Paper CC31

Page X of Y with Proc Report

Chang Y. Chung, Princeton, NJ
Toby Dunn, Manor, TX

ABSTRACT

PROC REPORT is not a straightforward procedure. Many programmers are confused when the data are less than perfectly suited. It is often difficult to discern whether the desired report can be generated with current options or features. For example, what can we do when we need to know which page we are on? As many of us have found out, PROC REPORT has no direct options to know the page number currently being written out. This paper addresses the page number problem. We first show how to find out the current page number. Secondly, we show how to avoid the page number being reset even when we BY-process. Finally, we show a simple and straightforward way to print "page X of Y" at the bottom of the page. It does so by using an EXECUTE call routine and a SYMGET function within a COMPUTE block. We assume the readers be intermediate to advanced SAS® users.

HOW PROC REPORT WORKS

In order to understand how to get page numbers in PROC REPORT, let us review how PROC REPORT builds a report. The procedure first consolidates all GROUP, ORDER, and ACROSS variables to create summary values regardless of whether they are printed or not. Then, it initializes all data step variables to missing. At this point it starts constructing the rows of the report. It does this by first initializing all the report variables to missing, then filling in the rows left to right. When a BREAK or RBREAK condition is reached, PROC REPORT constructs the BREAK values, then executes the statements inside the corresponding COMPUTE block, if there are any (note it does this each time the BREAK or RBREAK condition is reached). Computed variable values are computed at this time and they are written out.

PROC REPORT is designed to be robust and flexible. For flexibility, PROC REPORT has two important and versatile features. The first is "aliasing," which makes it possible to use a variable to get multiple statistics without pre-processing the input data. The second is the COMPUTE blocks that are associated with BREAK conditions. COMPUTE blocks allow users to break a report into multiple parts and insert values and text strings into the parts.

Despite the flexibility, however, doing the "page X of Y"-style page numbers remains non-trivial. SAS Institute has come out with a macro to post-process PROC REPORT output (SAS Institute(a)). Many users, also, have devised means to pre- or post-process data so that this can be done. We introduce one of our own, which is simpler and more straightforward than most. In doing so, we demonstrate a powerful, novel, programming technique of utilizing the flexibility of the PROC REPORT COMPUTE blocks combined with macro processing via CALL EXECUTE and SYMGET.

HOW TO GET PAGE NUMBER

So how does one get the current page number in a PROC REPORT? Recall the COMPUTE blocks mentioned earlier. They can be set to run before or after each page is output. (See Dunn(2004) for this and other interesting uses of the COMPUTE block.) We employ a temporary variable within the COMPUTE block, like so:

```
/* example data set */
data one;
  do var = 1 to 100;
    output;
  end;
run;

options linesize=64 nonumber nodate;
proc report data=one nowd headline missing;
  column var;
  define var / display;

  compute after _page_;                                 /* 1.a. */
    _page + 1;                                          /* 1.b. */
    line "page: " _page;
  endcomp ;
run;
```

Notice that we are using a COMPUTE block with _PAGE_ (1.a.). The block content, then, is executed each time when a page break occurs. The SUM statement (1.b.) in the block increments the page number variable (_PAGE) and retains its value. This way, we do not have to pre- or post-process the data to find out the current page number.

An interesting and useful variation is found when this is implemented with an EXECUTE call routine and a SYMGET function.

```
options linesize=64 nonumber nodate;
%let page = 0;                                          /* 2.a. */
proc report data=one nowd headline missing;
  column var;
  define var / display;

  compute after _page_;
    call execute('%let page = %eval(&page. + 1);');     /* 2.b. */
    _page = input(symget('page'), best.);               /* 2.c. */
    line "page: " _page;
  endcomp ;
run;
```

Here the temporary variable _PAGE is not retained. Instead, we create a global macro variable, PAGE (which is initialized to zero in the line 2.a), to store the current page number. Each time a page break occurs, the CALL EXECUTE pushes a %LET macro statement out. The statement increases the current value of the macro variable PAGE by one (2.b.). Since the %LET statement is executed right away, we can also immediately retrieve the updated value of the macro variable with a SYMGET function (2.c.). There is no magic to this — it is exactly the same way how the CALL EXECUTE works within a DATA step.

