

Beyond the Basics: Advanced PROC REPORT Tips and Tricks

Allison McMahon, SAS Institute Inc., Cary, NC

INTRODUCTION

This paper is a compilation of tips and tricks for producing the desired PROC REPORT output. The first section covers topics that can be used with listing output and with the Output Delivery System (ODS):

- ordering rows
- moving column headers
- changing summary values
- adding multiple summary rows
- repeating GROUP or ORDER variables
- adding text to the top and bottom of a report
- calculating variables under an ACROSS variable
- aligning decimal values
- creating various percentages

The second section covers topics that can only be used with an ODS destination:

- adding URLs
- changing styles using the CALL DEFINE statement
- adding lines
- using a format to change a style

The final section discusses new features and changes for PROC REPORT in SAS®9.2.

The discussion for each topic includes example code and output. The example output is a representation of one of the output destinations that could be used: listing, HTML, RTF, or PDF. The examples presented in this paper assume an advanced knowledge of PROC REPORT. For more basic information about the REPORT procedure, see *SAS OnlineDoc® 9.1.3* (support.sas.com/onlinedoc/913/docMainpage.jsp).

Let's get started with the tips and tricks ...

USING PROC REPORT FOR LISTING OUTPUT AND ODS DESTINATIONS

ORDERING ROWS

PROC REPORT orders GROUP, ORDER, and ACROSS variables based on the formatted value. This is the default. When the data set is already in the desired sort order, you can use the ORDER=DATA option to keep the order according to the input data set. When you use the ORDER=DATA option, the data is ordered based on all the observations, not just the group. If all possible data values are not in the first grouping, the data output might not be in the intended order. (For more information on the ORDER=DATA option, see "The TABULATE Procedure: Results: TABULATE Procedure, Understanding the Order of Headings with ORDER=DATA" in *SAS OnlineDoc® 9.1.3* (support.sas.com/onlinedoc/913/docMainpage.jsp).

The following example code and output (Figure 1) illustrate using the ORDER=DATA option:

```
DATA test;
INPUT type $ num;
count + 1;
CARDS;
a 1
a 2
b 3
a 4
b 5
b 6
b 7
;
```

```

data new;
set test;
retain hold ' ';
if _n_=1 then do;
  hold=type;
  flag=1;
end;
if hold ne type then do;
  hold = type;
  flag + 1;
end;
run;
proc report nowd nocenter;
col type num count;
  define type / order order=data;
  define num / order order=data;
title 'proc report not in the order of the data';
run;

```

PROC REPORT not in the order of the data		
type	num	count
a	1	1
	2	2
	4	4
b	3	3
	5	5
	6	6
	7	7

Figure 1: Listing Output

To achieve the order of the data according to the input data set, create a counter in the DATA step prior to the REPORT procedure. Use the counter as the first variable in the COLUMN statement and define the variable as an ORDER or GROUP with a NOPRINT option.

The following example code and output (Figure 2) illustrate how to change the ordering of rows:

```

proc report nowd;
col flag type num count;
  define flag / order order=data noprint;
  define type / order order=data;
  define num / order order=data;
title 'PROC REPORT in the order of the data';
run;

```

PROC REPORT in the order of the data		
type	num	count
a	1	1
	2	2
b	3	3
a	4	4
b	5	5
	6	6
	7	7

Figure 2: Listing Output

MOVING COLUMN HEADERS

When a column header is not desired, adding a blank header text of ' ' removes the header text. When removing the header text of an ACROSS variable, a blank row under the ACROSS variable header values appears. The default behavior is to blank out the header text under an ACROSS variable header.

The following example code illustrates the default output of an ACROSS variable header (Figure 3):

```
data example;
input Type $ Color $ Season $ Cost;
cards;
flower purple winter 1000
flower pink   spring 2000
;
run;
ods listing;
ods html file='myhtml.htm';
title 'Blank cells under the ACROSS variable';
proc report nowd data=example;
col type color season, cost;
  define type / group;
  define color / group;
  define season / across ' ';
  define cost / ' ';
run;
ods html close;
```

Blank cells under the ACROSS variable

		spring	winter
type	color		
flower	pink	2000	.
	purple	.	1000

Figure 3: ODS HTML Output

The data values of the ACROSS variable are on a different row than the header text of the other variables that are listed in the COLUMN statement. If you want the ACROSS variable values to be on the same row as the other column headers, then reverse the order of the ACROSS variable. It is not necessary to make any other changes. In ODS, a full row of blank headers is eliminated.

The following example code and output (Figure 4) illustrate how to move column headers under an ACROSS variable header:

```
title 'Changing the position of the ACROSS variable to remove blank cells';
ods html file='myhtml.htm';
proc report nowd data=example;
col type color cost, season;
  define type / group;
  define color / group;
  define season / across ' ';
  define cost / ' ';
run;
ods html close;
```

Changing the position of the ACROSS variable to remove blank cells

type	color	spring	winter
flower	pink	2000	.
	purple	.	1000

Figure 4: ODS HTML Output

Spanned headers add more information to the report header section. When a spanned header is added to an ACROSS variable and column variables, the result is a spanned header on different rows.

