USING FORMATS TO ENHANCE THE APPEARANCE OF YOUR REPORTS

Steve Schulz
Independent Contractor

Introduction

Formats are one of the most basic elements of any computer language. All languages require some method to display data that has been stored in the computer as binary digits and to recognize the form of incoming data. Few of us can make sense of data displayed in binary representation and, as a result, we can appreciate the much easier and more presentable ways to display data. Formats are among the simplest tools in the SAS system, but like many simple tools, they can be deceptively powerful. Not only can formats improve the appearance of reports, they can also improve efficiency in both processing time and data storage. This tutorial covers using formats for both input and output, creating your own custom formats, creating libraries of your custom formats, using formats to optimize and standardize your programs and to improve reports from both DATA and PROC steps.

FORMATS

As stated above, formats serve two primary functions. One of these is to facilitate getting data into the computer. In the SAS system, INFORMATS are used to accomplish this task. When reading data into a SAS dataset, an INFORMAT associates a format with a variable in the incoming data stream. This INFORMAT may be a default supplied by the system, it may be associated with the variable with a FORMAT statement, or it can be explicitly placed in the INPUT statement. Only if the INFORMAT statement is used is the INFORMAT permanently associated with the variable which is useful when using the interactive edit and browse facilities, PROC EDITOR, PROC FSEDIT and PROC FSBROWSE. In general, the self documenting nature of explicitly placing the INFORMAT in the INPUT statement is preferable even though an INFORMAT statement may already be present. In a large DATA step with several INPUT statements, an INFORMAT statement may not be immediately apparent. Placing the format on each read of a variable may be redundant, but it makes it absolutely clear how data is being read.

FORMATS

The second function of formats is to output data. These are called FORMATS in the SAS system. There are analogous situations for output to those described above for input. Again there are three ways in which a FORMAT can be assigned for output. If the user does not specify, SAS assigns a simple list format, a format may have been permanently associated with the variable with a FORMAT statement or the user can explicitly assign a format at output time. While there are only a few ways to read data, there are many ways of writing data. There are explicit PUT statements in DATA steps which are the complement of INPUT statements, but there are also many PROCs which report values of data. This is where associating formats with the variable using a FORMAT statement is necessary in order to assign the desired format for reporting that variable in the PROC. A FORMAT assigned in a DATA step is permanent and that format becomes the default to be used whenever that variable is displayed. A FORMAT assigned in a PROC step, however, is temporary and is used only for the duration of that PROC. You may override either the system default format or one that has been permanently assigned to the variable temporarily in this manner.

Built-in FORMATs and INFORMATs

The SAS system provides many FORMATs and INFORMATs for the convenience of the user including formats to read and write in most of the important ways that data are created by computer languages. A FORMAT or INFORMAT consists of three parts. First is the name such as PD for packed decimal. This is followed by the field width in columns which is separated by a period from the number of digits to the right of the decimal. The only format
that does not have a name portion is the simplest numeric format, w.d., which is used for simple list input or output of numbers. The w represents the width of the field and the d represents the number of digits to the right of the decimal. For example, a format of 5.2 is five columns wide with two digits to the right of the decimal. In this format the d or decimal portion is optional if there is no decimal part. A more complicated numeric format, P04., describes a four byte packed decimal field with no decimal places. Exactly how decimals are treated varies between INFORMATS and FORMATS. In INFORMATS, the d is used for specifying implied decimal points in data. See the SAS User’s Guide for further explanation.

Formats for reading and writing character variables are somewhat simpler. There can be no decimal place so the format is specified with just a name and a width. The width is optional in many cases in which case the system decides how to display the data. In most cases the entire field will be displayed. The simplest character format is $w. All character formats begin with a $ which indicates a character string is being formatted wide. The w specifie the width of the field. Other character formats deal with leading blanks, translating to and from hexadecimal representation and varying length strings.

A third class of formats is designed to read and display date and time values in many common arrangements. These translate data to and from the SAS system’s internal data and time representations. There are more FORMATS than there are INFORMATS. Numbers and especially dates can be written in some very elegant ways. For example, numbers up to 9999 can be written in Roman numerals. This would be the format one would use to display Super Bowl game numbers of course. One of the more elegant formats for writing dates, WORDDATE., will display a date as Monday, February 9, 1987.

