SESUG Proceedings (c) SESUG, Inc (http://www.sesug.org) The papers contained in the SESUG proceedings are the property of their authors, unless otherwise stated. Do not reprint without permission.
SEGUG papers are distributed freely as a courtesy of the Institute for Advanced Analytics (http://analytics.ncsu.edu).

SAS® Formats, More Than Just Another Pretty Face

Ying Liu, Toronto, ON, Canada

ABSTRACT

Most of us are familiar with using SAS® formats to change the display of data values in reports; most of us have used constructs like: • put totalAmount dollar12.2; or • put orderDate yymmdd10.; However, SAS formats can do far more than this. In this paper I will start with a review of some of the built in SAS formats. This will be followed by an introduction to PROC FORMAT, showing you how you can create your own formats. From there the paper will take PROC FORMAT a step further by showing how formats can be built dynamically from your existing data. Finally, a common but powerful use of formats will be demonstrated – using formats to create efficient lookup tables. The paper is aimed at intermediate SAS programmers who have yet to discover how a pretty face can make a program turn its head.

INTRODUCTION

It is not uncommon for us to have to report the same data in different ways. For example, the company financial officer for the Americas wants to see numbers separated by commas and a decimal point separating the pennies (e,g. 1,234.55), but her counterpart in France wants the French style with numbers separated by the decimal point and a commas separating the pennies (e.g. 1.234,55). As SAS programmers we know that we can simply apply a different format to the data to get a different report output; that is, we can easily convert the underlying data representation to an appropriate visual representation. As SAS programmers we are also aware there are sorts of built in formats that come with SAS that cover a wide range of requirements. However, no matter how many formats that are supplied with SAS, there will always be the need to apply a custom format – a different visual representation of our underlying data.

One common way creating a new visual representation of the data is to create classification groups. For example, test scores could be grouped into letter grades or respondent ages into age groups. Many times we only need to display the data in this new representation. However, sometimes we want to do some analysis or summaries on these classifications, not the underlying interval or continuous data. Some SAS procedures (e.g. PROC MEANS) can use interval/continuous data that use these formatted values as classification variables, but there will still be times we need to use the classifications, forcing us to create a new data variable based upon the these classifications.

In this paper, I will introduce the method (PROC FORMAT) to create your own formats, user defined formats, then show how to apply the formats. When applying these user defined formats I will show not only how to use them to change the displayed value but also how to use user defined formats as a powerful lookup tool.

INTERNAL DATA REPRESENTATION

Before we look at how data can be displayed – the visual representation, let's look at the how SAS manages the underlying or internal data representation. SAS has a very simple underlying data model; variables can be either numeric or character. Obviously the final order of bits and bytes will be determined by the platform and operating system, but for our purposes number and character will suffice. By applying formats we can achieve a rich display set from these two simple types.

VISUAL DATA REPRESENTATION

Although there are numerous internal formats supplied with SAS, most of us are familiar with the formats for numeric data (e.g. comma9.2) and dates (e.g. yymmdd10.), so lets see how these common formats work. First, we can apply the format 'permanently' when we create the dataset. By doing this, every time the variable is displayed, it will be displayed using the assigned format; in essence, this creates a default visual representation. This approach is commonly taken when using SAS dates; since SAS dates are simply the number of days since 1 Jan 1960, displaying a value like 31OCT2006 is far better than displaying the underlying numeric value of 17105.

We can also dynamically apply the format, that is, at the time the data are displayed; for example, in a PROC PRINT we can specify a format for any/all of the variables. What is valuable about applying the formats in this way is that we

can apply a format that is appropriate to the intended target audience. Moreover, a format applied like this overrides the format that may have been permanently assigned to the variable.

NUMERIC FORMATS

The numeric data in a SAS data set includes length and format attributes. The length attibute defines the number of bytes the SAS system to store the data. Numeric data has a default length of 8 bytes. The format attibute tells the SAS system how to present the data.

There are two common formats for displaying our usual numbers such as quantities, measures and dollar amounts, and the formats for displaying dates in the user recognized format. Let's look at each format.