HOW TO AVOID AUTOMATIC RESETTING OF PAGE NUMBER

When we use a retained temporary variable for the page number, it is automatically reset when we BY-process the report. The following code will print out a total of six pages. At the bottom of the page, it will print out a string “page:” and the page number that runs from 1 to 3 for the first BY-group (byVar=”a”). Then, the page number is automatically reset to 1 on the fourth page since it is where the second BY-group (byVar=”b”) starts.

```
/* example data set */
data two;
  do byVar = "a", "b";
    do var = 1 to 100;
      output;
    end;
  end;
run;

proc sort data=two out=twoSorted;
  by byVar;
run;

options linesize=64 nonumber nodate;
proc report data=twoSorted nowd headline missing;

  by byVar;                                             /* 3.a */

  column var;
  define var / display;

  compute after _page_;
    _page + 1;
    line "page: " _page;
  endcomp;
run;
```

Using the CALL EXECUTE and SYMGET, we can avoid the automatic resetting. The following will print out “page: 1” to “page: 6” without interruption.

```
options linesize=64 nonumber nodate;
```

```
proc report data=twoSorted nowd headline missing;

  by byVar;

  column var;
  define var / display;

  compute after _page_;
     call execute('%let page = %eval(&page. + 1);');             /* 4.a. */
    _page = input(symget('page'), best.);                        /* 4.b. */
    line "page: " _page;
  endcomp;
run;
```

HOW TO GET THE TOTAL NUMBER OF PAGES

There are many ways to get the total number of pages, but they can be grouped into three: (1) processing data before running PROC REPORT (“pre-processing”); (2) processing the already produced PROC REPORT output (“post-processing”); and (3) letting other people, or software application, count the pages for you (“delegating”).

Pre-processing involves creating “flag” variables in the input dataset or counting the number of rows before generating the report. This way, the total number of pages is already known when running the PROC REPORT. The flag variables are used “over the page to sort the data, cause page breaks, label the current page, calculate the last page, and to determine the end of the report (Trenery, 1999).” Also implementing this strategy are: Casas(2002), Leprince(2003), and Yen and Gudmundson(2000).

In the post-processing, on the other hand, we count either the output lines or some kinds of markers in order to get the total number of pages. A marker can be a form-feed character or a customized string like “@@” (as in the %PAGEOF macro in SAS Institute(a)). There are many interesting variations along the line, see Abernathy, Cheng(2000), Felty and Nicholson(1999), He(2002), Jin, Jin, and Wang(2003), and SAS Institute(a).

By far the most popular way, however, seems to be delegating this task to a word processor that understands RTF NUMPAGES field. By now, we are used to embed raw RTF codes with the help of an ODS escape character (Hamilton(2003), Haworth(2004), SAS Institute(b), Shannon(2002), Smoak(2004), Tong(2003), and Zhou(2001)). Viergever and Vyveman(2003) use DDE to put RTF codes in the document. SAS 9.0 and above make it simple, since both the TITLE and FOOTNOTE statements now understand the (ESCAPE){PAGEOF} option (see 5.a.). McNeil(2002) showcases the following, clean, code:

```
ods escapechar = '\';
title 'This document will have page x of y ' j=r 'Page \{pageof}'; /* 5.a. */
ods rtf file='foo.rtf';
  proc print data=sashelp.class;
  run;
ods rtf close;
```

PAGE X OF Y

Our method is not limited to ODS RTF destination, nor as complicated and cumbersome as pre- or post-processing. We just run the same PROC REPORT code twice. From the first run, we get the total number of pages and discard all the other output. The second time we run it, we write out “page X of Y” using the COMPUTE block and keep the output.

It is easy to package the method into a simple macro, which we call %PAGEXOFY. The macro implementation is straightforward with less than ten macro statements as shown below. Immediately following the macro is a usage example.

```
%macro pageXofY(
    report=        /* proc report code, quoted by %nrstr */
  , dummy=dummy    /* name of the dummy output file */
);

  %local page pages len;                                         /* 6.a. */

  %*-- first run --*;
    %let page = 0;
    %let len = 8;                                                /* 6.b. */
    filename _dummy “&dummy.”;                                   /* 6.c. */
    proc printto print = _dummy; run;
      %unquote(&report.)                                        /* 6.d. */
    proc printto; run;
    filename _dummy clear;

  %*-- second run --*;
    %let pages = &page.;                                        /* 6.e. */
    %let page = 0;
    %let len = %eval(%length(&pages.) * 2 + 4);                 /* 6.f. */
    %unquote(&report.)                                          /* 6.g. */

%mend pageXofY;


/* example data set */
data one;
  do var = 1 to 100;
    output;
  end;
run;


/* example usage */
options linesize=64 nonumber nodate;

%pageXofY(report=%nrstr(                                        /* 7.a. */

  proc report data=one nowd headline missing;
    column var;
    define var / display;

    compute after _page_;
      call execute('%let page = %eval(&page. + 1);');           /* 7.b. */
      length _XofY $&len.;                                      /* 7.c. */
      _XofY = symget('page') || ' of ' || symget('pages');      /* 7.d. */
      line 'page ' _XofY $&len..;                               /* 7.e. */
    endcomp;
  run;

))
```

The macro uses three local macro variables (6.a.): PAGE stores the current page number; PAGES the total number of pages; and LEN the length of the string that constitutes the “X of Y” part of the page number line. The initial value of LEN given in line 6.b. is arbitrary. It is adjusted once the total number of pages is known (6.f.).

