

# Using and Understanding SAS® Formats

Howard Levine, DynaMark, Inc.

## Objectives

The purpose of this paper is to enable you to use SAS formats to perform the following tasks more effectively:

- Improve the Appearance of Reports
- Reduce storage requirements for SAS data sets
- Use formats to perform table look ups [Relational Data Base]
- Use informats to read data
- Convert SAS Formats to SAS data sets and vice versa
- Permanently store SAS formats

## Topics

In order to meet the objectives, these general topics will be expanded upon:

- Types of SAS formats
- Syntax for creating simple formats
- Examples of how to use formats

## Types of SAS Formats

There are three ways that a format can be classified. Every format will have one characteristic from each of the following groups.

### **Character or Numeric**

Just as all SAS data set variables must be either character or numeric, so must SAS formats and informats be character or numeric. Character formats have their names start with a "\$" while numeric formats do not. Formats and informats must be matched with an appropriate type of variable. For example, a numeric variable cannot be displayed by using a character format. SAS would consider that an error.

### **Which statement creates format?**

There are two statements that create formats in SAS - Value and Picture. The Invalue statement creates informats.

#### **Value Formats - Value Statement**

Value formats are used for associating text with a variable value or range. This allows output to be identified more easily. Essentially, value formats are used for recoding one value into another. They are effective when doing table look-ups.

#### **Picture Formats - Picture Statement**

Picture formats allow numeric values to be displayed in a format desired by the programmer. It is powerful because it allows text, numbers, leading zeros, filler, and a variety of other features. All of the formatting can be customized for particular values or ranges of values too.

#### **Informats - Invalue Statement**

Informats are used to read data into SAS data sets. User defined informats can handle unusual data being read into the SAS System.

## Built-In (Part of the SAS® System)

These formats are part of the base SAS product. It includes old standbys such as w.d formats and informats. Also, \$CHAR. and SSN. are other examples. These formats and informats are documented in the SAS Language Guide. They are very valuable to understand.

## User Defined (created with Proc Format)

User defined formats and informats are created with Proc Format. These are the types of formats that we will be most concerned with in this presentation. These formats allow programmers a great deal of flexibility in setting up data bases, performing analysis, and improving the appearance of reports.

## Setting Up User Defined Formats and Informats

Here are some examples of how to define formats that can be used for improving the appearance of reports. The value format SEX is a numeric format used to display 1 as MALE and 2 as FEMALE.

```
proc format;
```

```
value sex
```

```
    1 = 'male'
    2 = 'female';
```

\$GENDER is a character format. It changes an M to male and an F to female.

```
value $gender
```

```
    'M' = 'male'
    'F' = 'female'
    other= 'unknown';
```

The picture format balance is used to add text to output depending on the value of balance. Sometimes, when the NOEDIT feature is used, only text is output with no numbers.

picture balance. This allows picture formats to act like numeric value formats and also have the flexibility to display actual numeric values for other values.

```
low -          < -99999.99 = 'value too low' (noedit)
-99999.99      - <0       = '00,009.99cr' (fill='*'
                        mult=100 prefix='$')
0              = 'zero balance' (noedit)
0<            - 99999.99 = '00,009.99' (fill='*'
                        mult=100 prefix='$')
99999.99< - high      = 'value too high'
                        (noedit);
```

Informats sex and \$gender are designed to read in character strings and convert them into SAS values. Sex converts input to numeric values. \$gender converts input data to other character values. Use of the \_SAME\_ key word allows input values in the range 'A'-'C' to retain their original value.

```
invalue sex 'male'      = 1
            'female'    = 2
            other        = .;

invalue $gender 'male' = 'm'
               'female' = 'f'
               'a' - 'c' = _same_
               other     = ' ' ;
```

## Improving the Appearance of a Report

Using formats to recode values can improve the readability of a report. Here is an example of two prints of a data set with a numeric variable SEX.

Without Format:

```
proc print data=A;
var name sex;
```

Result:

| OBS | NAME   | SEX |
|-----|--------|-----|
| 1   | ELAINE | 2   |
| 2   | JERRY  | 1   |
| 3   | TERRY  | 1   |

With Format:

```
proc print data=A;
var name sex;
format sex sex.;
```

Result:

| OBS | NAME   | SEX    |
|-----|--------|--------|
| 1   | ELAINE | FEMALE |
| 2   | JERRY  | MALE   |
| 3   | TERRY  | MALE   |

|   |                |      |
|---|----------------|------|
| 3 | 70,000 or more | 18.9 |
|---|----------------|------|

Beware!!! The actual value stored in income is the first numeric value encountered for that group.

## Grouping Values for Analysis and Reporting

Often, it is convenient to to group continuous variables such as income and age for reporting purposes. This can be accomplished with formats. Here is an example of how to use a format to compute the average grade completed for people in different income categories.

