

PROC REPORT Unwrapped: Exploring the Secrets behind One of the Most Popular Procedures in Base SAS® Software

Allison McMahill Booth, SAS Institute Inc., Cary, NC, USA

ABSTRACT

Have you ever wondered why a numeric variable is referenced in different forms within a COMPUTE block? Do you know the difference between a DATA step variable and a variable that is listed in the COLUMN statement? Then, this paper is for you! Welcome to PROC REPORT Unwrapped. We are looking at PROC REPORT and uncovering some of the behind-the-scenes details about this classic procedure. We will explore the components associated with PROC REPORT and discover ways to move column headings and change default attributes with styles and CALL DEFINE statements. We will also dig deep into example code and explore the new ability to use multilabel formatting for creating subgroup combinations. So for anyone who has ever written PROC REPORT code, stay tuned. It's PROC REPORT Unwrapped!

INTRODUCTION

Which popular SAS procedure has features of the PRINT, MEANS, and TABULATE procedures and features of the DATA step in a single report-writing tool? It enables you to create a variety of reports including a *detail report*, which contains a row of data for every input data set observation, or a *summary report*, which consolidates data so that each row represents multiple input data set observations. Here is another hint: this same procedure provides the ability to create both default and customized summaries, add text and statistics, and create columns of data that do not exist in the input data set. If you guessed PROC REPORT, you are correct!

For anyone who has written PROC REPORT code and has wondered what is going on behind the scenes, this is the paper for you. This paper explores some of the behind-the-scenes secrets of PROC REPORT. We will dig deep into example code as we begin to uncover some of the details of this classic report-writing procedure. As a bonus, you will discover some facts about the REPORT procedure that you might not have known.

By the way, the code output in this paper is based on the SAS® 9.3 default output destination of HTML. Although most of the paper content can also be applied to the LISTING destination, the code that is shown in this paper is intended to be used in an Output Delivery System (ODS) destination, unless otherwise indicated. With that being said...are you ready to explore? Welcome to PROC REPORT Unwrapped!

EXPLORING THE SECRETS (HOW IT'S MADE)

PROC REPORT first began life as a procedure many years ago in SAS® 6. Since then, it has been gaining popularity as the tool of choice for report writing. Even with such popularity, there are still aspects of the REPORT procedure that can be further explored. In this segment, we will unwrap and explore some of the secrets behind this most popular procedure with a focus on the following components:

- referencing a numeric variable in a COMPUTE block
- exploring the difference between an input data set variable and a DATA step variable
- discovering ways to move column headings
- changing default attributes with styles
- using the CALL DEFINE statement
- exploring the new ability in SAS 9.3 to use multilabel formatting for creating subgroup combinations

Let's start exploring the secrets!

REFERENCING A NUMERIC VARIABLE IN A COMPUTE BLOCK

All numeric variables are referenced the same way, right? Well, that depends on how the numeric variable is defined in the PROC REPORT DEFINE statement. Before we can explore more about the how a numeric variable is defined, we first need to understand some PROC REPORT basics. Then we will explore the many ways a numeric variable

can be defined in the DEFINE statement and how that definition determines the manner in which the variable is referenced in a COMPUTE block.

In the PROC REPORT statement, the input data set is listed using the option DATA= <data set name>. If the DATA= option is not specified, PROC REPORT will use the last data set that was created in the current SAS session. The input data set contains variables and observations. The variables are categorized as either character or numeric—that is it, character or numeric.