**PROC FORMAT * PICTURE**

Although many formats are provided, there are many ways to display data that are specific to individual applications and are not included among those in the system. Fortunately the SAS system provides a convenient method for the user to design custom formats. This tool is PROC FORMAT. One of the two kinds of format you can create using PROC FORMAT is PICTURE formats. These are similar to the built-in numeric formats.COMMA., SSN. and DOLLAR. which intersperse other characters in a number making it easier to read and understand. Picture formats are for numeric data only. They allow the user to define a template for displaying the digits of a number.

Take, for example, the PHONENUM. format described in the SAS User’s Guide. This is a nice simple example, but phone numbers are usually displayed with the area code in parentheses, not separated from the rest of the number by a slash. So a better format might be that generated by the PROC FORMAT in the first example below. Note that the structure of a basic PROC FORMAT is quite simple. The PICTURE statement contains five parts. First the keyword PICTURE identifies the type of format being created. PHONENUM. which is the name of the new format comes next. Third, comes the range definition specifying the values for which this

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Width (w)</th>
<th>Decimals (d)</th>
<th>Complete Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric list</td>
<td>na</td>
<td>5</td>
<td>2</td>
<td>5.2</td>
</tr>
<tr>
<td>Packed decimal</td>
<td>PD</td>
<td>4</td>
<td>0</td>
<td>PD4. or PD4.0</td>
</tr>
<tr>
<td>Character</td>
<td>$</td>
<td>10</td>
<td></td>
<td>$10.</td>
</tr>
<tr>
<td>Year month day</td>
<td>YYMDD</td>
<td>6</td>
<td></td>
<td>YYMDD.</td>
</tr>
</tbody>
</table>

**Figure 3 PICTURE formats for phone numbers**

PROC FORMAT;
PICTURE PHONENUM LOW-HIGH = '999) 999-9999'
displays 10 digit numbers as (415) 555-1212
but displays seven digits as (000) 555-1212

PROC FORMAT;
PICTURE PHONENUM 1000000-9999999 = '999-9999'
10000000000-9999999999 = '999) 999-9999' (prefix = '(')
displays 10 digit numbers as above
and displays 7 digit numbers as 555-1212
format applies. In this case the picture is intended to be used for all values and this is indicated by the keywords LOW and HIGH. The fourth part of the statement is the template for displaying numbers. Positions that are to be occupied by numbers have numeric digits in them. The digits 1-9 mean any number, including zero, will be displayed, while a 0 means a leading zero occurring in this position will not be displayed. All other characters are displayed as they occur in the picture. In this example, there is a blank, a close parenthesis and a dash or minus sign. The last part of the statement is the options for this particular template. The options part of the statement is not required. In this example, there is a prefix character to be printed before any of the numbers in our variable value is defined. This prefix is necessary because only display characters to the right of or in between numeric digits are printed. One or two characters to precede the first numeric digit can be displayed using the PREFIX option.

Now suppose local phone numbers are stored without the area code. In this case the number would appear as (000) 555-1212 when printed with the picture just defined. The shorter numbers could be printed using the format PHONENUM8 so that only the eight rightmost characters would be displayed. This would print the local numbers correctly, but would not show area codes for ten digit numbers. A better solution is shown in the second example in Figure 3. Note that the last three parts of the PICTURE statement, range definition, template and options, have been repeated with different specifications. These may be repeated as many times as needed in a PICTURE statement. This allows different templates to be defined for each class of phone number thus allowing them to be printed properly. Now there are separate templates for seven and ten digit numbers.

PROC FORMAT * NOEDIT

Both types of format are useful, but in some cases you may want to use a value type format for some ranges and a picture type for other values. For example, take the PHONENUM8 example. Only seven and ten digit numbers will be printed using the format. Other values will be presented simply as unedited strings of numbers. This is fine and even desirable for certain types of reports, but more elegant formats may be needed for other

Figure 5 An improved PICTURE format for phone numbers

proc format;
picture phonenum
   1000000 = '999-9999' (prefix = '(')
   1000000000 = '999-9999' (prefix = '(')
   other = 'Invalid' (noedit)
   end;
end;
end;
A missing value will print as a single dot using the format created above. This stands out in a column of numbers but may be difficult to see in a large custom report. The first picture in Figure 5 defines a full sized format for missing values with dots appearing where numbers would normally appear. The NOEDIT option is used to prevent an attempt to map numeric digits into the string. The last picture for this format displays the word Invalid for numbers that are not of the correct length to be a phone number. OTHER is the keyword used to define a format for all values of the variable that are not specifically defined. Again the NOEDIT option is used to say do not attempt to interpret the format as a picture, but rather use the exact character string found between the quotes. These features have allowed the definition of PICTURE formats for seven and ten digit numbers and, using NOEDIT, VALUE formats for all other values.

