Paper 045-29

A Format to Make the _TYPE_ Field of PROC MEANS Easier to Interpret

Matt Pettis, Thomson West, Eagan, MN

ABSTRACT: PROC MEANS analyzes datasets according to the variables listed in its Class statement. Its computed _TYPE_ variable can be used to determine which class variables were used for the analysis variable calculation. It can be very difficult to determine by inspection of the _TYPE_ variable which class variables were used in a calculation of any given row. The %TypeFormat macro takes the CLASS variable list and creates a format that associates the values of the _TYPE_ variable with a string listing the variables used in the calculation separated by the '*' character.

INDTRODUCTION:

The purpose of this paper is to present a macro that creates a format which formats the _TYPE_ variable in a PROC MEANS output of SAS® from a number into a descriptive string.

PROBLEMS READING THE _TYPE_ VARIABLE

Consider the following scenario: Every day, a PROC MEANS with five class variables is run on a dataset, and the result is appended to a base of daily PROC MEANS outputs. An analyst is asked to run a report on the base for a certain three class variable combination. How does the analyst retrieve the correct data for the report?

PROC MEANS provides a solution to this problem via the _TYPE_ variable. A group of rows with identical _TYPE_ values indicates that the same variables were used in calculating the analysis variables, and each row with this _TYPE_ value represents a different combination of the level of the variables. Rows with different _TYPE_ variables indicate that different combinations of variables were used.

While this is a mathematically efficient way to categorize the analysis, it is a difficult value to use in practice. In order to interpret the value of the _TYPE_ variable, the analyst must be familiar with binary mathematics. Directions on how to interpret the meaning of the _TYPE_ variable may be found in the SAS9 Language Reference. The analyst must be able to convert the _TYPE_ value into binary, then use that result to pick out the relevant analysis variables from the class statement. A solution to this problem is to use the %TypeFormat macro (code included), which associates the _TYPE_ variable value to a string that lists the classification variables used in a particular calculation.

For example, consider the following output of a simple fictitious dataset that has the location, home directory, and HTTP response code of each request to a server:

Figure 1: PROC MEANS and output – No _TYPE_ formatting

Code:

```
PROC MEANS data=work missing noprint;
   class Location Directory ResponseCode;
   output out=temp;
   run;
PROC PRINT data=temp;
   Title '_TYPE_ WITHOUT formatting'; run;
```

Output:

TYPE WITHOUT formatting

Obs	_TYPE_	Location	Directory	Response Code	_FREQ_
1	0				30
2	1				1
3	1			200	11
4	1			404	18
5	2		/Downloads		12
6	2		/Graphics		18
7	3		/Downloads		1
8	3		/Downloads	200	6
9	3		/Downloads	404	5
10	3		/Graphics	200	5
11	3		/Graphics	404	13
12	4	SiteA			13
13	4	SiteB			17
14	5	SiteA			1
15	5	SiteA		200	4
16	5	SiteA		404	8
17	5	SiteB		200	7
18	5	SiteB		404	10
19	6	SiteA	/Downloads		5
20	6	SiteA	/Graphics		8
21	6	SiteB	/Downloads		7
22	6	SiteB	/Graphics		10
23	7	SiteA	/Downloads		1
24	7	SiteA	/Downloads	200	2
25	7	SiteA	/Downloads	404	2
26	7	SiteA	/Graphics	200	2
27	7	SiteA	/Graphics	404	6
28	7	SiteB	/Downloads	200	4
29	7	SiteB	/Downloads	404	3
30	7	SiteB	/Graphics	200	3
31	7	SiteB	/Graphics	404	7

In this output, the _TYPE_=0 value identifies the summary with no classification variables, the _TYPE_=1 values identifies the summary by the ResponseCode variable, _TYPE_=2 identifies the summary by the Directory variable, _TYPE_=3 identifies the summary by Directory and ResponseCode variables, and so on through _TYPE_=7. Notice how difficult it can be to determine which CLASS variables were used for analysis in each row, especially if there are missing values, as is the case with one of the Response codes.

The %TypeFormat macro creates a format that associates values of the _TYPE_ variable with a string of the variable names used in the analysis separated by a '*' character (see Appendix A for the macro code). In order to do this, the %TypeFormat must be called with two arguments: one being the name of the format that will be created (formatname=), and the other being the list of variables in the Class statement of the PROC MEANS *in exactly the same order as it appears in the PROC MEANS statement* (var=):

Figure 2: PROC MEANS and output – _TYPE_ formatting.

