

GENERALIZING PROC REPORT WITH SAS®MACRO

Philip M. Pochon
Computer Task Group, Inc. Indianapolis, IN.

Abstract

The REPORT procedure in version 6 SAS® is a powerful tool for customizing reports based on tabular data structures. Data columns may be formatted, labeled and grouped into clusters with a few procedural commands such as COLUMN and DEFINE and options such as PANEL.

Time series data that must be clustered into intervals is a data structure well suited for reporting with the REPORT procedure. However, when the number of intervals may vary, and the number of points within each interval may vary the REPORT procedure must be further generalized.

This paper shows how the SAS® Macro facility can be used to control the creation of REPORT procedure code so that one macro function can produce clustered time point reports with a flexible definition of time point intervals.

1. PROC REPORT Overview

The REPORT procedure combines features from the base SAS® PRINT, MEANS and TABULATE procedures to produce a report writer that provides powerful tools for column and rows management with summarization capabilities. This paper focuses on the first of these tools: column management.

1.1 The COLUMN Statement

The COLUMN statement in PROC REPORT is the most direct way to define the arrangement of columns across a report page. The general form of the COLUMN statement is:

COLUMN *column-list*;

where each item in *column-list* is the name of a data set variable, a computed variable or a statistic that can be computed by PROC REPORT.

Columns may be separated into groups and assigned a header by enclosing a text string and a sub-list in parentheses. The statement:

```
COLUMN  ('Projected' var_1 var_2 var_3)
        ('Actual' var_4 var_5 var_6);
```

will print two groups of three columns each on a page, with the header line 'Projected' centered above the first trio of columns and the header line 'Actual' centered above the second trio of columns.

Items may be stacked one above the other by separating the items by a comma. The statements:

```
COLUMN animal weight, (mean stderr);
DEFINE animal / group;
```

compute and print as separate columns the mean and standard error of multiple measurements of an animal's weight.

1.2 The DEFINE Statement

The DEFINE statement in PROC REPORT defines the characteristics of an item. These characteristics include how PROC REPORT uses the item in a report, how it formats the item and what column header it prints above the column's values. The general form of the DEFINE statement is:

```
DEFINE item / <usage>
              <attribute-list>
              <option-list>
              <justification>
              <'column-header'>;
```

Attributes that can be specified include the column's format, its width and the spacing between it and the prior (next left) column.

The following statement defines a variable that is to be printed (DISPLAY) with a date format of DDDMMYYYY (FORMAT) in a column that is 10 character wide (WIDTH=10) and is separated from its left neighbor by four blank characters (SPACING=4):

```
DEFINE STRTDATE / DISPLAY
                  FORMAT=DATE9.
                  WIDTH=10
                  SPACING=4;
```

There are a number of other PROC REPORT features that relate to column management. The PANEL option is the most directly relevant.

* SAS is the registered trademark of the SAS Institute.

However, this paper will limit its discussion to the COLUMN and DEFINE statements.

2. SAS® Macro Overview

The macro language is a facility within base SAS® that allows the programmer to perform certain tasks outside the base SAS® interpreter. Three features of the macro language are crucial to this paper:

- macro variable word list processing
- %DO macro control statements
- macro functions

2.1 Macro Variable Word Lists

Macro variables are by definition character strings. Such strings may be composed of words. The macro language contains a pair of SAS® supplied basic functions (%SCAN and %QSCAN) to select words from a list.

When coupled with %DO loop processing macro variable word lists can walk through a list and use each word in the list in a specified operation or statement within the loop.

2.2 %DO Macro Control Statements

Macro statements or base SAS® statements that are to be performed conditionally or repetitively may be placed within a macro %DO block.

%DO execution may be conditional:

```
%IF (expression) %THEN %DO;
  .Base SAS® or macro statements;
%END;
```

%DO execution may also be repetitive:

```
%DO i=1 %TO 10;
  .Base SAS® or macro statements;
```

```
%END;
```

%DO %WHILE and %DO %UNTIL processing provide a mixture of conditional and repetitive processing.