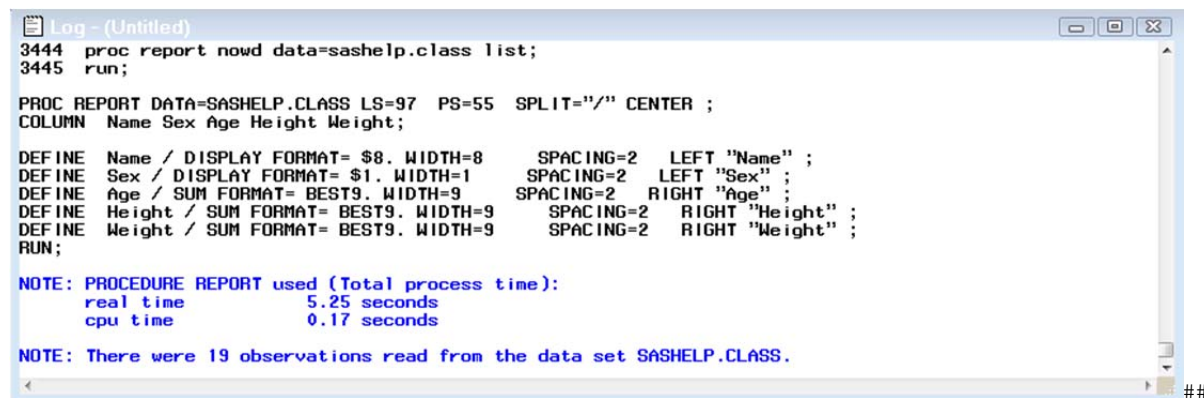
PROC REPORT does not use all of the variables from the input data set. Only the input data set variables that are listed in the COLUMN statement or in the BY statement are used. All of the report items, including the variables from the input data set that are listed in the COLUMN statement can be used in a COMPUTE block. Each report item in the COLUMN statement has an associated DEFINE statement. If a DEFINE statement for the report item is not supplied, PROC REPORT will create a default DEFINE statement behind the scenes.

If a COLUMN statement is not specified, PROC REPORT will create a COLUMN statement behind the scenes. The COLUMN statement will contain only the variables from the input data set in the order of the data set. DEFINE statements can be supplied without a supplied COLUMN statement. The minimum statements that are needed to run PROC REPORT are a PROC REPORT statement with an input data set and a RUN statement. Behind the scenes, PROC REPORT will create all the necessary minimum default statements.

To see the default statements, add the LIST option in the PROC REPORT statement. The LIST option will produce the basic code, including all of the DEFINE statements, in the SAS log. The NOWD option enables the report to run in the non-windowing mode. Here is an example of PROC REPORT code with the LIST option:

```
proc report data=sashelp.class nowd list;
run;
```

The SAS log is shown in Output 1.



```
Log - (Untitled)
3444 proc report nowd data=sashelp.class list;
3445 run;

PROC REPORT DATA=SASHELP.CLASS LS=97 PS=55 SPLIT="/" CENTER ;
COLUMN Name Sex Age Height Weight;

DEFINE Name / DISPLAY FORMAT= $8. WIDTH=8 SPACING=2 LEFT "Name" ;
DEFINE Sex / DISPLAY FORMAT= $1. WIDTH=1 SPACING=2 LEFT "Sex" ;
DEFINE Age / SUM FORMAT= BEST9. WIDTH=9 SPACING=2 RIGHT "Age" ;
DEFINE Height / SUM FORMAT= BEST9. WIDTH=9 SPACING=2 RIGHT "Height" ;
DEFINE Weight / SUM FORMAT= BEST9. WIDTH=9 SPACING=2 RIGHT "Weight" ;
RUN;

NOTE: PROCEDURE REPORT used (Total process time):
      real time      5.25 seconds
      cpu time       0.17 seconds

NOTE: There were 19 observations read from the data set SASHELP.CLASS.

##
```

Output 1. SAS Log Output

By default, the DEFINE statement for a numeric input data set variable that is listed in the COLUMN statement will be associated with the SUM statistic. An alias for the SUM statistic is ANALYSIS. The SUM statistic is the most common statistic that is used in PROC REPORT code. The SUM statistic can be replaced with any valid PROC REPORT statistic such as MIN or MEAN. At BREAK and RBREAK rows, the numeric input data set variable with an associated statistic will consolidate automatically based on the associated statistic.

When a numeric input data set variable with an associated statistic is referenced in a COMPUTE block, the form of the *variable-name.statistic* is used. In a COMPUTE block, if a numeric input data set variable name is used without the corresponding statistic (which is the statistic listed in the DEFINE statement), a note might be written to the SAS log. The following code will produce a note in the SAS log:

```
proc report nowd data=sashelp.class;
  col age height weight total;
  define age / group;
  define height--weight/ mean;
  define total / computed;
  compute total;
    total=height.mean/weight;
  endcomp;
run;
```

In the preceding code, the DEFINE statement for the WEIGHT variable lists MEAN as the statistic. The calculation in the COMPUTE TOTAL block for the TOTAL COMPUTED variable shows the WEIGHT variable without the statistic of MEAN. PROC REPORT requires this statistic and does not recognize the WEIGHT variable. A note, such as the following, is produced in the SAS log:

