Paper RI-13

Diverse Report Generation With PROC REPORT

Chris Speck, PAREXEL International, Durham, NC, USA

ABSTRACT

Automation is often the goal of SAS programming. If we could just hit "Submit" and watch our program generate all our tables and listings while making the right decisions at run time we could get a lot more accomplished. Of course, with the SAS macro facility, we can already do this…up to a point. We can equip our macros with macro logic, we can feed them different parameters, and then watch as they produce one output after another. This works great when all your outputs are based on the same data set and require the same number of columns.

But what if they don't? What if you need to automate report generation from a large number of different data sets? What if you must allow for any number of columns? Developing such a program using macro logic would be cumbersome indeed. You would need %IF %THEN blocks for every contingency, and your program would get so bogged down in logic that you'd be better off without automation at all.

This paper will demonstrate how SAS programmers can easily and gracefully automate Diverse Report Generation. The methods discussed in this paper use a patient profile program as a primary example and will make use of the REPORT procedure, the SQL procedure, and SASHELP views.

INTRODUCTION

Imagine a macro call that looks like this:

```
%MakeReport(ds=ADMH, vars=MHCAT MHTERM MHPTT MHSTDAT MHONGO,
       ttl=Medical History, headr=Safety Population, where=SAFETY='Y');
```

It wouldn't take a pharmaceutical SAS programmer very long to see what this macro aims to accomplish. Simply put, these are parameters for a fairly simple medical history listing. We have five variables coming from ADMH that fit nicely in a PROC REPORT COLUMN statement. We subset ADMH on safety and provide titles and headers. It seems like a neat package for a series of five-column listings.

But what if the next call to this macro looked like this?

```
%MakeReport(ds=ADAE, vars=AESOC AEPT AESTDT AEENDT AESER AESEV AEREL AEOUT AEACN,
       ttl=Adverse Events, headr=Safety Population, where=SAFETY='Y');
```

To handle this with macro logic alone, %MakeReport, would have to include an %IF %THEN statement which specifies which code to use at any given time based on the value of &DS. How else to accommodate the different number of DEFINE statements that must appear in the PROC REPORT? This may be fine to use once or twice, but after ten times, your program will become bulky, prone to bugs, tedious to decode, and impossible to transport across studies. This is a great example of how over-automating can make a program worse rather than better.

Diverse Report Generation, on the other hand, allows the %MakeReport macro to write PROC REPORT code dynamically, without resorting to excessive %IF %THENs. Essentially, we do five things:

1. Subset data according to the WHERE criteria.
2. Add a blank line to the data set if no observations meet the WHERE criteria.
3. Capture the incoming data set's metadata (NAME and LABEL) using the SASHELP view VCOLUMN.
4. Feed this metadata into a PROC SQL step which generates PROC REPORT DEFINE statements within a macro variable.
5. Write our PROC REPORT using this macro variable instead of DEFINE statements.

With this approach, the first PROC REPORT (for medical history) will have only five DEFINE statements, whereas the second (adverse events) will have nine. In fact, there can be as many DEFINE statements in our PROC REPORT as there are values in the VARS parameter. And no hard coding or tedious macro logic is necessary. Moreover, adding

new reports with this approach becomes as easy as entering new parameters. You add the macro call, SAS does the rest.

This paper will go through the five steps outlined above as part of a patient profiles example program. It will also suggest ways to do more with Diverse Report Generation.

STEP 1: SUBSETTING DATA

The skeleton of the %MakeReport macro should begin like this:

```
%macro MakeReport (ds=, vars=, ttl=, headr=, where=1);
  %global define ls empty;
  %let ls=125;
  %let empty=N;

  data &ds.rpt0 (keep=&vars);
    retain &vars;
    set &ds (where=(&where));
  run;
```

We create a data set in WORK with the suffix RPT0 (the period after the macro variable &DS tells SAS where the macro variable ends and the rest of the data set name begins). This is based on &DS and contains only the variables in our VARS parameter. The RETAIN statement orders these variables as they appear in the VARS parameter regardless of whether they are character or numeric. The statement must appear before the SET statement, so SAS can establish variable order before reading incoming data into the program data vector.