The following example code and output (Figure 5) illustrate how spanned headers position column headers on different rows:

```

title 'Spanned headers change the column header positions';
ods html file='myhtml.htm';
proc report nowd data=example;
col ('spanned header 1' type color) ('spanned header 2' cost,season);
  define type / group;
  define color / group;
  define season / across ' ';
  define cost / ' ';
run;
ods html close;

```

Spanned headers change the column header positions

		spanned header 2	
spanned header 1			
type	color	spring	winter
flower	pink	2000	.
	purple	.	1000

Figure 5: ODS HTML Output

To have all the spanned header text on the same header row, remove the spanned header from the ACROSS variable header in the COLUMN statement. In the ACROSS variable DEFINE statement, add a column header with the spanned header text.

The following example code and output (Figure 6) illustrate how to position spanned headers on the same row:

```

title 'Spanned header text on ACROSS variable removes the blank cells';
ods html file='myhtml.htm';
proc report nowd data=example;
col ('spanned header 1' type color) ( cost,season);
  define type / group;
  define color / group ;
  define season / across 'spanned header 2';
  define cost / ' ';
run;
ods html close;

```

Spanned header text on ACROSS variable
removes the blank cells

spanned header 1		spanned header 2	
type	color	spring	winter
flower	pink	2000	.
	purple	.	1000

Figure 6: ODS HTML Output

Typically, when a spanned header is requested, the column headers are under the spanned header row. The following example code and output (Figure 7) illustrate the column headers under the spanned header:

```

title 'Column headers on row below spanned headers';
ods html file='myhtml.htm';
proc report nowd data=example;
col type ('spanned header 1' color cost ) season;
  define type / group;
  define color / group ;
  define season / group ;
run;
ods html close;

```

Column headers on row below spanned
headers

	spanned header 1			
type	color	cost	season	
flower	pink	2000	spring	
	purple	1000	winter	

Figure 7: ODS HTML Output

If you want the non-spanned column headers to be on the same row as the spanned header, then you need to add more spanned headers. The column header in the DEFINE statement is set to blank (' ') and the header text is added to the COLUMN statement as a spanned header.

The following example code and output (Figure 8) illustrate how to move column headers to the spanned header row:

```

title 'Column header text in COLUMN statement changes row position';
ods html file='myhtml.htm';
proc report nowd data=example;
col ('type' type) ('spanned header 1' color cost ) ('season' season);
  define type / group ' ';
  define color / group ;
  define season / group ' ' ;
run;
ods _all_ close;

```

Column header text in COLUMN statement changes row position

type	spanned header 1		season
	color	cost	
flower	pink	2000	spring
	purple	1000	winter

Figure 8: ODS HTML Output

CHANGING SUMMARY VALUES

The SUMMARIZE option in the BREAK and RBREAK statements produces a summary line for a set of observations. The automatic variable `_BREAK_` contains the GROUP or ORDER variable that is listed in the BREAK statement and contains `_RBREAK_` for the RBREAK statement. The values in the `_BREAK_` variable can come in handy when it is important to know when PROC REPORT is at a detail or at a particular summary row, especially if you want to change a value.

An IF statement in a COMPUTE block can be used to query the automatic `_BREAK_` variable. In the example here, when the row is at the break, the analysis variable changes format through the CALL DEFINE statement and is assigned the value of the alias variable. Note that the COMPUTE block is of the alias rather than of the analysis variable. PROC REPORT reads the variables in a left-to-right direction on the COLUMN statement so the alias is further to the right.

The following example code and output (Figure 9) illustrate how to change summary values:

```
ods listing;
ods html file='myhtml.htm';
ods rtf file='myrtf.rtf';
ods pdf file='mypdf.pdf';
proc format;
value stars other='*****';
run;
title 'Change format at BREAK, change value at RBREAK';
PROC REPORT nowd data=sashelp.class;
Where age=12;
col sex age weight weight=mean_weight;
  define sex / order;
  define age / order;
  define mean_weight / mean format=8. noprint;
break after sex / summarize;
rbreak after / summarize;
compute mean_weight;
if upcase(_break_)='SEX' then do;
  call define('weight.sum', 'format', '8.4');
  weight.sum=mean_weight;
end;
if upcase(_break_)='_RBREAK_' then do;
  call define('weight.sum', 'format', 'stars.');
```

Change format at
BREAK, change value at
RBREAK

Sex	Age	Weight
F	12	84.5
		77
F		80.7500
M	12	83
		99.5
		128
M		103.5000
#		*****

Figure 9: ODS HTML Output

ADDING MULTIPLE SUMMARY ROWS

There is currently no option to add multiple summary rows through the BREAK or RBREAK statements. A COMPUTE AFTER block can have multiple LINE statements. The problem with LINE statements is getting the values to line up in the columns, especially when using ODS. The solution is to add variables to the input data set to use as a way to create multiple BREAK summary rows.

Before entering the PROC REPORT code, add a new variable for each additional summary row. For additional summary rows at the end of the report, as shown in the following code, give the variables the same value for every observation. If you want to have multiple summary rows at the BREAK level, assign the new variables the value of a GROUP or ORDER variable that has a BREAK statement associated with it in the PROC REPORT code.

In the PROC REPORT code, add these new variables to the beginning of the COLUMN statement and define the variables as either GROUP or ORDER with a NOPRINT option. Create a separate BREAK AFTER statement for each variable. Keep in mind that the order of a break is right to left in the COLUMN statement and then the RBREAK statement. Use the COMPUTE blocks to assign new values to the existing variable columns. The result will be multiple summary rows that line up neatly in columns.