The line starting 6.c. shows how we discard unwanted output — we direct the output to a file, by default, named “dummy.” For the first run (6.d.), the macro variable PAGES is not initialized yet, so SYMGET(‘PAGES’) will return nothing. This is OK since we are discarding the output from the first run any way.

The total number of pages is the same as the value of the macro variable PAGE at the end of the first run. Before we run the report second time, we safely copy the total number into the macro variable PAGES (6.e.). Then, PAGE is reset to zero. We also calculate the proper length for the page number string (the “X of Y’’ part only) (6.f.). Finally, the PROC REPORT is run for the second time, generating the desired output (6.g.).

Notice again that above macro allows us to type-in the PROC REPORT code only once. In the PROC REPORT code, we only need one extra COMPUTE block with four lines of code to get the custom page number line. Line 7.b. increments the page number. Lines 7.c. and 7.d. prepare a temporary variable called _XOFY; while the line 7.e. outputs the page number line.

CONCLUSION

This paper shows a novel method of generating “page X of Y” style page number lines using `PROC REPORT`. The method described here is neither limited to the `ODS RTF` destination, nor requires complicated and cumbersome pre- or post-processing. We also demonstrate an interesting way to use `CALL EXECUTE` and `SYMGET` within the `PROC REPORT COMPUTE` block. We encourage readers to try this and other techniques and to continue exploring interesting ways to doing neat things with `PROC REPORT`.

NOTE

The `%NRSTR` macro function quotes the entire `PROC REPORT` code (`7.a.`). The function delays the resolution of the macro parameter `REPORT` until it is explicitly `%UNQUOTE`’d (`6.d.` and `6.g.`). Some people prefer putting the `PROC REPORT` code into a separate macro (say, `%REPORT`) to doing the macro quoting. Under this alternative design, the macro parameter would accept only the name of the separate macro, and the macro invocations become the construct, `%&REPORT.`, which is considered awkward by some.

REFERENCES

Abernathy, Tom. “%pageofpp: a post-processor to handle internal pagination of SAS output.” An alternative to pageof macro. <http://support.sas.com/rnd/base/topics/odsprinter/>. Source is available at <http://support.sas.com/rnd/base/topics/odsprinter/pageofpp_public.sas>.

Casas, Angelina Cecilia. (2002), “Data Dependent Footnotes using PROC REPORT.” Paper tt20. *Proceedings of the Pharmaceutical Industry SAS® Users Group Conference 2002*. <http://www.lexjansen.com/pharmasug/2002/proceed/TechTech/tt20.pdf>.

Cheng, Wei. (2000), “RPT_UTIL: A SAS Macro for Enhancing Reports in Clinicla Trials.” Paper cc01. *Proceedings of the Pharmaceutical Industry SAS® Users Group Conference 2000*. <http://www.lexjansen.com/pharmasug/2000/Coders/cc01.pdf>.

Dunn, Sharon. (2004), “Using Compute Blocks in Proc Report.” Paper CC07. *Proceedings of the Pharmaceutical Industry SAS® Users Group Conference 2004*. <http://www.lexjansen.com/pharmasug/2004/CodersCorner/CC07.pdf>.

Felty, Kelly and Diane Nicholson. (1999), “The Power of PAGEOF (A Valuable Page Numbering Macro).” Coders’ Coner Paper 81. *Proceedings of the SAS Users Group International 24*. <http://www2.sas.com/proceedings/sugi24/Coders/p081-24.pdf>.

Hamilton, Paul. (2003), “ODS to RTF: Tips and Tricks.” Paper 24-28. *Proceedings of the SAS Users Group International 28*. <http://www2.sas.com/proceedings/sugi28/24-28.pdf>.

Haworth, Lauren. (2004), “SAS with Style: Creating your own ODS Style Template for RTF Output.” Paper HW04. *Proceedings of the Pharmaceutical Industry SAS® Users Group Conference 2004*. <http://www.lexjansen.com/pharmasug/2004/HandsOnWorkshops/HW04.pdf>.