Figure 6 Formats in PROC PRINT

The simplest PROC PRINT generates this type of report

```sas
PROC PRINT;
OBS NAME ADDRESS CITY STATE
1 JOE M. BLOW 1234 MAIN STREET SAN FRANCISCO CA
2 MARY ELIZABETH WILSON 222 N ELM ST ELY NV
OBS SOCSEC BIRTHDAY SEX SALARY KGDATE LTDATE MTDATE
1 123456789 -0379 532500 01/02/78 06/09/83 12/15/84
2 111223333 -3130 42950 07/02/80 07/07/83 10/31/86
```

Adding formats improves the appearance and readability

```sas
PROC PRINT;
FORMAT STATE $STATE. SOCSEC SSN. SEX $SEX. SALARY DOLLAR8. BIRTHDAY MMDDYY8. KGDATE LTDATE MTDATE MMDDYY8.;
OBS NAME ADDRESS CITY STATE
1 JOE M. BLOW 1234 MAIN STREET SAN FRANCISCO CALIFORNIA
2 MARY ELIZABETH WILSON 222 N ELM ST ELY NEVADA
OBS SOCSEC BIRTHDAY SEX SALARY KGDATE LTDATE MTDATE
1 123-45-6789 08/08/49 MALE 832,500 01/02/78 06/09/83 12/15/84
2 111-22-3333 06/07/51 FEMALE 842,950 05/22/80 07/07/83 10/31/86
```

Formats can help limit the size of fields on the print line

```sas
PROC PRINT;
ID SOCSEC;
VAR NAME CITY STATE SEX BIRTHDAY KGDATE LTDATE;
FORMAT CITY $5. SOCSEC SSN. SALARY COMMA6. BIRTHDAY MMDDYY8.;
SOCSEC NAME CITY STATE SEX SALARY BIRTHDAY LTDATE
123-45-6789 JOE M. BLOW SAN F CA M 32,500 08/08/49 06/09/83
111-22-3333 MARY ELIZABETH WILSON ELY NV F 42,950 06/07/51 07/07/83
```

Things you can't do

One type of format that cannot be created using PROC FORMAT is date formats. These are much more complex than PICTURE or VALUE formats. Relatively complicated algorithms are used to derive dates from the numbers in which they are stored. Fortunately, a wide variety of formats are provided and one of these should meet most needs. In addition, there is not a PROC INFORMAT to complement PROC FORMAT, so custom input formats cannot be defined with the ease PROC FORMAT provides for output formats.

PROC PRINT

Possibly the easiest procedure to use in SAS is PROC PRINT. The minimum code necessary is just the name of the procedure. This is adequate for many purposes, especially for small datasets, but there are instances where you need to print a large dataset, however, and PROC...
PRINT can be very slow and output inconsistent and difficult to read in these situations. The reason for this is that left to its own devices, PROC PRINT lays out each page independently which can be expensive if there are a large number of pages to print. One can prevent this by using the UNIFORM option which causes all pages to be printed in the same manner. However, PROC PRINT then assumes all fields may reach maximum value and assigns field widths that are large enough to handle this eventuality. This usually causes the report to be spread out and wasteful. Explicitly assigning formats can prevent this problem. For example, if you have a numeric variable with two digit values called AGE, the statement (FORMAT AGE 2.) will cause the field in which it is printed to be two columns wide rather than as many as 17 columns that would be assigned normally by PROC PRINT.