The following example code and output (Figure 10) illustrate how to add multiple summary rows:

```
DATA example;
INPUT type $ val;
flag1='x'; flag2='y';
CARDS;
Apples 2
Bananas 4
Pears 6
Limes 8
;
PROC REPORT NOWD DATA=example;
COL flag1 flag2 type val val=valavg val=valmin;
  DEFINE flag1 / GROUP noprint;
  DEFINE flag2 / GROUP noprint;
  DEFINE val / SUM 'values';
  DEFINE valavg / MEAN NOPRINT;
  DEFINE valmin / MIN NOPRINT;
BREAK AFTER flag1 / SUMMARIZE;
BREAK AFTER flag2 / SUMMARIZE;
RBREAK AFTER / SUMMARIZE SUPPRESS;
```

```

COMPUTE AFTER flag2;
type='  MIN: ';
val.sum=valmin;
ENDCOMP;
COMPUTE AFTER flag1;
type='  AVG: ';
val.sum=valavg;
ENDCOMP;
COMPUTE AFTER;
type='  SUM: ';
ENDCOMP;
RUN;

```

Multiple Summary Rows	
type	values
Apples	2
Bananas	4
Pears	6
Limes	8
MIN:	2
AVG:	5
SUM:	20

Figure 10: Listing Output

REPEATING GROUP OR ORDER VARIABLES

By default, a GROUP or ORDER variable displays a value for each new grouping and for each time a GROUP or ORDER variable to the left shows a value. However, you might want to have all the rows populated with the GROUP or ORDER value, much like the results of a DISPLAY variable.

If the GROUP or ORDER variable is a character, then the blank rows underneath the printed value can be populated. In the example here, a COMPUTE block uses an IF statement to check the current row cell for a non-blank value. When the cell has a value, the value is assigned to a DATA step variable. When the cell is blank, it is assigned the value of the DATA step variable value.

The following example code and output (Figure 11) illustrate how to repeat GROUP or ORDER variables for every row:

```

PROC REPORT NOWD DATA=sashelp.class;
COL sex weight;
  DEFINE sex / ORDER;
COMPUTE sex;
IF sex NE ' ' then hold=sex;
IF sex EQ ' ' then sex=hold;
ENDCOMP;
RUN;

```

Repeat ORDER/GROUP	
Sex	Weight
F	84.5
F	77
M	83
M	99.5
M	128

Figure 11: Listing Output

If a GROUP or ORDER variable is a numeric, then you will need to create a new variable. In the example here, a COMPUTED variable is added to the right of the ORDER variable in the COLUMN statement. The NOPRINT option is added to the numeric ORDER variable in the DEFINE statement. A COMPUTE block for the COMPUTED variable uses an IF statement to check the current row cell for a numeric value. A DATA step variable is then assigned the numeric value. The COMPUTED numeric variable is then assigned the DATA step value for every row.

The following example code and output (Figure 12) illustrate how to repeat values for numeric ORDER variable rows:

```
PROC REPORT NOWD DATA=sashelp.class;
  COL age newage weight;
  DEFINE age / ORDER NOPRINT;
  DEFINE newage / computed 'AGE';
  COMPUTE newage;
  IF age NE . THEN hold=age;
  newage=hold;
  ENDCOMP;
  RUN;
```

REPEAT numeric GROUP/ORDER	
AGE	Weight
11	50.5
11	85
12	83
12	84.5
12	99.5
12	77
12	128
13	84
13	98
13	84

Figure 12: Listing Output

ADDING TEXT TO THE TOP AND BOTTOM OF A REPORT

One of the features of PROC REPORT that is different from the other Base SAS[®] procedures is the ability to add text. This is done through statements, usually a LINE statement, in a COMPUTE block. PROC REPORT does not support conditional LINE statements—a LINE statement will always print out. A way to achieve conditional text output is to create a DATA step variable and assign it a value that is based on some condition.

A COMPUTE BEFORE _PAGE_ block will activate at the beginning of a logical page. To conditionally print text that is based on a page number, use a counter in the COMPUTE BEFORE _PAGE_ block to determine the logical page and assign a DATA step variable accordingly.

The following code and output (Figure 13) illustrate how to add text to the top of a page:

```
data new;
  set
    sashelp.class
    sashelp.class
    sashelp.class
    sashelp.class
    sashelp.class
    sashelp.class
    sashelp.class;
run;

PROC REPORT nowd data=new;
  compute before _page_;
```

```

count+1;
if count ne 2 then
text='not page 2';
else
text='is page 2';
line text $;
endcomp;
run;

```

PAGE text change			_PAGE_ text change		
not page 2			is page 2		
Name	Age	Sex	Name	Age	Sex
Alfred	14	M	Carol	14	F
Alice	13	F	Henry	14	M
Barbara	13	F	James	12	M
Carol	14	F	Jane	12	F

Figure 13: Listing Output

A COMPUTE AFTER _PAGE_ block can be used to print conditional text as well. When using ODS, if there is an explicit page break that is caused by a PAGE option in the BREAK statement, the LINE statement text for the COMPUTE AFTER _PAGE_ block is honored.

In the following example, the text 'continued' is added to the bottom of the page when an ORDER variable is continued on the next page. The COMPUTE AFTER _PAGE_ block uses the values of the last printed detail row of the page (COMPUTE BEFORE _PAGE_ uses the first detail row of the page). A variable is added to the data set prior to the PROC REPORT code. The variable contains the LAST.VARIABLE value of either 0 or 1. When that variable is set to 0 for the last row, then the text is set to 'continued'; otherwise, the text is blank.