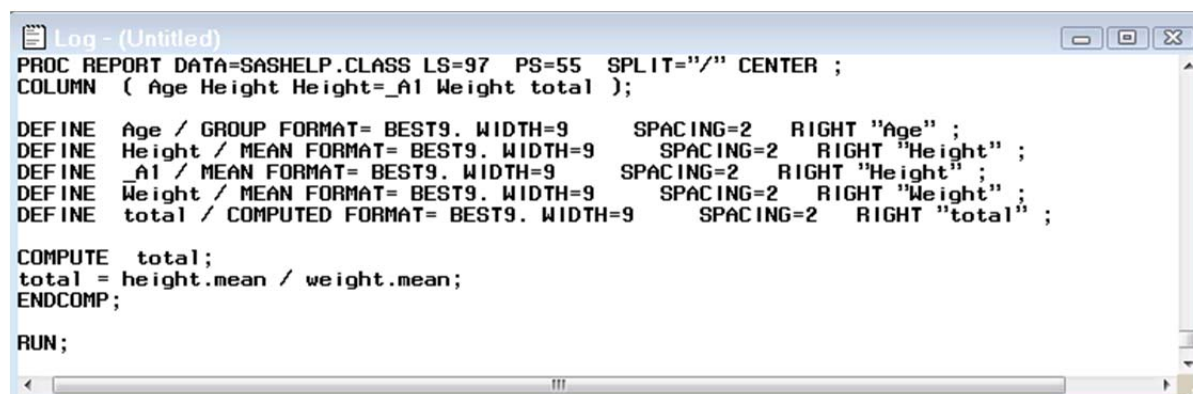
```
NOTE: Variable weight is uninitialized.
```

PROC REPORT allows duplication of report items in the COLUMN statement. This duplicated report item becomes an alias. When an alias of the numeric input data set variable is referenced in a COMPUTE block, the alias name is used without the associated statistic. Behind the scenes, any duplication of the same variable or statistic in the COLUMN statement will be associated with an alias name. If an alias name is not specified, PROC REPORT will create one.

To see the assigned alias name, add the LIST option to the PROC REPORT statement and review the SAS log for the code. Using the preceding code in this section, the HEIGHT variable is duplicated in the COLUMN statement as follows:

```
col age height height weight total;
```

The resulting SAS log is shown in Output 2.



```
Log - (Untitled)
PROC REPORT DATA=SASHELP.CLASS LS=97 PS=55 SPLIT="/" CENTER ;
COLUMN ( Age Height Height=_A1 Weight total );

DEFINE Age / GROUP FORMAT= BEST9. WIDTH=9 SPACING=2 RIGHT "Age" ;
DEFINE Height / MEAN FORMAT= BEST9. WIDTH=9 SPACING=2 RIGHT "Height" ;
DEFINE _A1 / MEAN FORMAT= BEST9. WIDTH=9 SPACING=2 RIGHT "Height" ;
DEFINE Weight / MEAN FORMAT= BEST9. WIDTH=9 SPACING=2 RIGHT "Weight" ;
DEFINE total / COMPUTED FORMAT= BEST9. WIDTH=9 SPACING=2 RIGHT "total" ;

COMPUTE total;
total = height.mean / weight.mean;
ENDCOMP;

RUN;
```

Output 2. SAS Log Output Showing an Alias Name of _A1 Assigned behind the Scenes

When the numeric input data set variable with an associated statistic is associated with an across variable, the column number, in the form of `_Cn_`, is used in a COMPUTE block. In the form of `_Cn_`, `n` is the column number. The position of the columns shown in the output report is based on the left-to-right placement of the *report-items* in the COLUMN statement.

For example, if a numeric variable with an associated statistic is placed as the first column under the ACROSS grouping but it is the second column in the output report, `_C2_` is the correct value to use in a COMPUTE block. Behind the scenes, all columns are considered to have a column number even if the column is not printed in the final output report. Here is an example COLUMN statement:

```
col sex age, (weight height);
```

In this column statement, the first value of the WEIGHT variable is in the second column in the report. AGE is an across variable and is not counted as a column. The first column of the WEIGHT variable is associated with the first value of AGE and is referenced in a COMPUTE block as `_C2_`. The next column of the WEIGHT variable that is associated with the second value of AGE is referenced in a COMPUTE block as `_C4_`.

