Picturing Your Character Variables with PROC FORMAT and SAS® 9.3

Brice Hart, Westat, Rockville, MD

ABSTRACT

Version 9.3 of SAS® software provides a way to format a value by first performing a function on the value. In using this feature we can create a customized template that will format character variables much the same way that a picture format is used to create a template for numeric variables. We can create character templates that will insert separators, fill digits, add a prefix, or translate values. Basically, any features of picture formats can be implemented for character variables by creating a user-written function that manipulates the value and then defining a format that calls the function.

This paper demonstrates the steps necessary to create a format that uses functions and how this technique can be used to create a picture-like format for character variables. Example code is given to illustrate each step of the process.

INTRODUCTION

All of us need to create listings and reports that display our variables in an easy-to-read format. When we display phone numbers, we like to wrap parentheses around the area code and insert a dash between the 3-digit exchange and the 4-digit number: (999) 999-9999. Although we could create a new variable that contains the value in the desired format, it is much easier if we can simply associate the original variable with an appropriate format. The PICTURE statement in PROC FORMAT is used to define this type of formatting template of numeric variables.

But what if we have a character variable whose values we want to format with a template in the same manner? Suppose we have a driver's license number stored as "H123456123456" and we wish to print it as "H-123-456-123-456". Since the PICTURE format is only for numeric variables, we cannot use it to create our format.

SAS® 9.3 provides a way to format a value by first performing a function on the value. By using this feature, we can create a user-written function that will manipulate the license value and insert the dashes. We can then define a $LICENSE format that will call the function. The result is that we have a format that will apply our customized template to the character variable.

The steps to formatting a character variable with PROC FORMAT and user-written functions are relatively easy and straightforward:

1. Create the function
2. Make the function available
3. Create the format to call the function
4. Use the format in your code as you would any other format

Let's explore each of these steps in detail and provide examples of each.

EXAMPLE DATA AND OBJECTIVE

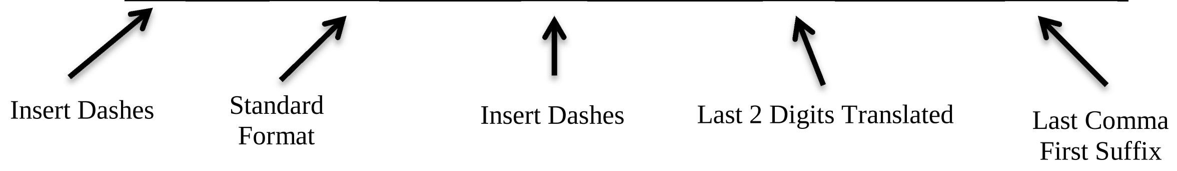
To start with, let us work with a study dataset consisting of the following variables and values:

StudyID	Phone	DriverNum	OfficeNum	Name
10120101	3015551111	A123456123456	123AC	Robert Smith
10120102	6145552222	B987654987654	456AC	Donald Puck
10120103	2025553333	C321321321321	789TR	Madonna

StudyID	Phone	DriverNum	OfficeNum	Name
10120104	9145554444	D789789789789	111AC	John Doe Jr.
10120105	8025555555	E567567567567	222TR	Lisa Mill

We are asked by the project manager to create a listing of the data that looks like the following. Notice how we will need to format each field in our listing.

Study ID	Phone Number	Driver License	Office Number	Name
101-201-01	(301) 555-1111	A-123-456-123-456	Accounting: 123	Smith, Robert
101-201-02	(614) 555-2222	B-987-654-987-654	Accounting: 456	Puck, Donald
101-201-03	(202) 555-3333	C-321-321-321-321	Travel Office: 789	Madonna
101-201-04	(914) 555-4444	D-789-789-789-789	Accounting: 111	Doe, John Jr
101-201-05	(802) 555-5555	E-567-567-567-567	Travel Office: 222	Mill, Lisa



A PROC CONTENTS shows that the variables StudyID and Phone are numeric and the variables DriverNum, OfficeNum, and Name are character.