%DO macro statements must be located within a macro function. They can not occur in open code.

2.3 Macro Functions

Macro function allow the programmer to isolate macro and/or base SAS® code within a function that may be called from another macro function, or from open code in a base SAS® program.

Macro functions allow values to be passed in as macro variable parameters. Parameters can not be used to pass result values back to the calling function.

3. The Base PROC REPORT Program

Let us start with a simple PROC REPORT program show in figure 1 which produces the report show in figure 2.

Figure 1. Base PROC REPORT Code

```
proc report data=work.weights headskip center
  nowindows colwidth=6;
  column animal
    ('Week 1' day_1 day_3 day_6)
    ('Week 2' day_8 day_10 day_13);
  define animal / display 'Animal';
  define day_1 / display 'Day 1' spacing=8;
  define day_3 / display 'Day 3';
  define day_6 / display 'Day 6';
  define day_8 / display 'Day 1' spacing=8;
  define day_10 / display 'Day 3';
  define day_13 / display 'Day 6';
run;
```

Figure 2. Body Weight (g) Report with a Fixed Number of Days and Weeks

Animal	Week 1			Week 2		
	Day 1	Day 3	Day 6	Day 1	Day 3	Day 6
0001	238.2	239.4	240.5	242.1	243.7	246.0
0002	231.9	233.0	234.3	235.8	237.2	238.8
0003	235.6	235.2	236.1	235.8	236.3	236.5

The code in figure 1 works well as long as results data are always collected for two weeks. If data values are collected for three weeks another PROC REPORT program must be written, or a macro function may be used.

The first step in generalizing a PROC REPORT macro function is to let the number of weeks vary. The code in Figure 3 is designed to process any data set with two or more weeks of results data.

Figure 3. Macro Function with General N of Weeks

```
%macro report1(_indsn=work.weights,
               _n=2);
%local week days;

proc report data=&_indsn headskip center
            nowindows colwidth=6;
  column animal
    %do week=1 %to &_n;
      %let days=day_%eval(1+7*%eval(&week-1));
      %let days=&days day_%eval(3+7*%eval(&week-1));
      %let days=&days day_%eval(6+7*%eval(&week-1));
      ("Week &week" &days)
    %end;
  ;
  define animal / display 'Animal';
  %do week=1 %to &_n;
    define day_%eval(1+7*%eval(&week-1)) /
      display 'Day 1' spacing=8;
    define day_%eval(3+7*%eval(&week-1)) /
      display 'Day 3';
    define day_%eval(6+7*%eval(&week-1)) /
      display 'Day 6';
  %end;
run;

%mend report1;
```

The data set name is now a parameter, so any data set may be sent into the function. The number of weeks is also a parameter. The COLUMN statement contains a macro %DO loop that processes weeks one at a time. Within the loop three %LET statements build the list of days to be included within the week. The semicolon that follows the %END statement closes the COLUMN statement.

The second %DO loop builds and issues three DEFINE statements for each day within each week.

Macro REPORT1 works well if we always start collecting data on week one and collect data every week thereafter. If either of these assumptions is not true the function will not work. In order to fully generalize the processing of weeks we should use list processing. In this situation the list will contain two or more integer values which indicate the week(s) to be processed.

Because the number of weeks is variable we first need a function to count the number of weeks in the list. Figure 4 shows such a function.

Figure 4. Macro Function to Count Words in a List

```
%macro cntwords(_string=1 3 5 7 9);
%local count word;

%let count=1;
%let word=%qscan(&_string,&count,%str( ));
%do %while (&word ne);
  %let count=%eval(&count+1);
  %let word=%qscan(&_string,&count,%str( ));
%end;
%eval(&count-1)

%mend cntwords;
```

The next to last line (%eval(&count-1)) is peculiar for SAS in that it does not end in a semicolon. The value generated is inserted into the calling function at the spot in which the call was made. We can thus use the CNTWORDS function as part of a %DO statement:

```
%DO i=1 %TO %cntwords(_string=1 3 5 7 9);
```

Our PROC REPORT function can now be generalized to process a list of weeks, as show in figure 5.