Please note that 125 is an arbitrary value for the LS macro variable which will be used later on in the program.

STEP 2: ADDING BLANK LINES TO THE REPORT DATASET

```
  data &ds.rpt0;
    if nobs>0 then stop;
    call symput("empty","Y");
    output;
    stop;
    modify &ds.rpt0 nobs=nobs;
  run;
```

This second DATA step uses the temporary variable NOBS to determine if any observations met the &WHERE criteria. If so, then processing stops and &DS.RPT0 is not changed. If no observations meet the criteria, however, processing continues until &EMPTY (which by default equals N) is set to Y and a blank observation is inserted into the data set. The MODIFY statement then updates the data set. Thus, &DS.RPT0 will always have at least one observation read into the final PROC REPORT.

STEP 3: CAPTURING METADATA

```
data &ds.rpt1 (keep=name label);
  set sashelp.vcolumn (where=(memname="%upcase(&ds.rpt0)"));
run;
```

We create &DS.RPT1 based on the metadata of RPT0. This is done by subsetting the SASHELP.VCOLUMN view, which lists variable metadata. In this case, we use VCOLUMN.MEMNAME which contains all data set names available in your SAS session. Note how we set the data set name to uppercase before we subset since this is how all text values appear in dictionary tables.

The point of this step is to ensure &DS.RPT1 looks something like this:

NAME	LABEL
MHCAT	Category for Medical History
MHTERM	Reported Term for Medical History
MHPTT	Preferred Term for Medical History
MHSTDAT	Start Date
MHONGO	Ongoing?

Table 1. The &DS.RPT1 Data Set at Step 3

Note there are the same number of observations as there are values in &VARS. Note also the dynamic nature of this approach. Entering nine values in VARS (as we did in the second macro call above) would give us nine observations in &DS.RPT1. What was previously horizontal is now vertical. These observations will form the basis of our PROC REPORT DEFINE statements in the upcoming steps.

STEP 4: CREATING THE DEFINE STATEMENT MACRO VARIABLE

%MakeReport should continue as so:

```
proc sql noprint;
  select 'define ' || cats(name) || ' / display left "' || cats(label) || '"'
    into :define separated by '; '
    from &ds.rpt1;
quit;
%let define=&define%str(;);
```

The point here is to use PROC SQL to create a macro variable that contains all the DEFINE statements for the upcoming PROC REPORT. No macro logic is needed. If you entered *x* variables in VARS you will get *x* DEFINE statements in your PROC REPORT, all separated by semi-colons. Decisions are not made at run time as would be the case when using the SAS macro facility. Instead, they are made prior to run time, when the programmer types in the parameters.

To break this down: we select the values of the NAME column (i.e., the variables in the original data set) and sandwich them amid what will ultimately be PROC REPORT code. We do the same with their labels, which will become the column headers of the report. Remember that labels in PROC REPORT appear in quotes. This is why we use single quotes for the SELECT text and then place lone double quotes on either side of the label value. Note also how we remove leading and trailing blanks in both variables.

After each observation in &DS.RPT1, we add a semi-colon and a space to distinguish lines of code. When it's done, you add a final semi-colon in a %LET statement, and your code is complete.

STEP 5: PRODUCE REPORTS

%MakeReport should conclude as follows:

```
title &ttl;
proc report data=&ds.rpt0 nowd split='|' missing headskip ls=&ls spacing=1;
  column ("&headr" &vars.);
  &define;
  %if &empty=Y %then %do;
    compute before;
      line @1 "No observations match criteria.";
    endcomp;
  %end;
run;
%mend MakeReport;
```

It's as simple as that. SAS resolves &DEFINE to DEFINE statements that you specified when you entered your parameters to begin with. With MPRINT on, the log code would look no different than if the programmer had written it him or herself.