NUMERIC DATA

Basic numeric formats have two common components. The width of the output and the number of decimal places. The SAS system uses floating-point representation referred to us as W.D, where W is the width and D is the number of digits to the right of the decimal place. As an example 8.2 will allocate a total of 8 spaces for the output. One space will be for the decimal and 2 for the digits to the right of the decimal; this will leave 5 spaces for the digits to the left of the decimal. In the below table, numeric data 12345.99 is demonstrated how to allocate digits and decimal in the SAS system using 8.2 format.

1	2	3	4	5	.	9	9

There are 3 common types of internal formats for outputting numeric data:

- simple W.D format: described in the above example of 8.2 format. The SAS system writes the number in the total of W spaces, one space for decimal and D spaces for the numbers of decimals.
- commaW.D format: as an example of comma8.2 will allocate a total of 8 spaces for the output. 1 space is allocated for the decimal, 2 spaces for the number of decimals and 1 space for comma as a separator in every 3 digits. The below is an example for the number 1,234.99 using 8.2 format.

1	,	2	3	4	.	9	9

- dollarW.D format: in an example of dollar8.1 will allocate a total of 8 spaces for the output. 1 space is reserved for the decimal, 1 spaces for the number of decimals and 1 spaces for a dollar sign and 1 space for comma as a separator in every 3 digits. Here we display the number $1,234.9 in 8.1 format.

$	1	,	2	3	4	.	9

The difference between these above 3 formats can also be illustrated on the variable income in the following SAS codes:

```
DATA DISPLAYINCOME;
```

```
    INCOME1 = 3500.6789;  PUT INCOME1 7.2;
    INCOME2 = 3500.6789;  PUT INCOME2 comma8.2;
    INCOME3 = 3500.6789;  PUT INCOME3 dollar9.2;
RUN;
```

As we can see from the output of this data step, if we want to display the value of income with 2 decimals, the width W of the output requires differently using these 3 types of internal formats

income1	income2	income3
3500.68	3,500.68	$3,500.68

We also notice that in the either simple W.D format, commaW.D format, or dollarW.D format, when the width is too small for the number to be printed, the decimal gets shifted by the "BEST" format.

The formatting sequence is as follows:

- Simple W.D format: whole number, decimals.
- CommaW.D format: whole number, decimals, comma.
- DollarW.D format: whole number, decimals, dollar sign, comma.

The following table illustrates how the base value is presented sequentially:

Base Value	FORMAT	Representation Value
3500.6789	6.2	3500.7
3500.6789	7.2	3500.68
3500.6789	comma6.2	3500.7
3500.6789	comma7.2	3500.68
3500.6789	comma8.2	3,500.68
3500.6789	dollar6.2	3500.7
3500.6789	dollar7.2	3500.68
3500.6789	dollar8.2	$3,500.68
3500.6789	dollar9.2	$3,500.68

DATE DATA

The SAS system stores and displays dates in numbers unless we explicitly specify to represent the dates in the user recognized date formats.

The following DATA STEP applies the different date formats in the data representation.

```
DATA DISPALYDATE;
    REPORTDATE = '31Oct2006'd;
    PUT REPORTDATE;
    REPORTDATE1 = REPORTDATE; PUT REPORTDATE1 yymmddN8.;
    REPORTDATE2 = REPORTDATE; PUT REPORTDATE2 yymmN6.;
    REPORTDATE3 = REPORTDATE; PUT REPORTDATE3 date9.;
    REPORTDATE4 = REPORTDATE; PUT REPORTDATE4 date7.;
    REPORTDATE5 = REPORTDATE; PUT REPORTDATE5 yymmdd10.;
    REPORTDATE6 = REPORTDATE; PUT REPORTDATE6 mmddyy8.;
RUN;
```

The results from the above data step show that reportDate only displays a number if we do not explicitly use PUT statement with the specific date format that we want to review.

reportDate	reportDate1	reportDate2	reportDate3	reportDate4	reportDate5	reportDate6
17105	20061031	200610	31OCT2006	31OCT06	2006-10-31	10/31/06

USER DEFINED FORMATS

User defined formats are formats that create representations over and above those that the SAS system provides (e.g. internal formats). User defined formats are created using PROC FORMAT.

