. For example, there were no subjects in the 0 to <20 age group. To see all of the possible format ranges in your table, you can use the PRELOADFMT option on the CLASS statement. Preloaded formats allow PROC TABULATE to list categories for which there are no data values.

Here is the PROC TABULATE code with the PRELOADFMT option and two additional options to control the printing of missing values. (They will be explained following the program.)

Program 19 Using the PRELOADFMT, PRINTMISS, and MISSTEXT options with PROC TABULATE

```
title "Demonstrating a Multilabel Format";
title2 "PROC TABULATE Example";
proc tabulate data=learn.survey;
   class Age Gender / MLF preloadfmt;
   table Age ,
         Gender / printmiss  misstext=' ';
   format Age agegroup.;
run;
quit;
```

The PRELOADFMT option forces the procedure to list all of the format ranges. This option will have no effect without the PRINTMISS option on the TABLE statement. Remember that PRINTMISS is an instruction to include categories that contain one or more missing values of a CLASS variable. Finally, the MISSTEXT= option tells the procedure what character (or characters) you would like printed that represents missing values (the default is a period). Here you choose a blank to represent missing values. Finally, the output:

```
Demonstrating a Multilabel Format
PROC TABULATE Example

                    Gender
                 F          M
                 N          N
Age
0 to <20
20 to <40                       3.00
40 to <60          2.00
60 to <80          1.00         1.00
80 +
> or = to 50       2.00         1.00
Less than 50       1.00         3.00
```

Now all the Age ranges appear in the table and the missing values are printed as blanks.

USING THE INPUTN FUNCTION TO PERFORM A MORE COMPLICATED TABLE LOOK-UP

This final example of the paper performs a two-way table look-up, using a function that allows you to compute an informat name in a DATA step. This section is a bit complicated and you may either want to skip it or refer to the SAS OnLine Doc™ for more information on the INPUTN function and control input data sets.

The problem: You have a table of years (1944 to 1949) and job codes (A through E). Each combination of year and job code has a benzene exposure (in parts per million) associated with it. The table below list these values:

	Job Code				
Year	A	B	C	D	E
1944	220	180	210	110	90
1945	202	170	208	100	85
1946	150	110	150	60	50
1947	105	56	88	40	30
1948	60	30	40	20	10
1949	45	22	22	10	8

One way to compute an exposure, given a year and job code, is to create an informat for each year, assigning each of the job codes a label containing the exposure value. An informat is used instead of a format since you can use an INPUT (or as you will see, an INPUTN) function with the user defined informats to obtain a numeric value directly. If you did this "by hand" you would proceed like this:

Program 20 Partial program showing how to create several informats

```
proc format;
   invalue exp1944fmt (upcase)
     'A' = 220
     'B' = 180
     'C' = 210
     'D' = 110
     'E' = 90
   other = .;
   invalue exp1945fmt (upcase)
     'A' = 202
     'B' = 170
     'C' = 208
     'D' = 100
     'E' = 85
   other = .;
   invalue exp1946fmt (upcase)
     'A' = 150
      . . .
run;
```

It takes far less work to create an input control data set (CNTLIN) to create the required informats. You proceed as follows:

Program 21 Creating several informats with a single CNTLIN data set

```
data exposure;
   retain Type 'I' Hlo 'U';
   do Year = 1944 to 1949;
      Fmtname = cats('exp',Year,'fmt');
      do Start = 'A','B','C','D','E';
         End = Start;
         input Label : $3. @;
         output;
      end;
   end;
   drop Year;
datalines;
220   180   210   110   90
202   170   208   100   85
150   110   150    60   50
105    56    88    40   30
 60    30    40    20   10
 45    22    22    10    8
;

title "Creating the Exposure Format";
proc format cntlin=exposure fmtlib;
run;
```

Type = 'I' tells PROC FORMAT that you want to create a numeric informat. The Value of 'U" for the HLO variable tells the procedure to use the UPCASE option on the INVALUE statement. You want informat names of the form:

```
expyyyyfmt
```

where yyyy is a year value from 1944 to 1949. The CATS function concatenates (joins) each of the arguments after first stripping off any leading or trailing blanks. This function also allows numeric arguments and performs the numeric-to-character conversion as well. Here are the first few observations in the EXPOSURE data set:

```
First 10 Observations of EXPOSURE

Type    Hlo     Fmtname      Start    End     Label

 I       U     exp1944fmt      A        A       220
 I       U     exp1944fmt      B        B       180
 I       U     exp1944fmt      C        C       210
 I       U     exp1944fmt      D        D       110
 I       U     exp1944fmt      E        E       90
 I       U     exp1945fmt      A        A       202
 I       U     exp1945fmt      B        B       170
 I       U     exp1945fmt      C        C       208
 I       U     exp1945fmt      D        D       100
 I       U     exp1945fmt      E        E       85
```

By the way, if you want to see selected informats using the SELECT statement of PROC FORMAT, you need to include an @ sign as the first character in the informat name. For example, to list the contents of exp1944fmt and exp1945fmt, you would use:

Program 22 Using a SELECT statement to display the contents of two informats

```
proc format;
   select @exp1944fmt @exp1945fmt;
run;
```

When you use a SELECT statement with PROC FORMAT, it is not necessary to include the FMTLIB option as well.

Continuing with our program. Now that you have an informat for each of the six years, all you have to do is use the year and job code to create an informat name and use that name with an INPUT function to retrieve the exposure. Here is the program:

Program 23 Using user-defined informats to perform a table look-up, using the INPUTN function

```
data read_exp;
   input Worker $ Year JobCode $;
   Exposure = inputn(JobCode,cats('exp',Year,'fmt8.'));
datalines;
001 1944 B
002 1948 E
003 1947 C
005 1945 A
006 1948 d
;
```

You cannot use the INPUT function in this program because the informat name must be a constant. The INPUTN function allows you to create the informat name in the DATA step. By the way, the 'N' in the name INPUTN stands for numeric. You need to tell SAS ahead of time if the informat you are going to create is going to be character or numeric (it can't tell if the name starts with a $ until it is created). There are two corresponding functions, PUTN and PUTC that allow you to create character or numeric format names in the DATA step.

When this program runs, the informat names are created with the CATS function. Each year value creates a different informat name to be used in the INPUTN function. Here is a listing of the resulting data set:

```
Listing of READ_EXP

                  Job
Worker    Year    Code    Exposure

 001      1944     B         180
 002      1948     E          10
 003      1947     C          88
 005      1945     A         202
 006      1948     d          20
```

The last few sections of this paper may seem, at first glance, to be a "bit off the deep end." However, if you try experimenting and writing a few programs of your own using these ideas, you will see how powerful these tools can be.

CONTACT INFORMATION

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

Ronald Cody
P.O. Box 5049
Camp Verde, TX 78010
E-mail: ron.cody@gmail.com