Each unique value of the across variable becomes a header. Under each ACROSS header are the columns of variables that are associated with each unique across variable value. Each variable associated with an across variable becomes a column under the unique variable value. The number of unique values of an across variable controls the number of columns that are created for a variable associated with the across variable from the COLUMN statement. Behind the scenes, PROC REPORT has to know the specific column placement of a variable that is referenced in a COMPUTE block. The `_Cn_` is used instead of the *variable-name.statistic*, the alias name, or the variable name.

The following example code shows this concept:

```
proc report nowd data=sashelp.class list;
  col age sex, (weight height total);
  define age / group;
  define sex / across;
  define height--weight/ sum format=8.2;
  define total / computed format=8.2;
  compute total;
    _c4_=_c2_/_c3_;
    _c7_=_c5_/_c6_;
  endcomp;
run;
```

The COMPUTE TOTAL block shows two assignment statements. Each assignment corresponds to a column of WEIGHT, HEIGHT, and TOTAL for each unique value of the across variable SEX.

The resulting output is shown in Output 3.

Age	Sex					
	F			M		
	Weight	Height	total	Weight	Height	total
11	50.50	51.30	0.98	85.00	57.50	1.48
12	161.50	116.10	1.39	310.50	181.10	1.71
13	182.00	121.80	1.49	84.00	62.50	1.34
14	192.50	127.10	1.51	215.00	132.50	1.62

Output 3. Output Using `_Cn_` in the COMPUTE TOTAL Calculations

A numeric input data set variable can also be defined as DISPLAY, GROUP, ORDER, or COMPUTED. Because there is no statistic associated with these definitions, the numeric input data set variable name is used in a COMPUTE block. Regardless of the definition, the numeric *report-item* can still be used in any computation. However, for GROUP or ORDER definitions, behind the scenes the values are evaluated from the printed output report instead of the input data. This means that if the ORDER or GROUP defined variable for a particular row and column shows as a blank on the printed output report, a blank is the value that will be used for any computation or evaluation.

The following code shows three different methods for assigning the value of the ORDER variable AGE to a COMPUTED variable.

```
proc report nowd data=sashelp.class;
  col age newage1 newage2 newage3;
  define age / order;
  define newage1 / computed;
  define newage2 / computed;
  define newage3 / computed;
  /* method 1 */
  compute newage1;
    newage1=age*1.5;
  endcomp;
  /* method 2 */
  compute newage2;
    if age ne . then hold_age=age;
    newage2=hold_age*1.5;
  endcomp;
  /* method 3 */
  compute before age;
    before_age=age;
  endcomp;
  compute newage3;
    newage3=before_age*1.5;
  endcomp;
run;
```

In the first method, the value for NEWAGE1 will contain a value only when AGE has a value for the same row. In the second method, the value of NEWAGE2 will contain a value for every row because it is obtaining a value from the DATA step variable HOLD_AGE. In the third method, the value of NEWAGE3 will contain a value for every row because it is obtaining a value from the DATA step variable BEFORE_AGE. The DATA step variable is created in the COMPUTE BEFORE AGE block. Behind the scenes, a DATA step variable changes values only through the code instructions.

Also, behind the scenes, GROUP and ORDER numeric input data set variables are internally set to a blank in the printed output report at the RBREAK level. A COMPUTE AFTER block with an assignment statement for a numeric GROUP or ORDER variable at the RBREAK level will be ignored. A DISPLAY is always set to a blank at the BREAK and RBREAK levels. If you are routing the report output to an ODS destination, using a COMPUTE block CALL DEFINE statement with the STYLE attribute name and a style option that will accept text, such as PRETEXT=, is a way to override the blank values.

A COLUMN STATEMENT VARIABLE VERSUS A DATA STEP VARIABLE

PROC REPORT creates a column type of output report based on the variables and statistics listed in the COLUMN statement. Any variable from the input data set that is to be used as a report column or used in a COMPUTE block has to be listed in the COLUMN statement. The placement of the report items, variables, and statistics in the COLUMN statement is very important.