Note the PROC REPORT line size in this example resolves to the LS macro variable value, which was assigned arbitrarily to 125 at the beginning of the macro. Note also that the COMPUTE block in the PROC REPORT is processed only if no observations meet the criteria in the WHERE parameter. Often in patient profiles and listings it's important to know whether patients do not appear in a certain data set.

SPECIFYING COLUMN WIDTHS IN PROC REPORT

Diverse Report Generation can provide the option to assign column widths to each variable or to auto-fit them evenly in the report. This can help prevent ugly text-wrapping when variables become too long for their PROC REPORT columns. To do this, the macro call would look something like this:

```
%MakeReport(ds=ADMH, vars=MHCAT MHTERM MHPTT MHSTDAT MHONGO, width=20 30 30 20 10,
       autofit=, ttl=Medical History, headr=Safety Population, where=SAFETY='Y');
```

The RPT1 DATA step from Step 3 should then be adjusted to produce and keep a WIDTH variable:

```
data &ds.rpt1 (keep=name label width);
  set sashelp.vcolumn (where=(memname="%upcase(&ds.rpt0)"));
  %if &width ne %then %do;
    width=put(scan("&width",_n_),3.);
  %end;
  %else %if %upcase(&autofit)=Y %then %do;
    %let varcount=%sysfunc(countw(&vars));
    %let width=%eval((&ls-&varcount)/&varcount);
    width=put(&width,3.);
  %end;
run;
```

The statement in the first %IF statement scans the values in &WIDTH by observation number and sets each WIDTH value alongside the appropriate variable. Since we used the RETAIN statement in the RPT0 DATA step, the variables are ordered exactly as the programmer wants, so RPT1 will be calibrated perfectly to match observation number with order. Note that the WIDTH variable will ultimately resolve to a WIDTH option in PROC REPORT and has nothing to do with a variable's length.

The new &DS.RPT1 data set should look like this:

NAME	LABEL	WIDTH
MHCAT	Category for Medical History	20
MHTERM	Reported Term for Medical History	30
MHPTT	Preferred Term for Medical History	30
MHSTDAT	Start Date	20
MHONGO	Ongoing?	10

Table 2. The &DS.RPT1 Data Set at Step 3 with the WIDTH Variable

If the AUTOFIT parameter is used, the above code will count the number of variables in &VARS to get an even distribution of the columns in the output. Because the PROC REPORT SPACING option in Step 3 was set arbitrarily to 1, we must divide the variable count not into &LS but into &LS minus variable count. In this case &LS resolves to 125, which means each column can be no wider than 24 since (125 – 5) ÷ 5 = 24. Then we will have room for the 4 additional spaces between columns. Of course, if SPACING=2 then the dividend in the above algorithm should be &LS – (2*&VARCOUNT). In any case, the WIDTH value in RPT1 should ultimately be the same for every observation.

The PROC SQL from Step 4 should then be amended to include WIDTH:

```
proc sql noprint;
  select 'define ' || cats(name) || ' / display left "' || cats(label) ||
    '" width=' || cats(width)
    into :define separated by '; '
    from &ds.rpt1;
quit;
%let define=&define%str(;);
```

Step 5 can remain unchanged. The widths specified in the macro call should be adhered to in the final output.

Please keep in mind that the code as it stands will not work as intended when outputting to RTF using ODS. RTF will not recognize WIDTH statements in a PROC REPORT. When using ODS to produce RTFs, a programmer should construct the DEFINE macro variable not with widths but with percentages of the report's total line size. These should appear in STYLE statements in order to determine column widths. In such an instance, Step 4 would look something like this:

```
proc sql noprint;
  select 'define ' || cats(name) || ' / display left "' || cats(label) ||
    '" style(column)=[cellwidth=' || cats(width) || '%]'
    into :define separated by '; '
    from &ds.rpt1;
quit;
%let define=&define%str(;);
```

Of course, with added complexity should come added error checking. For example, a programmer should check if the number of widths in &WIDTH matches the number of variables in the VARS parameter. A programmer should also sum the widths and the appropriate spacing values to determine if the sum is less than or equal to whatever &LS currently resolves to. Another check might be to ensure that both WIDTH and AUTOFIT are not populated at the

same time.

