Paper 83-25

What We Really Need is a %BY Statement – V2

Ray Pass
Ray Pass Consulting

Here's the scenario. You have to produce a series of different reports from the same overall population, and each report has to be run separately for each level of a specified bygroup (or combination of bygroups) in the population. The catch is that the client (boss, whatever) wants the report output to appear in bygroup order; i.e. all reports for bygroup-1, then all reports for bygroup-2, etc. Another catch is that you don't necessarily know a priori what the different levels of the bygroup are, or even how many there are; in fact, this is to be part of a routine production system where these bygroup levels will most assuredly be changing from run to run. There are undoubtedly many ways to handle a situation like this. The first version of this little paper showed one method. Since writing that paper, and using that method many times, I have found a "better" way to do it. This method was demonstrated on SAS-L a while back.

Assume the data exist in a SAS data set called **alldata** with a classifying variable called **byvar**. For this example, there happen to be three levels of **byvar** ('by1', 'by2', 'by3'), although the method works with any number of levels and you don't have to specify them beforehand. Actually, the method will only work with as many instances of **byvar** values as can be contained in a macro variable (plus one extra separator character per value), so it is SAS version dependent. This limitation should not really be a problem and can be dealt with by some other clever SAS programmer. You've got three reports to run and you need them to be produced in the following order:

Rep-1 for Bygroup-1
Rep-2 for Bygroup-1
Rep-3 for Bygroup-1

Rep-1 for Bygroup-2
Rep-2 for Bygroup-2
Rep-3 for Bygroup-2

Rep-1 for Bygroup-3
Rep-2 for Bygroup-3
Rep-3 for Bygroup-3

First you use PROC SQL to create two "counter" macro variables, **&mcount** and **&mbyvals**, which respectively contain the number of distinct values of **byvar**, and a list of the actual values of **byvar** separated by a separator character, "#" (any character can be used as the separator.) The code looks like:

```
*----------------------------------;
%global mcount mbyvals;
proc sql noprint;
   select count(*),
          byvar
   into   :mcount,
          :mbyvals separated by '#'
   from   (select distinct byvar
           from   alldata);
quit;
*----------------------------------;
```

The results of the above code execution are two macro variables with the following values:

&mcount - 3
&mbyvals – by1#by2#by3

The next part of the process compiles and runs a macro called **%runall** which calls the needed report code once for each level of **byvar** as follows:

```
%macro runall;
  %do m=1 %to &mcount;
     proc whatever1 data=alldata

(where=(byvar="%scan(&mbyvals,&m,#)"));
     /** REMAINING CODE FOR FIRST PROC **/
     run;

     proc whatever2 data=alldata

(where=(byvar="%scan(&mbyvals,&m,#)"));
     /** REMAINING CODE FOR SECOND PROC **/
     run;

     proc whatever3 data=alldata

(where=(byvar="%scan(&mbyvals,&m,#)"));
     /** REMAINING CODE FOR LAST PROC **/
         run;
  %end;
%mend;

%runall
```

The dataset **alldata** contains all the data to be reported on. The procedures used in the guts of the technique can be any SAS procs. The method works by setting up a macro do-loop to be executed once for each value of **byvar** as listed in the macro variable

&mbyvals. This do loop is run **&mcount** times, once for each value listed in **&mbyvals**. The code for all the report procedures to be run is contained in the do loop. Each time the code in the loop is run, a different subset of **alldata** is used as selected by the **where=(byvar=...)** data set option. The value of **byvar** is changed in each iteration of the loop by using the **%scan** macro function to incrementally set it equal to the next #-delimited "word" contained in **&mbyvals**. The macro generated code for our example would be:

```
proc whatever1 data=alldata
               (where=(byvar="by1"));
   /** REMAINING CODE FOR FIRST PROC **/
run;

proc whatever2 data=alldata
               (where=(byvar="by1"));
   /** REMAINING CODE FOR SECOND PROC **/
run;

proc whatever3 data=alldata
               (where=(byvar="by1"));
   /** REMAINING CODE FOR LAST PROC **/
run;

proc whatever1 data=alldata
               (where=(byvar="by2"));
   /** REMAINING CODE FOR FIRST PROC **/
run;

proc whatever2 data=alldata
               (where=(byvar="by2"));
   /** REMAINING CODE FOR SECOND PROC **/
run;

proc whatever3 data=alldata
               (where=(byvar="by2"));
   /** REMAINING CODE FOR LAST PROC **/
run;
```

```
proc whatever1 data=alldata
               (where=(byvar="by3"));
   /** REMAINING CODE FOR FIRST PROC **/
run;

proc whatever2 data=alldata
               (where=(byvar="by3"));
   /** REMAINING CODE FOR SECOND PROC **/
run;

proc whatever3 data=alldata
               (where=(byvar="by3"));
   /** REMAINING CODE FOR LAST PROC **/
run;
```

That's it. A simple method for determining the order of output which is totally data-driven. The method can easily be expanded to multiple classifying variables as well by creating multiple sets of "counter" macro variables and then nesting do loops with complex **where=** data set options. This is really a lot less imposing than it sounds. Try it.

The author of this paper can be contacted as follows:

Ray Pass
Ray Pass Consulting
5 Sinclair Place
Hartsdale, NY 10530

Voice: (914) 693-5553
Fax: on request
email: raypass@att.net