INDATA Dataset:

| OBS | GRADE | INCOME |
|-----|-------|--------|
| 1   | 13    | 33455  |
| 2   | 17    | 73454  |
| 3   | 15    | 45332  |

etc.

```
proc format;
value incgrp
low - <0 = 'Below 0'
0 = 'zero'
0< - 10000 = 'up to 10000'
10000< - 30000 = '10001-30000'
30000< - 50000 = '30001-50000'
50000< - 70000 = '50001-70000'
70000< - high = '70,000 or more';
```

```
proc summary data=indata;
class income;
format income incgrp.;
var grade;
output out=statout mean=;
proc print;
```

What does the output data set look like?

Result:

| obs | income      | grade |
|-----|-------------|-------|
| 1   | up to 10000 | 11.3  |
| 2   | 30001-50000 | 14.8  |

## Reducing Storage Requirements

Each of these related facts show you how formats can save disk space.

- Formats make your data base more relational.
- Format \$gender allows 1 byte to represent sex of individual.
- If the value is stored as 'MALE' or 'FEMALE', 6 bytes are required.
- Formats used for table look-ups can also save space by storing redundant information only once.
- Changing formats can change printed reports without changing the dataset.

## Table Look-ups (Relational Processing)

Value formats provide us with a very efficient method of performing table look-ups. Whenever a SAS format is needed, it is loaded into memory for the entire data or proc step that is using it. This insures that look-ups will proceed quickly. Also, a unique feature of SAS formats as a look-up tool is that they handle ranges of values very efficiently. Also, the main data set does not have to be sorted by look-up table sort key. These two factors can sometimes make using formats as look-up tables more desirable than merging two SAS data sets with a merge statement.

There are three table look-up applications we will examine: recoding a value, verifying a value is legitimate, and obtaining additional information for an observation based on a search key.

All of the table look-up methods involve using the PUT function in conjunction with the desired format or formats.

## Recoding

The following example will convert letter grades with pluses and minuses to simple letter grades. For example, an A+ will become an A.

```
proc format;
value $newgrd      'a' - <'B' = 'a'
                  'b' - <'c' = 'b'
                  'c' - <'d' = 'c';

data temp1;
    set data;
    newgrade = put(grade,$newgrd.);
```

## Verification

The next example is used to verify that only students having grades of A or B will be kept in data set temp2.

```
proc format;
value $ok      'a' , 'b'      = 'ok'
              other = 'no';

data temp2;
    set temp1;
    if put(newgrade,$ok.) = 'OK';
```

Notice that the \$OK format has an OTHER= condition to catch all non-legitimate values. This is critical because when formats do not get a HIT when trying to look-up a value, the original value being looked-up is returned by the PUT function. If the OTHER= condition had not been used, the code would have worked satisfactorily unless a student had a grade of 'OK'. While such a grade is unlikely to ever occur, we do not want the accuracy of our computer programs at the mercy of the values input into the data set. We want the program to be more stable than that. This is accomplished by using the OTHER= option.

Other key words are HIGH and LOW which may be used as the high or low ends of ranges, respectively.

## Additional Information

If we are using a large data set sorted by another variable, it may be impractical to sort the main data set for doing a standard SAS merge. Also, more than one key might be used such as zip code, type of residence, and income. Furthermore, income will use ranges rather than exact matches. In any of these cases, using the PUT function to "merge" data onto a SAS data set may be more efficient than using standard SAS merges.

Here, we are trying to obtain additional information for our data set based on the zip code of the household.

```
proc format;
value zip      55410 = '40000,3.2,17'
              55411 = '38349,3.4,16'
              etc. ;
```

```
data demograf;
    set hhinfo;
    info = put(zipcode,zip.);
    a v _ i n c                               =
input(scan(info,1,','),10.);
    a v _ h h s z                               =
input(scan(info,2,','),10.);
    a v _ e d                                   =
input(scan(info,3,','),10.);
```

## Informats for Reading Data

When reading in data, it may be desirable to recode the input values in order to save disk space or facilitate future processing. Here we are recoding the variable sex into a number rather than the characters MALE and FEMALE.

```
data sasdata;
input sex sex. name $;

cards;
MALE DAVE
FEMALE JULIE
;
```

```
proc print;
var name sex;
```

Result:

| OBS | NAME  | SEX |
|-----|-------|-----|
| 1   | DAVE  | 1   |
| 2   | JULIE | 2   |

## CNTLIN and CNTLOUT Data Sets

The ability to use SAS data sets as input to Proc Format and to get them as output is an excellent feature of version 6 of the SAS System. This allows data sets to be edited by someone and then fed directly into Proc Format. Proc Format can then later output a data set to be edited if the format needs to be modified. Formerly, to get data from a SAS data set into a format, the SAS macro facility had to be used. This method is much easier. Also, the ability to get an output SAS data set from a format used to be supported only by a SUGI supplemental procedure, Proc FMTLIB. Now, in version 6, this feature is supported by Proc Format. Here is a simple example demonstrating how this feature works.

