Paper 077-2010

Evolution of Formatting "On The Fly"

Allen Blackburn & Mikhail Gruzdev
United States Bureau of Census, Foreign Trade Division, Washington, DC

ABSTRACT

PROC FORMAT procedure is a powerfully productive tool. It allows us to assign labels to values without having to spend time combining data sets, changing values and molding the data to take on the look a user wants. Our SAS program essentially puts a mask on the data. However, sometimes those masks need to change. By formatting 'on the fly', our program saves on the maintenance of manually changing hard-coded PROC FORMAT statements, as well as the computing time of formatting more than necessary when formatting from a data set. Using a PROC FORMAT statement inside of a SAS program has its advantages, but in many cases, depending on how you use it, the statement could be a 'growing' problem. Updating a small number of PROC FORMAT statements is simple, but complexity 'expands' as the number of elements grows larger, with hundreds or even thousands of elements. Through 'evolutionary' use of data sets and SAS formats and data manipulation, SAS programs can save time and resources by creating formats from data sets "on the fly".

INTRODUCTION

PROC FORMAT procedure is a powerfully productive tool. It allows us to assign labels to values without having to spend time combining data sets, changing values and molding the data to take on the look a user wants. Our SAS program essentially puts a mask on the data. However, sometimes those masks need to change. By formatting 'on the fly', our program saves on the maintenance of manually changing hard-coded PROC FORMAT statements as well as the computing time of formatting more than necessary when formatting from a data set.

FIRST EXAMPLE

Depending on how often data labels change, PROC FORMAT users may need to change SAS code every week or every day.

For instance, with the following data set, we can use PROC FORMAT and PROC PRINT to display the category variables COUNTRY, PRODUCT and DISTRICT with descriptions.

COUNTRY	PRODUCT	DISTRICT	IMPORTS	EXPORTS
4270	100	04	1300	0
4270	100	04	2300	0
4270	100	30	550	0
4270	100	30	5500	2000
4270	100	31	1150	0
4270	100	31	4150	0
4270	200	04	10000	4500
4270	200	04	11000	1500
4270	200	31	8000	6400
4270	300	31	5000	6000
5130	100	55	0	1000
5130	100	55	0	1500
5130	100	04	0	1000
5130	100	04	0	9000
5130	100	30	0	3000
5130	100	30	0	2000
5130	300	31	0	2000
5130	300	31	0	5000

```
proc format ;
       value $country      4270 = 'FRANCE'
                           5130 = 'KUWAIT';
       value $product      100  = 'FOOD'
                           200  = 'METAL';
                           300  = 'WOOD';
       value $district     04   = 'BOSTON'
                           30   = 'SEATTLE'
                           31   = 'TAMPA'
                           55   = 'NEWYORK';
       run;

proc print data = One;
       format country $country.
                product $product.
               district $district.;
run;
```

Exhibit Data Set One

Obs	country	product	district	imports	exports
1	FRANCE	FOOD	BOSTON	1300	0
2	FRANCE	FOOD	BOSTON	2300	0
3	FRANCE	FOOD	SEATTLE	550	0
4	FRANCE	FOOD	SEATTLE	5500	2000
5	FRANCE	FOOD	TAMPA	1150	0
6	FRANCE	FOOD	TAMPA	4150	0
7	FRANCE	METAL	BOSTON	10000	4500
8	FRANCE	METAL	BOSTON	11000	1500
9	FRANCE	METAL	TAMPA	8000	6400
10	FRANCE	WOOD	TAMPA	5000	6000
11	KUWAIT	FOOD	NEWYORK	0	1000
12	KUWAIT	FOOD	NEWYORK	0	1500
13	KUWAIT	FOOD	BOSTON	0	1000
14	KUWAIT	FOOD	BOSTON	0	9000
15	KUWAIT	FOOD	SEATTLE	0	3300
16	KUWAIT	FOOD	SEATTLE	0	2000
17	KUWAIT	WOOD	TAMPA	0	2000
18	KUWAIT	WOOD	TAMPA	0	5000

Through PROC FORMAT, we have accomplished more than just making the data more descriptive, we have saved time, effort and resources. SAS now displays meaningful names using numeric-coded values. Otherwise, the SAS programmer would have to embed the descriptions into data sets, requiring more physical space on the hard drive. Since we're drawing our descriptions from one source, we know that the reports can dynamically change as the descriptions change. As any of the descriptions change the programmer updates the PROC FORMAT statement and simply reruns the code.