The following example code and output (Figure 14) illustrate how to add text to the bottom of a page:

```

ods rtf file='myfile.rtf';
data test;
retain pageit;
grp='a';
if _n_=1 then pageit=1;
do i=1 to 75;
  counter+1;
  if counter=27 then do;
    pageit+1;
    counter=0;
  end;
  if i=37 then grp='b';
  output;
end;
proc sort;
  by grp;
run;
data new;
set test;
by grp;
myval=last.grp;
run;
proc report nowd ps=40;
col pageit grp myval i;
define pageit / order noprint;
define grp / order;
define myval / display noprint;
compute after _page_;
if myval=0

```

```

    then text='continued';
else text='          ';
line text $9.;
endcomp;
break after pageit / page;
run;
ods rtf close;

```

	23
	24
	25
	26
continued	

Figure 14: ODS RTF Output

Text can also be added at the RBREAK summary row. However, if the cell is for a numeric GROUP or ORDER variable, there is no way to make any changes because PROC REPORT internally sets the cell to blank. As a workaround, create a COMPUTED variable that is the character version of the numeric GROUP or ORDER variable. Use an IF statement to query the automatic `_BREAK_` variable to determine if the current row is at the `_RBREAK_` level. If it is, then change the text of the COMPUTED variable.

The following example code and output (Figure 15) illustrate how to change the text for a numeric ORDER variable at the RBREAK level:

```

PROC REPORT NOWD DATA=sashelp.class;
COL age newage weight height;
  DEFINE age / ORDER NOPRINT;
  DEFINE newage / COMPUTED FORMAT=$8.;
COMPUTE newage/CHAR;
IF age NE . THEN newage=PUT(age,$8.);
ELSE newage=' ';
IF _break_='_RBREAK_' THEN
newage="  total";
ENDCOMP;
RBREAK AFTER/SUMMARIZE;
RUN;

```

change numeric ORDER/GROUP at RBREAK		
newage	Weight	Height
14	112.5	69
	102.5	62.8
	102.5	63.5
	90	64.3
15	112.5	62.5
	112	66.5
	133	67
	112	66.5
16	150	72
total	1027	594.1

Figure 15: Listing Output

When routing output to an ODS destination, a different method can be used. This method is for both character and numeric GROUP and ORDER variables. In a COMPUTE block, query the automatic `_BREAK_` variable to determine if the current row is at the RBREAK level. When it is, use a CALL DEFINE statement and the style attribute of the PRETEXT= option to add the desired text to the cell.

The following example code and output (Figure 16) illustrate how to change the text at an RBREAK level when using ODS:

```
ODS RTF FILE='c:\myrtf.rtf';
PROC REPORT NOWD DATA=sashelp.class;
COL age weight height;
  DEFINE age / ORDER;
COMPUTE age;
IF _break_='_RBREAK_' THEN
  CALL DEFINE('age','style','style=[pretext="total"]');
ENDCOMP;
RBREAK AFTER / SUMMARIZE;
RUN;
ODS _ALL_ CLOSE;
```

Age	Weight	Height
14	112.5	69
	102.5	62.8
	102.5	63.5
	90	64.3
15	112.5	62.5
	112	66.5
	133	67
	112	66.5
16	150	72
total	1027	594.1

Figure 16: ODS RTF Output

CALCULATING VARIABLES UNDER AN ACROSS VARIABLE

In a COMPUTE block, each column under an ACROSS variable has to be referenced by column number, not by name. When a COMPUTED variable is under the ACROSS variable, the number of ACROSS groupings has to be known. Each COMPUTED column needs a separate statement. This can be tedious, and it requires a bit of coding.

An alternative is to macroize the process. In the example code here, the number of ACROSS groupings are found first by sorting the data by the ACROSS variable and setting up a counter to calculate the number of FIRST.ACROSS variables. This value is entered into a macro variable using a CALL SYMPUT statement for use later in the macro. The column number of the first computed variable under the ACROSS variable, the number of variables before the ACROSS variable, and the number of variables under the ACROSS variable are added as input to %LET statements. Once these values are known, a macro %DO loop is used to create the COMPUTED variable calculations. The LIST option in the PROC REPORT statement shows the underlying code.

The following example code and output (Figures 17 and 18) illustrate a calculation under an ACROSS variable using a macro:

```
DATA example;
INPUT flower $ color $ cost;
CARDS;
violet purple 3.50
violet pink 5.00
rose pink 8.50
```

```

rose   purple  9.00
rose   white   3.35
rose   coral   7.50
;
RUN;

PROC SORT DATA=example;
  BY color;
RUN;

DATA _NULL_;
SET example;
  BY color;
IF FIRST.color THEN count+1;
CALL SYMPUT('num',TRIM(LEFT(count)));
RUN;

%MACRO create;
/* the position of the first computed var under the across */
%LET startcol=3;
/* the number of variables before the across */
%LET varsleft=1;
/* number of variables under the across */
%LET varsunder=2;
ods html file='myhtm.htm';
PROC REPORT DATA=example NOWD list ls=100;
COL flower color,(cost sale);
  DEFINE flower / GROUP;
  DEFINE color / ACROSS;
  DEFINE cost / SUM FORMAT=DOLLAR8.2;
  DEFINE sale / COMPUTED FORMAT=DOLLAR8.2;
COMPUTE sale;
%DO i=&startcol %TO %EVAL(&num*&varsunder+&varsleft) %BY &varsunder;
_C%EVAL(&i)_%_C%EVAL(&i-1)_*10;
if _C%eval(&i)_ > 45 then
  call define("_C%eval(&i)_" , 'style', 'style=[background=yellow]');
%END;
ENDCOMP;
RUN;
ods html close;
%MEND;
%create