There are two common types of user defined formats.

- **Binary Values:** for example, we often refer 0/1 to YES/NO, or Female/Male.

```
PROC FORMAT
    VALUE SEXFMT
    0  - 'Female'
    1  -  'Male'
    ;
```

- **Multiple Values:** we often see in the questionnaire, 0 refers to YES, 1 means NO and 2 implies UNKNOWN. This type of problem is easily accomplished though the PROC FORMAT. The following code generates format ANSWER.

```
PROC FORMAT;
    VALUE ANSWER
    0  -  'YES'
    1  -  'NO'
    2  -  'UNKOWN'
    ;
```

PROC FORMAT SYNTAX

Syntax:

```
PROC FORMAT
   VALUE <$> format-name
          Format-values = formatted-value
;
```

PROC FORMAT NAME

Format-name: There are 2 types of format names:

- Character format name
- Numeric format name

The naming requirements:

- Character format name must start with $ as the first letter.
- A format name is composed of a mixed letters, numbers and underscore.
- The first and last letters of format name can not be a number.
- The format name can not be a SAS reserved function name.

The length requirements:

- SAS V8: The length can not exceed 8 characters including $ for character format name.
- SAS V9: The length can be up to 32 characters.

PROC FORMAT VALUES

Format-values: There are 3 types of the format values:

- A single value
  - Numeric: 88 or **.** (missing)
  - Character: 'AA' or ' ' (blank) or ''(missing). If the quotes are missing, PROC FORMAT defaults single quotation to character values.

- A value range set with a comma separated.
  - Numeric: 1, 2, 3
  - Character: 'A', 'B', 'C'

- A range of values:
  - Numeric: 111-999
  - Character: 'A' – 'Z'. Be sure to enclose each character string with the single quotation marks. Otherwise, the string will be interpreted as single character value.

Note: Each value or range can be up to 200 characters.

The below is the table of Moody's Ratings:

Categories	Moody's Ratings
Exceptional	**Aaa, Aaa1, Aaa2, Aaa3**
Excellent	**Aa, Aa1, Aa2, Aa3**
Good	**A, A1, A2, A3**

Adequate	**Baa, Baa1, Baa2, Baa3**
Questionable	**Ba, Ba1, Ba2, Ba3**
Poor	**B, B1, B2, B3**
Very Poor	**Caa, Caa1, Caa2, Caa3**
Extremely Poor	**Ca, Ca1, Ca2, Ca3**
Lowest	**C**

Moody's ratings can be classified into different categories through PROC FORMAT:

```
PROC FORMAT;
    VALUE $grade
    'Aaa', 'Aaa1', 'Aaa2', 'Aaa3' = 'Exceptional'
    'Aa', 'Aa1, 'Aa2','Aa3'       = 'Excellent'
    'A', 'A1', 'A2', 'A3'         = 'Good'
    'Ba' - 'Ba3'                  = 'Questionable'
    'B'  - 'B3'                   = 'Poor'
    'Caa', 'Caa1' - 'Caa3'        = 'Very Poor'
    'Ca', 'Ca1'   - 'Ca3'         = 'Extremely Poor'
    'C'                           = 'Lowest'
;
```

We can see that the above PROC FORMAT code applies these 3 types of format value.

- A single value: 'C'
- A character range set separated by a comma:
  - 'Aaa', 'Aaa1', 'Aaa2', 'Aaa3'
  - 'Aa', 'Aa1, 'Aa2','Aa3'
  - 'A', 'A1', 'A2', 'A3'
- A range of character values separated by a hyphen:
  - 'Ba' - 'Ba3'
  - 'B' - 'B3'
- The mixed single value with a range of values:
  - 'Caa', 'Caa1' - 'Caa3'
  - 'Ca', 'Ca1' - 'Ca3'

PROC FORMAT DISPLAY

Formatted-value: always are character strings regardless of the format type.

- If the quotation marks are missing, PROC FROMAT defaults the single quotation marks to character strings.
- Similar to format ranges, the formatted values can be up to 200 characters.
- If a formatted value contains a single quotation mark, enclose it with two separate single quotation marks.