CAN THERE BE A PROBLEM?

Using a PROC FORMAT statement inside of a SAS program has its advantages, but in many cases, depending on how you use it, the statement could be a 'growing' problem. Updating a few PROC FORMAT statements is simple, but complexity 'expands' as the number of elements grows larger, with hundreds or even thousands of elements.

How does SAS format allow us to automate the creation and running of a PROC FORMAT statement with "always current" information?

MORE EXAMPLES MOVING TO "FORMATTING ON THE FLY"

The place to start is with a data set. Given the formats we used in our previous example, let's first create a data set with country codes and their descriptions.

Data Set Country

CTRYCODE	CTRYDESCRIPTION
4270	FRANCE
5130	KUWAIT
3210	RUSSIA
4010	CHINA
5850	CANADA

SECOND EXAMPLE

Emulate our previous procedure and still have the benefits of an automated system by writing the PROC FORMAT statement to an external file and then including the file in our program.

First, the system must have a permanent place to write the file:

```
filename convimdc 'frmtimdc.sas';
```

Then create 'header' information of the PROC FORMAT statement in this file.

```
data _null_;
      file convimdc;
      set country end=eof;
      if _n_ = 1 then do;
            put "libname library 'c:\temp\frmt';";
            put 'proc format library=library;';
            put 'value $country';
      end;
```

Now insert data into the file, including those country code and descriptions users want displayed in lieu of the actual numeric country codes. Include a RUN statement at the end of the data set.

```
      do;
            put '"' ctrycode +(-1)'"' '=' '"'
                  ctrydescription +(-1)'"';
      end;


      if eof then do;
            put ';';
            put 'run;';
      end;
      run;
```

Finally, after all this information is written to the FRMTIMDC.SAS file, "%include" it in our SAS code.

```
%include 'frmtimdc.sas';
```

SAS has automated the same process we did before. However, it is done in a way that's different. Data is written to a physical file. It's not just a file in the temporary WORK area. It is a permanent file and must be accessible for write and read ability. If this technique is unreasonable or questionable, then perhaps our third example is better.

THIRD EXAMPLE

PROC FORMAT can be fed values, labels and format names from a data set using the CNTLIN option. Our data set must contain at least three fields:

1.) FMTNAME: The name of the format.
2.) START: The value you wish to format.
3.) LABEL: The label you want to associate with the value.

If you're creating character formats, you must do one of two things:

1.) Begin the format name with a "$"
2.) Add a fourth field, TYPE, with a value of "c"

Given these rules, let's take our original COUNTRY data set and transform it into a data set we can use with the CNTLIN option.

```
data two;
      length label $ 11;
      set country(rename=(ctrycode=start ctrydesciption=label));
      retain fmtname 'country' type 'c';
      output;
run;

proc sort data=Two;
      by start;
run;
```

Data Set Two

	Label	start	fmtname	type
1	RUSSIA	3210	country	c
2	CHINA	4010	country	c
3	FRANCE	4270	country	c
4	KUWAIT	5130	country	c
5	CANADA	5850	country	c

Once the data set is formatted correctly, there are only two lines that need to be run.

```
proc format library= work cntlin=two;
run;
```

Now, we've automated the production of the PROC FORMAT statement with current data. Also we've done it with only one assumption: The program is able to read a data set with current information. If you're running a business critical application, it's probably safe to assume that you can get access rights to current reference files.

FOURTH EXAMPLE "FORMATTING ON THE FLY"

What if the SAS needs only one tenth of the formats you've just produced? With just four formats, like Data set Two above, that is not an issue. But what if you have 25,000 possible format items and an application needs a handful of items? Should we go through all the mechanics of producing those formats even when they're not needed?

For our example, our system needs to create formats for COUNTRY, PRODUCT and DISTRICT. In a U.S. Census Bureau real world database, there are about 250 countries, about 50 districts and over 25,000 commodities. Of those 25,300 possible formats, an application might need to use less than 50 formats.