```

```

COMPUTE sale;
_C3_ = _C2_ * 10;
if _C3_ > 45 then
  call define("_C3_" , 'style', 'style=[background=yellow]');
_C5_ = _C4_ * 10;
if _C5_ > 45 then
  call define("_C5_" , 'style', 'style=[background=yellow]');
_C7_ = _C6_ * 10;
if _C7_ > 45 then
  call define("_C7_" , 'style', 'style=[background=yellow]');
_C9_ = _C8_ * 10;
if _C9_ > 45 then
  call define("_C9_" , 'style', 'style=[background=yellow]');
ENDCOMP;

```

Figure 17: Listing Output in the SAS Log

	color							
	coral		pink		purple		white	
flower	cost	sale	cost	sale	cost	sale	cost	sale
rose	\$7.50	\$75.00	\$8.50	\$85.00	\$9.00	\$90.00	\$3.35	\$33.50
violet	.	.	\$5.00	\$50.00	\$3.50	\$35.00	.	.

Figure 18: ODS HTML Output

ALIGNING DECIMAL VALUES

PROC REPORT provides a way to align the column headers and data values in a LEFT, CENTER, or RIGHT justification. A picture format can be used to produce a decimal alignment. For ODS RTF, the JUST=D option produces a column that is decimal-aligned.

The following example code and output (Figure 19) illustrate decimal place alignment using a picture format:

```
proc format;
  picture twodec low-high=' 99.99';
  picture onedec low-high=' 99.9 ';
  picture nodec low-high='009  ';
run;
data new;
input type $ value;
cards;
x 33.33
y 44.4
z 555
;
run;

ods html file='myfile.htm';
ods pdf file='myfile.pdf';
proc report now data= new;
define value/style(column)=[asis=on];
compute value;
if type='x' then call define('value.sum','format','twodec. ');
if type='y' then call define('value.sum','format','onedec. ');
if type='z' then call define('value.sum','format','nodec. ');
endcomp;
run;
ods _all_ close;
```

Format for a
decimal
alignment

type	value
x	33.33
y	44.4
z	555

Figure 19: ODS HTML Output

The following example code and output (Figure 20) illustrate decimal place alignment using the JUST=D option style attribute:

```
ods rtf file='myfile.rtf';
ods pdf file='myfile2.pdf';
proc report now data= new;
  define value/style(column)=[just=d];
run;
ods _all_ close;
```

type	value
x	33.33
y	44.4
z	555

Figure 20: ODS RTF Output

CREATING VARIOUS PERCENTAGES

PROC REPORT provides a PCTN statistic for a percentage of the overall N and a PCTSUM statistic for a percentage of the overall sum. Other percentages can be calculated through COMPUTE blocks.

To compute the percentage per grouping, use a COMPUTE BEFORE block to assign the value of the analysis variable to a DATA step variable. This will contain the total for the grouping. Then, in a COMPUTE block, use the DATA step variable as the denominator.

The following example code and output (Figure 21) illustrate a calculation of a percentage per grouping:

```
proc report nowd data=sashelp.class;
col age weight pctsum_per_group;
  define age / order;
  define pctsum_per_group / computed format=percent8.2;
break after age / summarize skip;
compute before age;
holdsum=weight.sum;
endcomp;
compute pctsum_per_group;
pctsum_per_group=weight.sum/holdsum;
endcomp;
run;
```

Age	Weight	Percent per group
11	50.5	37.27%
	85	62.73%
11	135.5	100.0%
12	83	17.58%
	84.5	17.90%
	99.5	21.08%
	77	16.31%
12	128	27.12%
	472	100.0%

Figure 21: Listing Output

Percentages across the row can also be calculated. In the following example, the row percentage of the total is under an ACROSS variable. A dummy column was added to the right of the ACROSS variable to help process the percentage calculations. Adding a dummy column eliminates the need for PROC REPORT to move back and forth under the ACROSS variable.

The following example code and output (Figure 22) illustrate a calculation of a percentage under an ACROSS variable:

```
DATA example;
INPUT flower $ color $ cost;
CARDS;
violet purple 3.50
violet pink 5.00
rose pink 8.50
rose purple 9.00
rose white 3.35
rose coral 7.50
;
RUN;
PROC REPORT DATA=example NOWD list ls=100;
COL flower cost=all color,(cost mypct) dummy;
  DEFINE flower / GROUP;
  DEFINE color / ACROSS;
  DEFINE cost / SUM FORMAT=DOLLAR8.2;
  DEFINE mypct / COMPUTED FORMAT=percent8.2;
  DEFINE dummy / NOPRINT;
COMPUTE dummy;
_C4_ = _C3_ / all;
_C6_ = _C5_ / all;
_C8_ = _C7_ / all;
_C10_ = _C9_ / all;
ENDCOMP;
RUN;
```

flower	cost	color							
		coral		pink		purple		white	
		cost	my pct	cost	my pct	cost	my pct	cost	my pct
rose	28.35	\$7.50	26.46%	\$8.50	29.98%	\$9.00	31.75%	\$3.35	11.82%
violet	8.5	.	.	\$5.00	58.82%	\$3.50	41.18%	.	.