Please note that the CNTLOUT option will only work when the format is stored in a SAS catalog (version 6 style) rather than in a load library (Version 5 style). Use Proc V5TOV6 to convert load library formats into catalog formats if it is desired to read those formats as input to Proc Format.

```
proc format cntlout=sasuser.zipdata;
value zip 55410 = '40000,3.2,17'
          55411 = '38349,3.4,16'
          etc. ;
```

```
proc format cntlin=sasuser.zipdata;
```

```
/* This second proc format will
generate the same format as the first
*/
```

## Permanently Storing SAS® Formats

SAS formats can be permanently stored from session to session.

## Why?

- Allows formats to exist from one SAS session to another without having to rerun proc format. (Efficiency)
- Keeps routine programs from being cluttered with proc formats.
- Allows formats to be stored in one place to minimize maintenance if a format changes. (This can be accomplished using other methods, but not as efficiently.)
- Only useful if formats will not be updated frequently compared to how often they are used.

## How?

```
LIBNAME LIBRARY 'system name of sas
library';
```

```
PROC FORMAT CNTLIN=SASUSER.ZIPDATA
LIBRARY=LIBRARY;
```

- All formats will be stored in a SAS catalog LIBRARY.FORMATS
- Each format and informat is an entry in the catalog
- Access the formats in other SAS sessions by executing the LIBNAME statement shown above. Make sure that LIBRARY is the libname used and that the system identifier for the library is the same as when formats are created.
- Only the FORMATS catalogs in the WORK and LIBRARY SAS libraries will be accessed to find formats. Unlimited concatenation of formats catalogs available in Version 6.07.

## Syntax

There are two parts to using SAS formats. One is creating and manipulating them. This is accomplished with Proc Format.

The other part is using them. That is accomplished with put and input statements as well as with the PUT function as we have previously seen. A statement that has a profound impact on the use of a format is the format statement. The format statement in a data step permanently associates a format with a variable. In a Procedure, it temporarily associates a format with a variable just for the duration of the procedure. It will override an existing, permanent format association.

## Proc Format

```
proc format [cntlin=SAS-data-set
             cntlout=SAS-data-set
             library=library fmlib page];
```

```
value format name {specification} ;
```

```
invalue informat name {specification}
;
```

```
picture format name {specification} ;
```

```
select entry-list ;
```

```
exclude entry-list;
```

## Format Statement

```
format variable-list format-name ;
```

Example:           format income revenue  
                    incgrp.;

Create Data Set-   Permanently  
                    associates   format  
                    with   data   set  
                    variable

Use Data Set-       Associates   format  
                    with a variable only  
                    for duration of the  
                    Proc.   Overrides  
                    format permanently  
                    associated   with  
                    variable on data set.

## Data Conversions

Converting values from character to numeric, and vice versa, can be made substantially more efficient by using the PUT function with an appropriate SAS format. Here is an example of how to set this up in a data step.

```
data temp1;
    x0 = 5;
    xc = left(put(x0,3.));
    /* number to character */

    x1 = input(xc,3.);
    /* character to number */

    output;
    stop;

run;
```

## Details and Limitations

There are a few details you need to be aware of in order to avoid silly mistakes that could be costly. Note that many limitations have been eased for version 6.07.

- Left side of format can contain up to 16 characters (200 Characters for version 6.07)
- Right side of format can have up to 40 characters (200 Characters for version 6.07)
- Formats with more than 5000 items become unwieldy to run. 10,000 items is a practical limit. In version 6.07, the sizes of your formats are limited by available memory. Still, the larger the number of items, the more costly it is to set up a format.
- The NOFMterr option can be set to allow the FORMAT NOT FOUND error to not stop all processing in a SAS session.
- Overlapping values of the keys is not allowed.

Format names must conform to SAS naming standards. Also, they must be 7 or fewer characters and must not end with a number. Character formats must have a \$ in front of the format name, but the \$ does not count as part of the name. It is a required prefix for character formats and informats.

## References

- SAS Language Guide, SAS Institute Inc.
- SAS Procedures Guide, SAS Institute Inc.
- SAS Technical Report P-222 Changes and Enhancements to Base SAS® Software, SAS Institute Inc.

## Conclusion

SAS formats are powerful tools and are even better in version 6.07. They are useful for improving the appearance of output, grouping values for analysis, recoding values, performing data validations, and "merging" information onto data sets.

## Author

If you have any questions or comments, the author may be contacted at

DynaMark, Inc.  
4290 Fernwood St  
St Paul MN 55112-5730  
612-486-1793