Again, we have three example data sets:

Data Set Country

CTRYCODE	CTRYDESCRIPTION
3210	RUSSIA
4010	CHINA
4270	FRANCE
5130	KUWAIT
5850	CANADA

Data Set Product

PRODUCTCODE	PRODUCTDESCRIPTION
100	FOOD
200	METAL
300	WOOD

Data Set District

DISTRICTCODE	DISTRICTDESCRIPTION
04	BOSTON
30	SEATTLE
31	TAMPA
55	NEWYORK

Using PROC SQL, our program creates one data set but with three formats.

```
proc SQL;

      create table formats as

      /* Create the FMTNAME field with the value $COUNTRY. */

      select distinct '$country' as fmtname label="fmtname",

      /* Use CTRYCODE from database C as the value in   */
      /* the START field.                                */

      c.ctrycode as start label="start",

      /* Use CTRYDESCRIPTION from database C as the value  */
      /* in the LABEL field.                               */

      upcase(c.CTRYDESCRIPTION) as label label = "label",

      /* Use COUNTRY from database B as the value in       */
      /* the START field.                                  */

      b.country as start label="start"

      /* Data set B is WORK.ONE and data set C is            */
      /* WORK.COUNTRY.                                     */

      from work.one b , work.country c

      /* Combine the two data sets where CTRYCODE equals */
      /* COUNTRY.                                          */

      where c.ctrycode = b.country

      UNION

      select distinct '$product' as fmtname label="fmtname",
      e.productcode as start label="start", upcase(e.productDESCRIPTION)
      as label label = "label", d.product as start label="start"
      from work.one d , work.product e
      where e.productcode = d.product

      UNION

      select distinct '$district' as fmtname label="fmtname",
      f.districtcode as start label="start",
      upcase(f.districtDESCRIPTION) as label label = "label", g.district
      as start label="start"
      from work.one g , district f
      where f.districtcode = g.district  ;

      quit;
run;
```

The resultant data set *FORMATS*, contains required formats based totally on **actual and variable data being analyzed**.

Data Set FORMATS

fmtname	start	label
$country	4270	FRANCE
$country	5130	KUWAIT
$district	04	BOSTON
$district	30	SEATTLE
$district	31	TAMPA
$district	55	NEWYORK
$product	100	FOOD
$product	200	METAL
$product	300	WOOD

Now, run the PROC FORMAT and PROC PRINT.

```
proc format library= work cntlin=formats;
run;

proc print data=Two;
       format country $country. product $product.
              district $district.;
run;
```

The resultant *PROC PRINT* looks very much like *Exhibit Data Set One*.

Exhibit Data Set Two
PROC PRINT

Obs	country	product	district	imports	exports
1	FRANCE	FOOD	BOSTON	1300	0
2	FRANCE	FOOD	BOSTON	2300	0
3	FRANCE	FOOD	SEATTLE	550	0
4	FRANCE	FOOD	SEATTLE	5500	2000
5	FRANCE	FOOD	TAMPA	1150	0
6	FRANCE	FOOD	TAMPA	4150	0
7	FRANCE	METAL	BOSTON	10000	4500
8	FRANCE	METAL	BOSTON	11000	1500
9	FRANCE	METAL	TAMPA	8000	6400
10	FRANCE	WOOD	TAMPA	5000	6000
11	KUWAIT	FOOD	NEWYORK	0	1000
12	KUWAIT	FOOD	NEWYORK	0	1500
13	KUWAIT	FOOD	BOSTON	0	1000
14	KUWAIT	FOOD	BOSTON	0	9000
15	KUWAIT	FOOD	SEATTLE	0	3300
16	KUWAIT	FOOD	SEATTLE	0	2000
17	KUWAIT	WOOD	TAMPA	0	2000
18	KUWAIT	WOOD	TAMPA	0	500

The underlying difference between *Exhibit Data Set One* and *Exhibit Data Set Two* is that *Exhibit Two* utilized much less 'hard-coded' format code, and utilized formats from potentially vast data sets 'on the fly', thereby using fewer resources.

CONCLUSION

Through 'evolutionary' use of data sets and SAS formats and data manipulation, SAS programs can save time and resources by creating formats from data sets "on the fly". This also ensures that formats are accurate and don't use unnecessary permanent storage.

CONTACT INFORMATION

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

Mikhail Gruzdev
U.S. Census Bureau
Foreign Trade Division
Rm 6K502
Washington, D.C. 20233
Phone: 301-763-2206
E-mail: mikhail.g.gruzdev@census.gov

Allen Blackburn
U.S. Census Bureau
Foreign Trade Division
Rm 6K106
Washington, D.C. 20233
Phone: 301-763-6921
E-mail: allen.j.blackburn@census.gov