He, John. (2002), “ODS to Create Tables in PDF Format from Different Output Sources.” *Proceedings of the North East SAS Users Group Conference 15*. <http://www.nesug.org/html/Proceedings/nesug02/ad/ad009.pdf>.

Jin, Jiang, Ye Jin, and Diane Wang. (2003), “If Only ‘Page 1 of 1000.’” Paper 81-28. *Proceedings of the SAS Users Group International 28*. <http://www2.sas.com/proceedings/sugi28/81-28.pdf>.

Leprince, Daniel J. and Elizabeth Li. (2003), “A Plot and a Table per Page Times Hundreds in a Single PDF File.” Paper 141-28. *Proceedings of the SAS Users Group International 28*. <http://www2.sas.com/proceedings/sugi28/141-28.pdf>.

Landi, Peter C. and Kevin P. Delaney. (2001). “Creating High-Quality Reports in SAS Version 8: How to Make It Look Like Your Boss Wants It.” *Proceedings of the North East SAS Users Group Conference 14*. <http://www.nesug.org/html/Proceedings/nesug01/gr/gr6002.pdf>.

McNeill, Sandy. (2002), “What’s New in the Output Delivery System, Version 9.0.” Paper 128-27. *Proceedings of the SAS Users Group International 27*. <http://www2.sas.com/proceedings/sugi27/p128-

27.pdf>.

Parker, Chevell. (2003), “Generating Custom Excel Spreadsheets using ODS.” Paper 12-28. *Proceedings of the SAS Users Group International 28*. <http://www2.sas.com/proceedings/sugi28/12-28.pdf>.

SAS Institute Inc. (a), “Pageof Macros.” <http://support.sas.com/rnd/base/topics/odsprinter/pageof.sas>.

SAS Institute Inc. (b), “Q: In ODS RTF, can I get my page numbers in page X of Y format?.” *SAS FAQ # 4010. Product or Solution: Base SAS. Operating System: All Operating Systems. Components: ODS:rtf, ODS:template.* <http://support.sas.com/faq/040/FAQ04010.html>.

Shannon, David. (2002), “To ODS RTF and Beyond.” Paper 1-27. *Proceedings of the SAS Users Group International 27*. <http://www2.sas.com/proceedings/sugi27/p001-27.pdf>.

Smoak, Carey G. (2004), “Creating Word Tables using PROC REPORT and ODS RTF.” Paper TT02. *Proceedings of the Pharmaceutical Industry SAS® Users Group Conference 2004*. <http://www.lexjansen.com/pharmasug/2004/TechnicalTechniques/TT02.pdf>.

Tong, Cindy. (2003), “ODS RTF: Practical Tips.” Paper AT007. *Proceedings of the North East SAS Users Group Conference 16*. <http://www.nesug.org/html/Proceedings/nesug03/at/at007.pdf>.

Trenery, David R. (1999), “Jazzing up Your Reports – Some Tricks with PROC REPORT.” Coders’ Coner Paper 80. *Proceedings of the SAS Users Group International 24*. <http://www2.sas.com/proceedings/sugi24/Coders/p080-24.pdf>.

Viergever, William W. and Koen Vyveman. (2003), “Fancy MS Word Reports Made Easy: Harnessing the Power of Dynamic Data Exchange – Against All ODS, Part II .” Paper 16-28. *Proceedings of the SAS Users Group International 28*. <http://www2.sas.com/proceedings/sugi28/16-28.pdf>.

Yen, Peng-fang and Jeff Gudmundson. (2000), Paper 32-25. “One Macro Does It All – Advanced Enhancement of PROC REPORT.” *Proceedings of the SAS Users Group International 25*. <http://www2.sas.com/proceedings/sugi25/25/ad/25p032.pdf>.

Zhou, Jianlin ‘Jay’. (2001), “From SAS ASCII Output to Word Document – A SAS Macro Aproach.” Paper ad10. *Proceedings of the Pharmaceutical Industry SAS® Users Group Conference 2001*. <http://www.lexjansen.com/pharmasug/2001/Proceed/AppDev/ad10_zhou.pdf>.

CONTACT INFORMATION

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

Chang Y. Chung
Princeton University
#216 Wallace Hall
Princeton, NJ 08544
(609) 258-2360
cchung@princeton.edu
http://changchung.com/

Toby Dunn
11604 Marshal St
Manor, Texas 78653
(512) 289-1824
tobydunn@hotmail.com