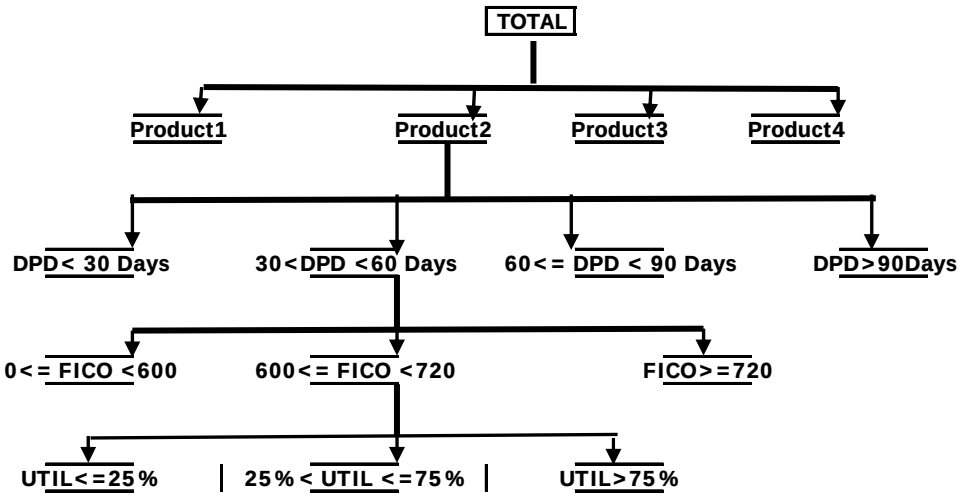
The following is an example of a formatted value containing a single quotation mark.

```
PROC FORMAT;
    VALUE GRADE
    1 = 'LEVEL1''S CUSTOMER'
    2 = 'LEVEL2''S CUSTOMER'
    3 = 'LEVEL3''S CUSTOMER'
    ;
```

DECISION TREE

We have been looking at creating formats and using formats to change how data are displayed, now let's look at how we can use formats to assist us in some of our data management problems. The example I will use it creating a decision tree.

When building probability of default or expected loss given default segmentations for BASEL, the decision trees are often applied to split nodes in the segmentation process. Here is the example of the decision tree.



From the above tree, we can see that there are 4 levels of split. 1st split is by product type, 2nd split is by days past due (DPD), 3rd split is by FICO and last split is by utilization (UTIL). Each level of split tries to group and classify customers into the specified nodes by the ranges defined in the tree.

As an example of days past due, from the below table, we can see that we want to group different range of DPD into the delinquent bucket 00, 01, 02 and 03.

Days Past Due (DPD)	Delinquent (Bucket)	Bucket Description (Bucket_Desc)
0 – 29	00	Not Due
30 – 59	01	30 Days Past Due
60 – 89	02	60 Days Past Due
90+	03	90+ Days Past Due

This task can be often accomplished by IF/THEN/ELSE statement in the DATA step. Here is the SAS code using IF statements to generate variable Bucket.

```
DATA DEFBUCKET;
    SET ALLPRODUCT1;
    IF DPD >= 0 AND DPD < 29          THEN BUCKET = '00';
    ELSE IF DPD >= 30 AND DPD < 59    THEN BUCKET = '01';
    ELSE IF DPD >= 60 AND DPD < 89    THEN BUCKET = '02';
    ELSE                                   BUCKET = '03';
RUN;
```

From the above data step, it can be found that although IF statement could provide the results, it also generates some problems. Each time if DPD range is changed, programmers have to go the program and explicitly modify the codes. If this code appears several times in the program or in multiple programs, it will increase the time to make the modifications, and with many changes and with many changes there is an increasing chance of generating errors.