Figure 5. Macro Function with Week List

```
%macro report2(_indsn=work.weights,
               _wklist=1 3 5 7 9);
%local i week days;

proc report data=&_indsn headskip center
            nowindows colwidth=6;
  column animal
    %do i=1 %to %cntwords(_string=&_wklist);
      %let week=%scan(&_wklist,&i);
      %let days=day_%eval(1+7*%eval(&week-1));
      %let days=&days day_%eval(3+7*%eval(&week-1));
      %let days=&days day_%eval(6+7*%eval(&week-1));
      ("Week &week" &days)
    %end;
  ;
  define animal / display 'Animal';
  %do i=1 %to %cntwords(_string=&_wklist);
    %let week=%scan(&_wklist,&i);
    define day_%eval(1+7*%eval(&week-1)) /
      display 'Day 1' spacing=8;
    define day_%eval(3+7*%eval(&week-1)) /
      display 'Day 3';
    define day_%eval(6+7*%eval(&week-1)) /
      display 'Day 6';
  %end;
run;

%mend report2;
```

The macro function REPORT2 is fully generalized for weeks, but its processing of days is still very rigid. Each week must have data collected on day 1, day 3 and day 6. We could establish another parameter to pass a list of days, but this still assumes that data is collected on the same day within each week.

A more flexible approach is to pass only a list of days, and to let the function determine how many weeks are present, and how many days fall within each week. Figure 6 shows a macro function designed to process a day list.

Figure 6. Macro Function with Day List

```
%macro report3(_daylist=1 3 6 15 17 20);
%local i oldweek newweek day dayn collist;
%let oldweek=%eval(%scan(&_daylist,1)/7);
%let collist=;
%let dayn=%cntwords(_string=&_daylist);

proc report data=work.weights headskip center
nowindows colwidth=6;
column animal
%do i=1 %to &dayn;
%let day=%scan(&_daylist,&i);
%let newweek=%eval(&day/7);
%if (&newweek>&oldweek) %then %do;
("Week %eval(&oldweek+1)" &collist)
%let collist=day_&day;
%let oldweek=&newweek;
%end;
%else %do;
%let collist=&collist day_&day;
%if (&i=&dayn) %then %do;
("Week %eval(&oldweek+1)" &collist)
%end;
%end;
%end;
;
define animal / display 'Animal';
%let oldweek=%eval(%scan(&_daylist,1)/7);
%do i=1 %to &dayn;
%let day=%scan(&_daylist,&i);
%let newweek=%eval(&day/7);
%let weekord=%eval(&day-%eval(7*&newweek));
%if (&newweek>&oldweek) or (&i=1) %then %do;
%let oldweek=&newweek;
define day_&day /
display "Day &weekord" spacing=8;
%end;
%else %do;
define day_&day / display "Day &weekord";
%end;
%end;
run;

%mend report3;
```

The processing in macro function REPORT3 maintains the macro %DO loop to create the COLUMN statement and the macro %DO loop to create the DEFINE statements. The processing within each loop is more complex.

The function starts by determining the week of the first day, and determining the number of days in the parameter list. The CNTWORDS function could be used wherever the macro variable DAYN is found, but given the number of times that DAYN is used it is more efficient to compute it once at the start of the function.

Within the PROC REPORT code the first step is to determine the week by dividing the day by seven. Macro division with the %EVAL function is integer division, so the week value is only the integer (truncated, not rounded) portion of the result.

Once the week is known the processing to follow can be determined. In the COLUMN statement section if a week boundary has been crossed the code fragment defining the header and the days to include is set, and a new day list is started. In the DEFINE statement section if a week boundary is being crossed the SPACING option is added to the define statement.