DOING MORE WITH DIVERSE REPORT GENERATION

Diverse Report Generation requires that data sets be pre-programmed, sorted, and ready for reporting by the time their macro calls are written. It also requires that variable labels become the column headers in the final report. It works best for patient profile, listing, or similar reporting in which data receives little treatment in PROC REPORT. It may not be the best approach if you expect to employ complex COMPUTE blocks or use PROC REPORT to alter your data in any way. So the closer the data sets appear to the final report, the better.

However, complexity in the form of BY, GROUP, ORDER, and BREAK statements can be introduced through additional parameters. Complete error checking and handling will need to be employed for this approach to be truly portable. For example, further tests should be implemented to determine if &DS exists or if the variables listed in &VARS actually exist in &DS.

The power of Diverse Report Generation lies not so much in the code above, but what further can be done with it.

CONCLUSION

When attempting to automate code, programmers should always be aware of alternatives to macro logic. Where macro logic makes decisions at run time, decisions are made prior to run time with Diverse Report Generation. This allows for shorter programming times, more elegant and portable code, and better ways to utilize the power of SAS.

MAKEREPORT MACRO (FULL CODE WITH DOCUMENTATION)

```
*************************************************************************
MakeReport Macro
DS     =Dataset from which output is based
VARS   =Variables to be presented in output, in left-to-right order
WIDTH  =Column widths of variables. Should match order of VARS or be missing
AUTOFIT=Flag for auto-fitting columns in output
TTL    =Title associated with output
HEADR  =Text appearing above variables in output
WHERE  =Criteria with which to subset DS. Default=1
*************************************************************************;
%macro MakeReport (ds=, vars=, width=, autofit=, ttl=, headr=, where=1);
  %global define ls empty;
  %let ls=125;
  %let empty=N;

  *************************
   Step 1: Sub-setting data
  *************************;
  data &ds.rpt0 (keep=&vars);
    retain &vars;
    set &ds (where=(&where));
  run;

  ***********************************************************
   Step 2: Setting macro variable EMPTY to Y and adding blank
           observations if no observations match criteria.
  ***********************************************************;
  data &ds.rpt0;
    if nobs>0 then stop;
    call symput("empty","Y");
    output;
    stop;
    modify &ds.rpt0 nobs=nobs;
  run;

  ********************************************************************
   Step 3: Collecting metadata from the SASHELP view VCOLUMN. Assigning
           WIDTH according to macro parameters.
  ********************************************************************;
  data &ds.rpt1 (keep=name label width);
    set sashelp.vcolumn (where=(memname="%upcase(&ds.rpt0)"));
    %if &width ne %then %do;
      width=put(scan("&width",_n_),3.);
    %end;
```

```
    %else %if %upcase(&autofit)=Y %then %do;
      %let varcount=%sysfunc(countw(&vars));
      %let width=%eval((&ls-&varcount)/&varcount);
      width=put(&width,3.);
    %end;
  run;

  ****************************************************************
   Step 4: Building define statements within a macro variable using
           PROC SQL
  ****************************************************************;
  proc sql noprint;
    select 'define ' || cats(name) || ' / display left "' || cats(label) ||
      '" width=' || cats(width)
      into :define separated by '; '
      from &ds.rpt1;
  quit;
  %let define=&define%str(;);

  ************************
   Step 5: Printing Report
  ************************;
  title &ttl;
  proc report data=&ds.rpt0 nowd split='|' missing headskip ls=&ls spacing=1;
    column ("&headr" &vars.);
    &define;
    %if &empty=Y %then %do;
      compute before;
        line @1 "No observations match criteria.";
      endcomp;
    %end;
  run;
%mend MakeReport;
```

CONTACT INFORMATION

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

Name: Chris Speck
Enterprise: PAREXEL International
Address: 2520 Meridian Parkway, Suite 200
City, State ZIP: Durham, NC, 27713
E-mail: chris.speck@parexel.com