Frequently PROC PRINT is used as a reporting vehicle in a debugging or exploratory mode. In these cases, it is often useful to set as much information as possible on one line. Again assigning formats explicitly can help. Variables that contain names and addresses tend to be long character fields and these take up a lot of space on a print line. For exploratory purposes, some of these fields may be of little interest and so can be omitted from the report. Also, it may not be necessary to display the whole field to be able to discern what the information is. City name is a good example. This may be a variable with a length of as much as 30, but it may be possible to print only the first five or ten characters of the field and still have enough information. Thus a statement such as, FORMAT CITY $10., can remove twenty columns from the printed field and leave that much space to display other data. The third example in Figure 6 demonstrates several techniques to make the output from PROC PRINT consistent and concise. The ID statement causes the variables designated to be printed at the left of each line rather than observation number. This can save a few columns, but also causes more meaningful data to be used as identification in multiple line reports. The VAR statement can be used to select only variables of interest for the current report. In this example, address and two dates are omitted. The FORMAT statement limits the field size for some variables and displays other fields in a more presentable manner. SSN. is a built-in format that displays social security numbers in the customary manner. The $5. format assigned CITY limits the number of columns this field occupies with considerable savings. The COMMA. format assigned to SALARY makes it easier to read, but also limits the size of that field that might be generated when the UNIFORM option is used.

PROC FREQ

Another of the most frequently used procedures is PROC FREQ and formats can be used to improve the appearance of reports generated by this procedure also. The formats created in Figure 4 are fine for PROC PRINT or one dimensional PROC FREQs, but in a two dimensional table they may not look good as column labels. This is because PROC FREQ will divide them into two lines after eight characters. A solution to this is to have special formats to use on the top of PROC FREQs that will make good column headers.

**Figure 7** Column heading formats for PROC FREQ

```plaintext
PROC FORMAT;
VALUE STATES 'ID' = 'IDAHO' 'NC' = 'NORTH CAROLINA' 'UT' = 'UTAH' 'MD' = 'MARYLAND' 'NV' = 'NEVADA' 'VA' = 'VIRGINIA';
```

This format is used for column labels in the following table:

<table>
<thead>
<tr>
<th>TABLE OF ORIG BY DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIG</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>IDAHO</td>
</tr>
<tr>
<td>MARYLAND</td>
</tr>
<tr>
<td>NORTH CAROLINA</td>
</tr>
<tr>
<td>VIRGINIA</td>
</tr>
<tr>
<td>TOTAL</td>
</tr>
</tbody>
</table>

107
labels. This is done by centering the text in eight column sections. The example in Figure 7 shows a format that works well on a two dimensional table. Text is divided into eight character sections so that for the value NC the first line will be NORTH with two blanks on the left and one on the right while the second line will be CAROLINA. With a little care, many labels can be divided easily. Words longer than nine characters still present problems that can only be addressed with abbreviation or hyphenation.

Formats also allow you to combine more than one data value into a single cell in your tables. This is because PROC FREQ generates its cells by the values to be displayed, the formats, not the data values in the dataset. This means the user can control the grouping by using FORMAT statements. If the formats defined above are used to display a state variable, different reports can be generated using the same variable but changing the format statement. It is possible to generate tables by state or by region by using different formats with the same variable. In some cases, it will be possible to have fewer variables on the dataset using this technique.

One thing to remember though is that PROC FREQ displays and groups by only the first sixteen characters of the value to be displayed. Therefore if there is a 25 character field and there is more than one value of that field in which the first sixteen characters are the same, they will be treated as equal values by PROC FREQ. This holds true whether actual data values or formatted values are being displayed.

PROC TABULATE

PROC TABULATE is a much more complicated and much more elegant table generator than PROC FREQ, but formats can be used in many of the same ways. Again formatted values can be used for grouping and for row and column labels. The user has more control over the size of cells in this procedure, so the eight characters per line of label is not a limitation here nor is the sixteen column total label length a problem. The procedure automatically divides labels at blanks and attempts to format labels logically. In addition to the things that can be done in PROC FREQ, the user can format values being reported in the cells and this can contribute greatly to the appearance of the tables. In version

Figure 8 A format to display disparate values

```
proc format;
picture brief (max = 8)
  0-999 = '009' 
  1000-999999 = '009.999' (mult = .1)
  1000000-9999999 = '009.9999' (mult = .0001)
  100000000-999999999 = '009.99999' (mult = .000001);
```

```
proc tabulate data = data.state4
  formchar = 'B3C4DAC28FC3C584COC1D9'X;
class orig dest;
  freq count;
table orig dest * (n*f=brief8.);
  label orig = 'Origin'
    dest = 'Destination';
  format orig dest @state.;
  keylabel n = ' ';
```