PROC FORMAT can provide the same results by using the PUT function. It requiresd less code and can be easily be maintained. When the change is required, a programmer does not need to go through the whole program, but simply updating the range of DPD in the PROC FORMAT statement. Here is the code using PROC FORMAT statement:

```
PROC FORMAT;
    VALUE DEL
    0  - 29    = '00'
    30 - 59    = '01'
```

```
    60 - 89    = '02'
    90 - HIGH = '03'
    ;
```

After defining the range in PROC FORMAT statement, we execute the below data step with the PUT statement, all customers that purchased product1 are grouped into 4 delinquent buckets.

```
DATA DEFBUCKET;
    SET ALLPRODUCT1;
    BUCKET = PUT(DPD, DEL.);
RUN;
```

FORMAT has lots of usages. Besides using formats to change the displayed value of data, we also use formats as part of our data management toolkit. In this particular decision tree subject, we are going to discuss how we use formats to group data, identify missing data and transform data

GROUPING DATA

PROC FORMAT groups data in the meaningful distributions. There are 4 different ranges for the format values:

- Start – End : including both start and end points.
- Start <- End: excluding start point and including end point.
- Start - < End: including start point and excluding end point.
- Start <- < End: excluding both start and end points.

.

In the level4 of the decision tree, we can see that we need to group utilization into low, medium and high utilization groups. Here is the table information for utilization.

Utilization	Utilization Node	Utilization Description
UTIL <= 25%	00	Low Utilization
25% < UTIL <= 75%	01	Medium Utilization
UTIL > 75%	02	High Utilization

The utilization description in the above table will be simply represented using the following PROC FORMAT statement:

```
PROC FORMAT;
    VALUE  UTIL
    LOW   - 25    = 'Low Utilization'
    25 <  - 75    = 'Medium Utilization'
    75 <  - HIGH  = 'High Utilization'
    ;
```

IDENTIFYING MISSING DATA

If format values are ranges, the ranges can be specified with the special key words:

- LOW: define the lowest value.
  - For numeric formats, LOW does not format missing values.
  - FOR character formats, LOW includes missing values and blank values.
- HIGH: define the highest values.
- OTHER:
  - For character formats, OTHER defines other non-missing values.
  - For numeric formats, OTHER includes missing values.

In the level 3 split of the decision tree, FICO contains missing data information. The missing FICO and FICO range d from 0 to 600 (excluding 600) is required to be grouped into one node 01. The following table contains FICO range and its description:

FICO	FICO Node	Fico Description
Missing	01	Missing, 0 <= Fico < 600
0 <= FICO < 600	01	
600 <= FICO < 720	02	600 <= Fico < 720
FICO >= 720	03	Fico >= 720
Other	99	Error

Although LOW and OTHER can be used to define missing values for character formats and numeric formats, it is always safe to explicitly specify the dot (.) for missing numeric values and the '' or ' ' for missing or blank character values. The following codes explicitly specify missing instead of using key words.

```
PROC FORMAT;
    VALUE FICO;
    .           = '01'
    0    -  <600 = '01'
```

```
    600  -  <720 = '02'
    720  -  HIGH = '03'
    OTHER       = '99'
    ;
RUN;
```

Note: keep in mind if your data includes other missing values such as .A, .B, …, .Z, you should explicitly declare each missing format value.

TRANSFORMING DATA

Once data is grouped in the large data set, we want to represent data in the easy understanding format in the report.

The following codes generate the report based on the base data information:

```
PROC FREQ DATA=ALLPRODUCT1;
   TABLES DPD  /NOCOL;
   FORMAT DPD DEL.;
RUN;
```

The results display the basic data information for the delinquent bucket.

Bucket	Frequency	Percent
00	85000	58.95%
01	35000	24.27%
02	15000	10.40%
03	8000	6.38%

However, if people do not have knowledge or forget what each bucket means, the above report does not have any business meaning. Therefore, it is important to interpret the numbers in words. The PUT function can complete this data transformation. It creates a new variable BUCKET_DESC and transforms the variable BUCKET though the statement **BUCKET_DESC = PUT (BUCKET, $DPDDESC.):**

```
PROC FORMAT;
   VALUE  $DPDDESC
   '00' = 'Not Due'
   '01' = '30 Days Past Due'
   '02' = '60 Days Past Due'
```

```
    '03' = '90+ Days Past Due'
    ;
```

```
DATA DPDDATA;
    SET ALL PRODUCT1;
    BUCKET      = PUT(DPD, DEL.);
    BUCKET_DESC = PUT(BUCKET, $DPDDESC.);
RUN;
```