Figure 22: Listing Output

An alternative to manually adding the calculations for each percentage column under the ACROSS variable is to macroize the process. The following code illustrates the process using a macro. Using a macro is handy when the number of ACROSS groupings is not known.

The following example code and output (Figure 23) illustrate a calculation of a percentage under an ACROSS variable using a macro:

```
DATA example;
INPUT flower $ color $ cost;
CARDS;
violet purple 3.50
violet pink 5.00
rose pink 8.50
rose purple 9.00
rose white 3.35
rose coral 7.50
;
RUN;
PROC SORT DATA=example;
  BY color;
```

```

RUN;
DATA _NULL_;
SET example;
  BY color;
IF FIRST.color THEN count+1;
CALL SYMPUT('num',TRIM(LEFT(count)));
RUN;
options mprint mlogic symbolgen;
%MACRO autocalc;
/* the position of the first computed variable under the across */
%LET startcol=4;
/* the number of variables before the across */
%LET varsleft=2;
/* number of variables under the across */
%LET varsunder=2;
PROC REPORT DATA=example NOWD list ls=100;
COL flower cost=all color,(cost mypct) dummy;
  DEFINE flower / GROUP;
  DEFINE color / ACROSS;
  DEFINE cost / SUM FORMAT=DOLLAR8.2;
  DEFINE mypct / COMPUTED FORMAT=percent8.2;
COMPUTE dummy;
%DO i=&startcol %TO %EVAL(&num*&varsunder+&varsleft) %BY &varsunder;
_C%EVAL(&i)_%C%EVAL(&i-1)_/all;
%END;
ENDCOMP;
RUN;
%MEND;
%autocalc

```

<pre> COMPUTE dummy; _C4_ = _C3_ / all; _C6_ = _C5_ / all; _C8_ = _C7_ / all; _C10_ = _C9_ / all; ENDCOMP; </pre>
--

Figure 23: Listing Output from the SAS Log

USING PROC REPORT FOR ODS DESTINATIONS ONLY

ADDING URLS

PROC REPORT provides a way to create a hyperlink by using the Uniform Resource Locator (URL) attribute name in a CALL DEFINE statement. Each cell of the column ID becomes a link to the specified URL.

The following example code and output (Figure 24) illustrate how to create a hyperlink using a variable value as part of the URL:

```

ods html file="c:\report.html" ;
data example;
input category $25. site $9.;
count + 1;
cards;
About SAS Institute      corporate
Employment              jobs
News & Events           news
Partners                partners
Software Solutions     solutions
Service & Support      service
;
run;
ods listing close;

```

```

proc report data=example nowd style=[background=yellow];
column count site category;
  define category / group 'Web Selections'
    style=[font_weight=bold background=yellow];
  define count / group noprint;
  define site / group noprint;
compute category;
urlstring = "http://www.sas.com/"||trim(left(site))||"/index.html";
CALL DEFINE(_COL_, 'URL', urlstring );
endcomp;
run;
ods _all_ close;

```

Web Selections
About SAS Institute
Employment
News & Events
Partners
Software Solutions
Service & Support

Figure 24: ODS HTML Output

A LINE statement in a COMPUTE BEFORE or COMPUTE AFTER block can also be used to create a link in ODS HTML. A string that contains HTML tags creates the URL.

The following example code and output (Figure 25) illustrate how to create a URL with a LINE statement:

```

%let html=c;
ods html
  file="&html\report.html" ;
data example;
input category $25. site $9.;
count + 1;
cards;
About SAS Institute      corporate
Employment              jobs
News & Events           news
Partners                partners
Software Solutions      solutions
Service & Support       service
;
ods listing close;
proc report data=example nowd style=[background=yellow];
column count site category;
  define category / group 'Web Selections'
    style=[font_weight=bold background=yellow];
  define count / group noprint;
  define site / group noprint;
compute before _page_;
urlstring='<A '||compress('HREF="http://www.sas.com/'||site||
'/index.html')||">'||trim(site)||'</A>';
line urlstring $;
endcomp;
break after site / page;
run;
ods html close;

```

corporate
Web Selections
About SAS Institute

jobs
Web Selections
Employment

Figure 25: ODS HTML Output

Hyperlinks can also be added by using the URL= option in a STYLE statement. The URL= option in the STYLE statement can be used in any ODS destination.

The following example code and output (Figure 26) illustrate how to use the URL= option in a DEFINE statement:

```
%let html=c;;
ods escapechar='^';
ods html
  file="&html\report.html" ;
data example;
input category $20. type $14.;
count + 1;
call symput('type' || trim(left(count)),trim(left(type)));
call symput('count',trim(left(count)));
cards;
About SAS Institute corporate
Employment          jobs
News & Events       news
Partners            partners
Software Solutions  solutions
Service & Support   service
;
run;
ods listing close;
%macro loopit;
%do i=1 %to &count;
proc report data=example nowd style=[background=yellow]
style(lines)=[protectspecialchars=off];
where type="&&type&i";
column count category;
  define category / group 'Web Selections'
    style=[font_weight=bold background=yellow];
  define count    / group style(header)=[font_weight=bold background=yellow
    url="http://www.sas.com/&&type&i/index.html"] "&&type&i" ;
run;
%end;
%mend;
%loopit
ods html close;
```

corporate	Web Selections
1	About SAS Institute

Figure 26: ODS HTML Output

CHANGING STYLES USING A CALL DEFINE STATEMENT

The CALL DEFINE statement is often used to change the attribute of a cell. Prior to ODS, the only attribute that could be changed in the non-windowing mode was the format. In ODS, there are more options such as STYLE and URL. The first field in the CALL DEFINE statement is the column ID, which requires the variable names from the COLUMN statement to be quoted.