If the week list '1 3' is submitted to function REPORT2 and the day list '1 3 6 15 17 20' is submitted to function REPORT3 the PROC REPORT code lines produced from the two are identical.

The function REPORT3 still contains two limiting assumptions about time: that data values are collected on a day basis and that days are grouped into weeks. If we wish to fully generalize the function we must pass as parameters two lists: one list with the time points, and a second list with the break point values for each grouping within the time points. We must also pass as parameters variable name and labeling information. Figure 7 shows such a fully generalized macro function.

Figure 7. Fully Generalized Macro Function

```
%macro report4(_pntlist=1 3 6 15 17 20,
               _prefix=day,
               _pntlabl=Day,
               _brklist=0 5 10 15 20,
               _intlabl=Period);
%local i j point pointn intvrln collist;
%let pointn=%cntwords(_string=&_pntlist);
%let intvrln=%cntwords(_string=&_brklist);

proc report data=work.weights
    headskip center nowindows colwidth=6;
column animal
%do i=1 %to %eval(&intvrln-1);
    %let collist=;
    %do j=1 %to &pointn;
        %let point=%scan(&_pntlist,&j);
        %if (&point>%scan(&_brklist,&i)) and
            ((&point<%scan(&_brklist,%eval(&i+1))) or
             (&point=%scan(&_brklist,%eval(&i+1))))
            %then %let collist=&collist &_prefix.&point;
        %end;
        %if (&collist^= ) %then %do;
            ("&_intlabl &i" &collist)
        %end;
    %end;
;
%let i=2;
%do j=1 %to &pointn;
    %let point=%scan(&_pntlist,&j);
    %if (&j=1) %then %do;
        define &_prefix.&point /
            display "&_pntlabl &point" spacing=8;
        %end;
    %else %do;
        %if (&point>%scan(&_brklist,&i)) %then %do;
            %let i=%eval(&i+1);
            define &_prefix.&point /
                display "&_pntlabl &point" spacing=8;
            %end;
        %else %do;
            define &_prefix.&point /
                display "&_pntlabl &point";
            %end;
        %end;
    %end;
run;

%mend report4;
```

The macro function REPORT4 can group seconds into minutes, hours into days, days into weeks or any time point type into arbitrary groups. The groups do not have to be equal, for the span between break points can vary and the number of points in each group can vary.

The COLUMN section starts with a loop (i) to generate one COLUMN statement fragment for each interval. Within each interval loop a second loop (j) checks each day to determine if it falls within the

interval. A time point falls within the interval if it is greater than the low break point and equal to or less than the high break point.

Note that after all days have been checked within an interval the setting of the COLUMN statement fragment to the base SAS® interpreter is conditional (%IF (&COLLIST^=)). If no time points fall within the interval the interval is omitted from the report.

The DEFINE section contains only one loop (j) which produces one DEFINE statement for each time point. The first time point in each interval has the SPACING option.

The _PREFIX parameter contains the column name prefix for time points (e.g. DAY for columns DAY_1, DAY_3, ..., DAY_n). The interval header and time point column label text is also passed as parameters (_INTLABL and _PNTLABL). The following call for REPORT4 will produce our base report:

Figure 8. Call for Base Report

```
%report4(_pntlist=1 3 6 8 10 13,
         _prefix=day,
         _pntlabl=Day,
         _brklist=0 7 14,
         _intlabl=Week)
```

The function REPORT4 does still have certain limitations. One is that time values must be integers. This limit is imposed by the %EVAL macro function which can perform only integer arithmetic. A second implicit assumption is that intervals are ordered by values running from 1 to n. A third parameter list of interval header values could be passed to remove this limitation. The fixed use of ANIMAL as an identifier column could also be removed by creating a column name list parameter to pass this name, or any other set of column names.

The function REPORT4 does represent a major improvement over the rigid PROC REPORT code of Figure 1. PROC REPORT is in its own right a powerful reporting tool. When coupled with SAS® macro its power can be applied without constant tinkering with, and rewriting of code.