

PROC REPORT Techniques

Steve Prust, Covance, Leeds, UK

ABSTRACT

This paper looks at three diverse but useful techniques for PROC REPORT covering debugging, sub-headings within the report body and a means to control pagination.

INTRODUCTION

PROC REPORT can produce seemingly almost any type of listing or table. But sometimes it can be difficult to manipulate. Here are some techniques that may help. The first technique is a means to inserting a sub-heading within the main body of a report. This type of PROC REPORT intervention may leave you wanting more control over paging which is the subject of the second technique. The last technique addresses one of the thorny problems of debugging, that is how do you examine variable contents within COMPUTE blocks.

CREATING SUB-HEADINGS WITHIN THE MAIN BODY OF A REPORT

Some reports call for sub-headings within the body of a page. Some solutions to doing this in PROC REPORT have been offered that involve creating extra observations in the output dataset. This method opts for the simpler approach of making a variable a GROUPING keyword and taking advantage of the COMPUTE BLOCK feature.

Suppose you wished, in an Adverse Events report where the first column was "Event", to add sub-headings of "System Organ Class" perform the following :

1. add the "System Organ Class" variable – SOC (say) – to the column statement.
2. add a DEFINE statement for SOC with the parameters GROUP and NOPRINT
3. add COMPUTE blocks similar to these :

```
compute before soc;
  line @1 'SOC=' soc $50.;
endcomp;

compute after soc;
  line ' ';
endcomp;
```

This would give output such as :

```
SOC=Muscular-Skeletal
Leg pain
Pain in extremities
Aching joints

SOC=Cardiovascular ...
...
```

If indentation is required this could be achieved by a preparatory data step adding blanks before each event description, e.g. :

```
data final2;
  set final;
  event = ' ' || event;
run;
```

PhUSE 2006

with a define statement of

```
define event / display width=... 'SOC# Event'; /* '#' is the split character */
```

and "SOC=" removed from the compute block the output now looks like :

```
SOC
  Event

Muscular-Skeletal
  Leg pain
  Pain in extremities
  Aching joints

Cardiovascular
  Xxx ...
```

CONTROLLING PAGING WITH PROC REPORT

Controlling where PROC REPORT throws pages sometimes becomes an issue - especially with techniques such as the one above. A radical but effective approach is to not let it do any paging! Instead set up your own paging control.

This is done by firstly, taking the dataset destined for printing and pre-processing it to create two extra variables : one variable for the page number ("page" is an obvious name and causes no problems with PROC REPORT) and one variable to record the current number of lines used on the "virtual page". At the end of this data step write to a global macro variable the number of pages used. Next, wrap the PROC REPORT code within a macro that has a loop of the form :

```
%do i = 1 %to &numpages;

  <PROC REPORT code >

%end;
```

And lastly amend your PROC REPORT code by amending the dataset used and by adding a WHERE statement such as :

```
where page = &i;
```

The data step code might look like this :

```
data paging;
  set final end=last;
  by soc;
  retain page 1 line 0;
  line = line + 1 + (first.soc*2);
  if line > 50 then do;
    page = page + 1;
    line = 1 + (first.soc*2);
  end;
  if last then
    call symput('numpages', compress( put(page,best.) ) );
run;
```

Notice the method by which intervening sub-headings are catered for by use of the first.soc system variable (generated because of the use of the BY statement).

Similar line size calculations could be used when including the FLOW parameter on PROC REPORT DEFINE statements. If the variable comment is given a width of 30 then the line calculation could be varied to

```
line = line + max(1, ceil( length(comment)/30 ) ) + (first.soc*2);
```

Or if there are several columns using flow :

```
line = line + max(1, ceil( length(comment)/30 )
                  ceil( length(action)/15 )
                  ceil( length(outcome/20 ) )
                  + (first.soc*2));
```

Of course PROC REPORT will now be run as many times as there are pages. In testing we have found little difference in how much time was taken to run this new approach as opposed to the old method.

Two further advantages of this method are :

- if you require your report to reference the number of pages it contains then the information is available (e.g. "Page &i of &numpages");
- you can move the footer information on your report from the bottom of the page to directly below the output by using "page" rather than "_page_" in your compute block e.g.

```
compute after page;
  line @1 'Program: xxxx ';
endcomp;
```

DEBUGGING INSIDE A COMPUTE BLOCK

If you are having trouble with your compute block it is always worth considering issues such as scope of the variables you are using. Sometimes you may wish to view the contents of a variable, unfortunately if you place a PUT statement in a compute block, e.g.

```
compute after cohort;
  put subject=;
  put cohort=;
endcomp;
```

you get the following result :

```
ERROR: PUT statement is not valid in this context.
```

However if you cheat using CALL EXECUTE, e.g.

```
compute after cohort;
  call execute ('%put subject=' || put(subject,3.));
  call execute ('%put cohort=' || cohort);
endcomp;
```

What you get is :

```
subject= 1
cohort=1
NOTE: PROCEDURE REPORT used:
      real time          0.04 seconds
      cpu time           0.00 seconds
```

```
NOTE: There were 9 observations read from the data set WORK.FINAL.
```

```
NOTE: CALL EXECUTE routine executed successfully, but no SAS statements were
generated.
```

CONCLUSION

The techniques involving sub-headings and debugging may not need using all the time but are a useful part of the programmers toolkit. The technique of controlling paging has been found to be robust and effective.

PhUSE 2006

CONTACT INFORMATION

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

Steve Prust
Covance
Springfield House,
Hyde Street,
Leeds, UK
LS2 9NG
Email: stephen.prust@covance.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.