Note that PROC TABULATE has divided the state names and the brief8. format used to format cell totals

<table>
<thead>
<tr>
<th>Destination</th>
<th>Maryland</th>
<th>North Carolina</th>
<th>Nevada</th>
<th>Utah</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>9</td>
<td>5.38G</td>
<td>458.74M</td>
<td>459.87K</td>
</tr>
<tr>
<td>Idaho</td>
<td>4.54K</td>
<td>4.55G</td>
<td>346.67M</td>
<td>739</td>
</tr>
<tr>
<td>Maryland</td>
<td>4.35M</td>
<td>45</td>
<td>23.45M</td>
<td>4.55M</td>
</tr>
<tr>
<td>North Carolina</td>
<td>38</td>
<td>7.85G</td>
<td>34.57M</td>
<td>45.56K</td>
</tr>
<tr>
<td>Virginia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
§2.4 the formatting capability amounts to little more than the ability to control the width of the cell, but in versions 5 and 6, you can use COMPARE formats to improve readability of large values. DOLLAR formats if you are tabulating money or you can use your own custom formats. You need not even display numbers in the cells. You might have a format that reports the value 6 as ZERO or all values from 1 to 10 as FEW. The PICTURE format in the example in Figure 8 allows display of widely disparate values in roughly the same size cells. If COMPRA was used for values that vary from one to twelve digits, cell widths of fifteen columns would be necessary. Using BRIEF to keep the size down would work, but large values would be in exponential notation and this is not very pretty. The BRIEF format displays up to five and no fewer than three significant digits for integer values from one to one trillion, is quite readable, and occupies only 7 columns. This can be useful in those situations where it is desirable to fit a large number of cells with widely varying values into a single page report. The DOLLAR option is used to scale larger values down to the appropriate length.

Data Validation

Formats can also be used as very efficient validation tools. By adding one more range to the $STATE.format above we can create a tool to validate incoming data. It would be somewhat cumbersome to write a routine to validate state abbreviations. The classic method is to create an array with the 50 valid values in alphabetical order and write a binary search to validate the data as it comes in. This works, but there is a much easier way using formats. The example in Figure 9 shows how this can be done. The PUT and INPUT functions allow the use of formats to move data between variables as well as to input and output devices. The OTHER value for the format is set to a string that can be easily checked. Then rather than a check to see if one of the fifty valid data values is present, a check for the single invalid formatted value can be made. This is much easier to do and more easily transportable to other programs when needed.

Format Libraries

There is only one option available on the PROC FORMAT statement. Depending on the operating system, it is either the LIBRARY or DECK option. This option allows user defined formats to be saved in a permanent library which makes them accessible to other programs without the necessity of including the code to generate them in those programs. In addition to convenience, this allows a standardization among programs and users. All users can use the same formats for common variables. This eliminates the variations that occur when each user defines his own formats. With a company, or department wide format library with the one set of formats available, all users can be displaying the same values in the same way. Because the generated libraries of formats are system dependent, the exact details how to do this vary. Check the appropriate SAS User’s Guide for exact details on creating format libraries in your environment.

Summary

In review, FORMATS and INFORMATS are used to read and write data in SAS. SAS provides many standard formats, but also provides PROC FORMAT as a tool for the creation of user-defined formats. These can be used in many PROCs and in creating custom reports in a DATA step to improve their appearance and in some cases, their efficiency. Formats also can be used as a data validation tool. Formats are indeed a powerful tool helping in many aspects of data handling and report generation in the SAS system.

* SAS is the registered trademark of SAS Institute Inc., Cary, NC, USA

Author’s address:

Steve Schulz
594 Corbett
San Francisco, CA 94114
(415) 431-8373

Figure 9 A format that can be used for data validation

PROC FORMAT;
VALUE $STATE
'ID' = 'IDAHO'
'MD' = 'MARYLAND'
'NC' = 'NORTH CAROLINA'
'NV' = 'NEVADA'
'UT' = 'UTAH'
'VA' = 'VIRGINIA'
OTHER = 'XX';

which can be used as follows to validate state abbreviations

IF PUT (STATE, $STATE.) = 'XX' THEN
error routine here;
ELSE
normal processing;

109