Code:

```
%TypeFormat(formatname=testtyp,var=Location Directory ResponseCode);
PROC PRINT data=temp;
   var _type_ Location Directory ResponseCode _freq_;
   format _type_ testtyp.;
   title '_TYPE_ WITH formatting'; run;
```

Output:

TYPE WITH formatting

Obs	_TYPE_	Location	Directory	Response Code	_FREQ_
1					30
2	ResponseCode				1
3	ResponseCode			200	11
4	ResponseCode			404	18
5	Directory		/Downloads		12
6	Directory		/Graphics		18
7	Directory*ResponseCode		/Downloads		1
8	Directory*ResponseCode		/Downloads	200	6
9	Directory*ResponseCode		/Downloads	404	5
10	Directory*ResponseCode		/Graphics	200	5
11	Directory*ResponseCode		/Graphics	404	13
12	Location	SiteA			13
13	Location	SiteB			17
14	Location*ResponseCode	SiteA			1
15	Location*ResponseCode	SiteA		200	4
16	Location*ResponseCode	SiteA		404	8
17	Location*ResponseCode	SiteB		200	7
18	Location*ResponseCode	SiteB		404	10
19	Location*Directory	SiteA	/Downloads		5
20	Location*Directory	SiteA	/Graphics		8
21	Location*Directory	SiteB	/Downloads		7
22	Location*Directory	SiteB	/Graphics		10
23	Location*Directory*ResponseCode	SiteA	/Downloads		1
24	Location*Directory*ResponseCode	SiteA	/Downloads	200	2
25	Location*Directory*ResponseCode	SiteA	/Downloads	404	2
26	Location*Directory*ResponseCode	SiteA	/Graphics	200	2
27	Location*Directory*ResponseCode	SiteA	/Graphics	404	6
28	Location*Directory*ResponseCode	SiteB	/Downloads	200	4
29	Location*Directory*ResponseCode	SiteB	/Downloads	404	3
30	Location*Directory*ResponseCode	SiteB	/Graphics	200	3
31	Location*Directory*ResponseCode	SiteB	/Graphics	404	7

CONCLUSION

Notice how much easier the Figure 2 format is to read than Figure 1. The variables that are involved in the calculation are now clearly identified and the levels of these variables can now be read easily from the appropriate columns. This also makes identifying missing data easier, as the CLASS variable used is listed even if its data value is missing.

REFERENCES

SAS Institue Inc. (2002), *SAS9 Language Reference: Dictionary, Volumes 1 and 2*. Cary, NC: SAS Institute Inc.

SAS Institue Inc. (2002), *SAS9 Macro Language: Reference*. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

Matt Pettis
Thomson West Group
610 Opperman Dr.
Eagan, MN 55417
+1 (651) 848-3976
Matt.pettis@thomson.com

Appendix A: %TypeFormat SAS code:

```
%macro TypeFormat(formatname=typefmt,var=x1 x2 x3 x4);
   /* Count the number of variables, put into var_count */
   %local var_count;  %let var_count = 1;
   %do %until (%scan(&var,&var_count) eq); %let var_count = %eval(&var_count+1); %end;
   %let var_count = %eval(&var_count-1);

   /* Assign each variable name to an indexed macro &&var_val&i */
   %local i; %let i = %eval(&var_count);
   %do %until (&i <= 0);
      %local var_val&i;
      %let var_val&i = %scan(&var,&i); %let i = %eval(&i-1);
      %end;

      /* Create temp dataset to use as format */
   data _tmp;
      keep label start fmtname type;
      retain fmtname "&formatname" type 'n';
      length label $ 256 sep $ 1;
      sep = '*'; * Separator character;

      /* Loop through the type combinations */
      type_iter = 2**(&var_count) - 1; * Loop through the types;
      var_iter = &var_count; * Loop over the binary digits;
      do start = 0 to type_iter; * Type iteration loop;
         i_tmp = start; label = '';
         do j = var_iter to 1 by (-1); * Binary digit loop;
            bin_digit = int(i_tmp/(2**(j-1)));
            if bin_digit = 1 then do;
               * Get appropriate variable name;
             x = symget('var_val'||left(trim(put(&var_count - j + 1,3.))));
             * Add the separator to the string;
             x = left(trim(x))||sep;
             * Append selected variable name to label string;
             newlabel = trim(label)||x;
             * Reassign label as newlabel;
             label = newlabel;
             * Decrement i_tmp if bin_digit is in types binary representation;
             i_tmp = i_tmp - 2**(j-1);
             end;
            end;
            label = left(label); * justify label test to left;
            len = length(label);
            * Take off separator that was appended to end;
            label = substr(label,1,len-1);
            output;
         end;
      stop;
      run;

* create the format from _tmp dataset;
proc format cntlin=_tmp; run;
* Delete _tmp dataset to clean up work library;
proc datasets; delete _tmp; quit; run;
%mend TypeFormat;
```