The following example code and output (Figure 27) illustrate how to change a style using numeric expressions in a CALL DEFINE statement:

```
data one;
input id $ a b;
cards;
a 10 9
b 3 2
c 5 3
;
run;
ods html file='myfile.htm';
proc report data=one nowd;
column id a b diff;
  define id / group;
  define diff / computed format=8.;
compute diff;
diff=abs(a.sum-b.sum);
x=_col_;
call define(x-1,'style','style=[background=green]');
endcomp;
run;
ods html close;
```

id	a	b	diff
a	10	9	1
b	3	2	1
c	5	3	2

Figure 27: ODS HTML Output

If the data contains values that are to be used as input to a style attribute, a CALL DEFINE statement allows the insertion of variable values. In the value for the STYLE= option, add the variable or valid DATA step statement enclosed in concatenation bars (|| ||). The concatenation bars allow PROC REPORT to insert the value for the variable.

The following example code and output (Figure 28) illustrate how to change a style using the value for another variable in a CALL DEFINE statement:

```
data test;
input type $ x1 x2 y1 $ y2 $;
cards;
a 1 2 purple green
b 1 2 red pink
;
run;
ods html file='myfile.htm';
proc report nowd data=test;
compute y2;
call define('x1.sum','style','style=[background='||y1||']');
call define('x2.sum','style','style=[background='||y2||']');
endcomp;
run;
ods html close;
```

type	x1	x2	y1	y2
a	1	2	purple	green
b	1	2	red	pink

Figure 28: ODS HTML Output

The CALL DEFINE statement can be nested within an IF statement. In the code here, when the current row is a summary row (the `_BREAK_` variable will have a value), then the CALL DEFINE statement changes the attribute of the cell. If you want to change the attributes for the entire row, use the column ID of `_ROW_` instead. To change the attributes for the entire column, use the `_COL_` variable for the appropriate column ID.

The following example code and output (Figure 29) illustrate how to change a style in a CALL DEFINE statement using IF conditions:

```
ods html file='myfile.htm';
proc report nowd data=sashelp.class;
col sex age name weight;
  define sex / order;
break before sex / summarize;
compute weight;
if _break_ ne ' ' then do;
  if sex='F' then do;
    call define('age.sum', 'style', 'style=[background=green]');
    call define('weight.sum', 'style', 'style=[background=pink]');
  end;
end;
endcomp;
run;
ods html close;
```

Sex	Age	Name	Weight
F	119		811
F	13	Alice	84
	13	Barbara	98
	14	Carol	102.5

Figure 29: ODS HTML Output

ADDING LINES

By default, ODS output contains vertical and horizontal lines (rules) and a frame. Each ODS destination allows different levels of control when producing lines. In ODS HTML, each line location (left, right, top, and bottom) can be controlled using the HTMLSTYLE option.

The following example code and output (Figure 30) illustrate how to add lines in ODS HTML using the HTMLSTYLE option:

```
ods html file='temp.html';
proc report data=sashelp.class nowd
  style(report)={cellspacing=0 rules=groups};
col sex name height;
define sex / order;;
define height / sum;
compute height;
  call define(_col_, "style",
    "style={htmlstyle='border-top:1px solid white;
```

```

border-left:1px solid white'}");
endcomp;
compute name;
call define(_col_,"style",
           "style={htmlstyle='border-top:1px solid white;
           border-left:1px solid white'}");
endcomp;
compute sex;
if sex ne ' ' then
call define(_col_,"style",
           "style={htmlstyle='border-top:1px
           solid white'}");
endcomp;
run;
ods html close;

```

Sex	Name	Height
F	Alice	56.5
	Barbara	65.3
	Carol	62.8
M	Alfred	69
	Henry	63.5

Figure 30: ODS HTML Output

In ODS RTF, the horizontal lines can be added by using RTF control words. For more information on RTF control words, refer to the Microsoft Rich Text Format (RTF) Specification, Version 1.6 ([msdn2.microsoft.com/en-us/library/aa140277\(office.10\).aspx](https://msdn2.microsoft.com/en-us/library/aa140277(office.10).aspx)).

The following example code and output (Figure 31) illustrate how to produce lines with RTF control words:

```

data test;
  length soc event $20 subjid $5;
  soc="Head";
  event="Headache";
  subjid='ABC'; output;
  subjid='DEF'; output;
  subjid='GHI'; output;
  soc="Head";
  event="Bruise";
  subjid='ABC'; output;
  subjid='DEF'; output;
  subjid='GHI'; output;
  soc="Muscular";
  event="Pain";
  subjid='ABC'; output;
  subjid='DEF'; output;
  subjid='GHI'; output;
  soc="Muscular";
  event="Strain";
  subjid='ABC'; output;
  subjid='DEF'; output;
  subjid='GHI'; output;
run;
ODS RTF file="C:\TEMP\ODS_TE4ST.RTF";
proc report data=test nowindows spacing=3 headline headskip
style={rules=cols cellspacing=0 cellpadding=0}
style(header column)=[protectspecialchars=off ];