We can easily print the numeric variables by using PICTURE formats.

```
PROC FORMAT;
  picture IDfmt
      low-high = '999-999-99';

  picture phonefmt (default=14)
      2000000000 - 9999999999 = '999) 999-9999'
      (prefix = '(');
```

To achieve the desired appearance of our character variables, we will need to write code to manipulate the values. Our code will need to insert dashes into the driver's license, translate the office number, and change the order of the parts of the name.

One solution is to program another DATA step and derive new variables that have the desired formatting (dashes, spacing, translations, etc.). After that, we could print the new variables. All other procedures or DATA steps might need to be changed to reference the new derived variables.

Another solution is to create functions to do the value manipulations and then format the variables with the functions. This solution is our objective. Let's see how we do this and walk through each step of the process using some example code.

STEP 1: CREATE A FUNCTION

The first step in creating a format that uses functions is to create the functions. These functions will perform the desired value manipulations. The Function Compiler Procedure (FCMP) is the tool we use to create and store functions and CALL routines. For a detailed discussion on how to build your own functions, please refer to the paper "*User-Written DATA Step Functions*" (Secosky, Jason). The author presents an excellent overview and explains the syntax necessary to use FCMP for building a library of functions.

For our discussion here, suffice it to say that the FCMP procedure uses statements and syntax that are similar to those used in a DATA step. The function is compiled, stored, and then called during execution just as built-in SAS functions are called.

In our example listing, we need to create 3 functions (one for each of the three character variables). The code within each function will manipulate the parameter values to obtain the desired formatting results. Each of these 3 functions will in turn be used by one of our variable formats. Note that to keep the examples simple, we assume that the incoming data are clean, rather than adding OTHERWISE clauses or other code to deal with unexpected values.

Invoke FCMP and specify the library and dataset of where to store the functions.

```
PROC FCMP outlib=work.functions.PICTUREs;
```

This is our License function. DATA step syntax is used within the function to insert dashes between the appropriate digits.

```
function LicenseFunc(DriversNum $) $;
  length LicWithHyphens $17;
  LicWithHyphens = catx('-',substr(DriversNum,1,1),
                            substr(DriversNum,2,3),
                            substr(DriversNum,5,3),
                            substr(DriversNum,8,3),
                            substr(DriversNum,11,3));
  return(LicWithHyphens);
endsub;
```

The Office Number function translates the OfficeNum parameter into the department name and reinserts it as a prefix.

```
function OfficeNumFunc(OfficeNum $) $;
  length OfficeNumFormatted $20;
  select;
    when (substr(OfficeNum,4,2) = 'AC')
       OfficeNumFormatted = cat('Accounting: ',
                                 substr(OfficeNum,1,3));

    when (substr(OfficeNum,4,2) = 'TR')
       OfficeNumFormatted = cat('Travel Office: ',
                                 substr(OfficeNum,1,3));
  end;
return(OfficeNumFormatted);
endsub;
```

Our last function is to process the NAME variable. With this function, our code flips the order in which the names are listed. Instead of First Last Suffix; they become Last Comma First Suffix. Again, DATA step syntax is used within the function to process and manipulate the value.

```
function NameFunc(Name $) $;
  length NameFormatted $20;
  NumNames = countw(Name);
  select;
    when(NumNames = 1) NameFormatted = Name;
    when(NumNames = 2) NameFormatted =
            catx(', ',scan(Name,2),scan(Name,1));
    when(NumNames = 3) NameFormatted =
            cat(scan(Name,2),', ',scan(Name,1),
                ' ',scan(Name,3));
  end;
  return(NameFormatted);
endsub;
```

STEP 2: MAKE THE FUNCTION AVAILABLE

The second step in our process is to make our functions available to our programs.