Now when we execute PROC FREQ statement in the below, the report shows more meaningful information about the distribution of customers in the different range of their days past due status.

```
PROC FREQ DATA=DPDDATA;
    TABLES BUCKET_DESC/NOCUM;
RUN;
```

Bucket	Frequency	Percent
Not Due	85000	58.95%
30 Days Past Due	35000	24.27%
60 Days Past Due	15000	10.40%
90+ Days Past Due	8000	6.38%

CONTROL DATA SET

As we can see, all the variable information in the decision are listed in the tables. Instead of creating a format for each variable using PROC FORMAT, we can create efficient lookup tables to generate formats.

The SAS system generate both numeric and character formats though a SAS data set known as a cntlin data set.

A simplified cntlin data set looks like:

FMTNAME	START	END	LABEL	SEXCL	EEXCL	TYPE
$PROD	PRODUCT1	PRODUCT1	00	N	N	C
$PROD	PRODUCT2	PRODUCT2	01	N	N	C
$PROD	PRODUCT3	PRODUCT3	02	N	N	C
$PROD	PRODUCT4	PRODUCT4	03	N	N	C

The **Mandatory** fields in the above input data set will be FMTNAME, START and LABLE. FMTNAME is either numeric or character formats. The rule of naming FMTNAME follows the naming requirement described in the section PROC FROMAT NAME. START is the beginning of the range of the format value. Label is the description of the format value. In the above control data set, each product is encrypted into the unique product code. PRODUCT1 is represented by 00.

Among each field in the above product data set, the **Optional** fields are in the following:

- END: since the format value is a single value, END variable can be omitted. However, we must specify the END variable if the input values are in the range;
- TYPE: must specify a TYPE variable with the value C since FMTNAME is a character format; the SAS system defaults N to a numeric format.
- SEXCL and EEXCL: SEXCL means START value is excluded; EEXCL implies END value is excluded. The SAS system defaults inclusion N to both SEXCL and EEXCL. In the above example, these two fields are optional since the range values must be included. However, if the range values are noninclusive, the both fields must be specified with a value of Y.

The above control table can be accomplished using PROC FORMAT CNTLIN statement:

```
PROC FORMAT CNTLIN = FMTS.PRODUCT;
RUN;
```

Then we can review the PROD format by executing PROC FORMAT CNTLOUT statement.

FMTNAME	START	END	LABEL	MIN	MAX	DEFAULT	LENGTH	FUZZ	PREFIX	MULT	FILL	NOEDIT	TYPE	SEXCL	EEXCL	HLO	DECSEP	DIG3SEP	DATATYPE	LANGUAGE
PROD	PRODUCT1	PRODUCT1	01	1	40	2	2	0		0		0	C	N	N					
PROD	PRODUCT2	PRODUCT2	02	1	40	2	2	0		0		0	C	N	N					
PROD	PRODUCT3	PRODUCT3	03	1	40	2	2	0		0		0	C	N	N					
PROD	PRODUCT4	PRODUCT4	04	1	40	2	2	0		0		0	C	N	N					
PROD	**OTHER**	**OTHER**	99	1	40	2	2	0		0		0	C	N	N	O				

CONCLUSION

As we saw SAS formats can do far more than just change the look of the display. We saw how the use of formats simplify our programme logic, following that we saw how we can future simplify our programme by creating our formats from data files. The results of this means we can quickly and easily transform the information from the complex trees through simple table management process.

ACKNOWLEDGMENTS

I would like to thank Mr. Peter Eberhardt. He reviewed the paper carefully and his insights and suggestions helped me to create this paper.

CONTACT INFORMATION

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

Ying Liu

HSBC Financial

3381 Steeles Avenue East, Suite 300

Toronto, Ontario M2H 3S7 Canada
Work Phone: (416) 443-3699
Fax: (416) 443-3749
E-mail: ying.x.liu@hsbc.ca

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.