```

```

column soc event subjid ;
  define soc / "SOC" order width =20 flow;
  define event / "Event" order width=20 flow ;
  define subjid / "Patient" width=10;
compute subjid;
if soc ne ' ' then
  call define('_c1_', 'style', 'style=[pretext="\brdrt\brdrs\brdrw11 " ]');
if event ne ' ' then
  call define('_c2_', 'style', 'style=[pretext="\brdrt\brdrs\brdrw11 " ]');
if subjid ne ' ' then
  call define('_c3_', 'style', 'style=[pretext="\brdrt\brdrs\brdrw11 " ]');
endcomp;

run;

ODS RTF Close;

```

SOC	Event	Patient
Head	Bruise	ABC
		DEF
		GHI
	Headache	ABC
		DEF
		GHI
Muscular	Pain	ABC
		DEF
		GHI
	Strain	ABC
		DEF
		GHI

Figure 31: ODS RTF Output

It's possible to add a line in ODS PDF, but it's not as simple as adding lines in ODS HTML or RTF. To add a line in ODS PDF, IN-LINE formatting is used within a POSTTEXT option. A negative value tells the pointer to back up that number of pixels and insert the character values, such as ' '. Sometimes you need to adjust the negative number and underscores to get the line in the desired location.

The following example code and output (Figure 32) illustrate how to add a line when using ODS PDF:

```

data new;
set sashelp.class;
count+1;
call symput('last',trim(left(count)));
run;
ods escapechar='^';
options missing=' ';
ods pdf file='mypdf.pdf' startpage=never;
title '^S={posttext='^'-500x_____}'this title is underlined";
proc report nowd data=new style=[rules=none frame=void background=white]
style(header)=[background=white protectspecialchars=off posttext="^-
350x_____"];
;
col count name sex weight height age ;
compute age;
if count.sum=&last then
call define(_col_, 'style', 'Style=[posttext="^S={foreground=red}^-
480x_____"]');
endcomp;
run;
ods pdf close;

```

<u>this title is underlined</u>					
<u>count</u>	<u>Name</u>	<u>Sex</u>	<u>Weight</u>	<u>Height</u>	<u>Age</u>
1	Alfred	M	112.5	69	14
2	Alice	F	84	56.5	13
3	Barbara	F	<u>98</u>	<u>65.3</u>	<u>13</u>

Figure 32: ODS PDF Output

USING A FORMAT TO CHANGE A STYLE

An alternative to hard-coding the foreground or background color in a CALL DEFINE statement or a STYLE= option is to use a format. The format is created with the FORMAT procedure. Values are assigned for the desired color. The format is used in place of the color in the CALL DEFINE statement or the STYLE= option.

The following example code and output (Figure 33) illustrate how to use a format in a STYLE= option:

```
proc format;
value colorme low -<0='red'
                0<-high='orange';
run;
data test;
input num;
cards;
-1
0
1
;
run;
ods html file='myfile.htm';
proc report nowd data=test;
define num / display style(column)=[background=colorme.];
run;
ods html close;
```

num
-1
0
1

Figure 33: ODS HTML Output

WHAT'S NEW FOR PROC REPORT IN SAS®9.2

There are some exciting new features and changes for PROC REPORT in SAS®9.2. Three of the following features that have been implemented for SAS®9.2 were suggested on the SASware Ballot.

- Currently, there is no way to concatenate styles using a CALL DEFINE statement. Each time the CALL DEFINE statement is executed, it replaces any previous style information. The STYLE/MERGE attribute name option has been added to allow the styles to be concatenated.
- The BY statement is now allowed when requesting an output data set with the OUT= option in the PROC REPORT statement. In addition, an output data set can be created by using an ODS OUTPUT statement, and this also supports the BY statement.
- The ODS Document now supports PROC REPORT!

- The new Table of Contents (TOC) now supports the CONTENTS= option in the BREAK, RBREAK, and DEFINE statements.
- There will be a new wizard in SAS® Enterprise Guide®, called the List Report Wizard, which uses PROC REPORT to create detail or summary reports.
- The BYPAGENO=*n* option had been added to reset the page number between BY groups. The value of *n* has to be a positive value that is greater than 0. This feature was ranked 26th on the 2007 SASware Ballot.
- The SPANROWS option has been added for the PROC REPORT statement. This option allows the GROUP and ORDER variables to be contained in a box rather than blank cells appearing underneath the GROUP or ORDER variable values. This option was ranked 18th on the 2007 SASware Ballot.
- The SPANROWS option also allows GROUP and ORDER variables values to repeat when the values break across pages in PDF, PS, and RTF destinations. YEAH! This feature was ranked 9th on the 2007 SASware Ballot.

CONCLUSION

PROC REPORT is a powerful and flexible tool to create reports. Although the PROC REPORT default report works for many reporting needs, there are times when customization is desired. The example code in this paper highlights some of the advanced tips and tricks that can be used to achieve customized report output from PROC REPORT.

SAS Technical Support Resources (support.sas.com/techsup/intro.html) offers additional information on PROC Report through links to frequently asked questions, samples, SAS notes, and more.

CONTACT INFORMATION

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

Allison McMahill
SAS Institute Inc.
300 SAS Campus Drive
Cary, NC 27513
E-mail: Allison.mcmahill@sas.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.