This is accomplished with the CMPLIB system option.

```
options cmplib=(work.functions);
```

The CMPLIB system option lists one or more datasets that contain subroutines to include during program compilation. The libref.dataset listed in this option is the same libref.dataset given in the outlib= option of the FCMP procedure above. Once the CMPLIB is specified and we have told SAS where our functions are located, they are available for use throughout our session.

STEP 3: CREATE A FORMAT

The third step is to create our formats using the functions we created with the FCMP procedure in step 1. The function is specified within the PROC FORMAT as part of the value-range-set in the value statement. The following is the syntax:

value-or-range-1<..., value-or-range-n>= [USER-DEFINED FUNCTION NAME()]

The function name is given in square brackets. We do not enclose the name in quotation marks. We do not specify the function parameter; the variable that is associated with the format will be the parameter passed to the function.

In our example, we create 3 formats – one for the driver's license, one for the office number, and one for the name.

We are defining each format such that every value of the variable (from low to high) will be formatted with the function result.

```
proc format;
  value $Licensefmt (default=17)
    low-high = [LicenseFunc()];

  value $OfficeNumfmt (default=20)
    low-high = [OfficeNumFunc()];

  value $Namefmt (default=14)
    low-high = [NameFunc()];
run;
```

Other than the call to the function within the value-range-sets, all other rules for defining the format are as usual. One note of caution is to pay close attention to the length of your formats, especially when we are inserting characters (dashes, spaces, etc.). The DEFAULT= value option is used to define the maximum length possible.

STEP 4: USE THE FORMAT IN YOUR CODE

The last and final step is to use our formats in our SAS code. We associate a character variable to our formats and reference them as we would any other format. The same syntax is used.

We associate our formats to our variables using the standard format statement.

```
proc print data=test label noobs;
  format StudyID IDfmt.
         Phone phonefmt.
         Name $Namefmt.
         OfficeNum $OfficeNumfmt.
         DriverNum $Licensefmt.;
run;
```

When each variable is printed, the format passes the raw data values to the function. The function manipulates the values and then returns another value. This value is the formatted value that is printed in our listing.

CONCLUSIONS

Creating a picture-like format for character variables can be accomplished by using a format that calls a function. We create a customized function that manipulates the values of our character variables and then we use the function when defining our format. When a variable is displayed with the format, the raw data value is converted to the value that is returned from the function. The steps we follow to implement this trick are easy and straightforward:

1. Create the function
2. Make the function available

3. Create the format to call the function
4. Use the format in your SAS code as you would any other format

By expanding on this idea, one could develop almost limitless formats. Any feature of the PICTURE format could be replicated for character variables. You are limited only by the complexity of the function code needed to perform the value manipulation.

REFERENCES:

[1]. User-Written DATA Step Functions, Jason Secosky, SAS Institute Inc., Cary, NC
http://www2.sas.com/proceedings/forum2007/008-2007.pdf

[2]. Getting in to the Picture (Format), Andrew H. Karp, Sierra Information Services, Sonoma, CA USA
http://www2.sas.com/proceedings/sugi31/243-31.pdf

[3]. FORMAT Procedure, Base SAS 9.3 Procedures Guide, Second Edition
http://support.sas.com/documentation/cdl/en/proc/65145/HTML/default/viewer.htm#n1c16dxnndwfzyn14o1kb8a4312m.htm

[4]. FCMP Procedure, Base SAS 9.3 Procedures Guide, Second Edition
http://support.sas.com/documentation/cdl/en/proc/65145/HTML/default/viewer.htm#p10b4qouzgi6sqn154ipglazix2q.htm

ACKNOWLEDGMENTS

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.

A special thanks to Mike Rhoads for sharing this idea with me and for the countless suggestions and guidance he has given me. Also thanks to the too-many-to-mention Westat colleagues for the constant assistance, brainstorming, and sharing of ideas.

DISCLAIMER: The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

CONTACT INFORMATION

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

Brice Hart
Westat
1600 Research Blvd.
Rockville, MD 20850
Phone: (301) 294-4411
Email: bricehart@westat.com