PROC REPORT reads and processes the report items from the COLUMN statement in a left-to-right, top-to-bottom direction. Until the report item is processed, it will be initialized to missing for numeric variables and blank for character variables. Once the entire COLUMN statement *report-items* are processed for a row, PROC REPORT reinitializes all of the *report-items* back to missing for numeric and blank for character variables. Then PROC REPORT begins the process all over again for the next row of data by processing the report items in the COLUMN statement in a left-to-right direction. Behind the scenes, PROC REPORT consolidates all the input data set variables and statistics listed in the COLUMN statements for the execution of RBREAK BEFORE and BREAK BEFORE statements. For example, the RBREAK, meaning the report break, in the following code is calculated first:

```
proc report nowd data=sashelp.class;
  col sex age,(height weight);
  define age / group;
  define height / min format=8.2 'Height min';
  define weight / max format=8.2 'Weight max';
  rbreak before / summarize;
run;
```

The output is shown in Output 4.

		Age											
		11		12		13		14		15		16	
Sex		Height min	Weight max	Height min	Weight max	Height min	Weight max	Height min	Weight max	Height min	Weight max	Height min	Weight max
		51.30	85.00	56.30	128.00	56.50	98.00	62.80	112.50	62.50	133.00	72.00	150.00
F		51.30	50.50	56.30	84.50	56.50	98.00	62.80	102.50	62.50	112.50		
M		57.50	85.00	57.30	128.00	62.50	84.00	63.50	112.50	66.50	133.00	72.00	150.00

Output 4. PROC REPORT Output Showing the RBREAK Values

COMPUTE blocks are also sensitive to the placement of the variables and statistics in the COLUMN statement. As PROC REPORT processes the *report-items* in a left-to-right direction, any associated COMPUTE blocks are also processed in the same order. This means that in a COMPUTE block that is based on a COLUMN statement *report-item*, any referenced variable or statistic to the right of the COMPUTE block variable is missing. Simply put, PROC REPORT does not know about any *report-item* that is to the right of the COMPUTE block variable in the COLUMN statement.

A DATA step variable, also referred to as a *temporary variable*, is different from the COLUMN statement variable. A DATA step variable is created and used in a COMPUTE block. It is not part of the COLUMN statement. The value of the DATA step variable comes directly from the code in a COMPUTE block. DATA step variables are often used in IF statements when there is a comparison of the current row value to that of the value in the DATA step variable.

PROC REPORT recomputes a COMPUTED variable value at every row, including at the BREAK and RBREAK rows. Values are not accumulated. An accumulated value can be calculated quickly using a DATA step variable in a

COMPUTE block because the value changes through the code only. Behind the scenes, DATA step variables used to accumulate values also include values at the BREAK and RBREAK levels. Adding an IF statement to check the value of the `_BREAK_` automatic variable will help control when the accumulations takes place. In the following code, the computed variable `TOTAL_AGE` is the sum of two variables from the COLUMN statement. `ACCUM_AGE` is the accumulated value of `AGE` stored in the DATA step variable `TEMP_AGE`.

```
proc report nowd data=sashelp.class;
  col age total_age accum_age height weight;
  define age / group;
  define height / min format=8.2 'Height min';
  define weight / max format=8.2 'Weight max';
  define total_age / computed;
  define accum_age / computed;
  compute total_age;
    if _break_ eq ' ' then total_age+age;
  endcomp;
  compute accum_age;
    if _break_ eq ' ' then temp_age+age;
    accum_age=temp_age;
  endcomp;
  rbreak after / summarize;
run;
```

The output is shown in Output 5.

Age	total_age	accum_age	Height min	Weight max
11	11	11	51.30	85.00
12	12	23	56.30	128.00
13	13	36	56.50	98.00
14	14	50	62.80	112.50
15	15	65	62.50	133.00
16	16	81	72.00	150.00
		81	51.30	150.00

Output 5. Comparison of the TOTAL_AGE Column and the ACCUM_AGE Column

Notice the difference between the `TOTAL_AGE` column and the `ACCUM_AGE` column in Output 5. The `TOTAL_AGE` and `AGE` values are reinitialized for every row so that the values are not accumulated. The `ACCUM_AGE` and `AGE` values are reinitialized for every row but the `TEMP_AGE` value is not. `TEMP_AGE` is a DATA step variable and is not listed in the COLUMN statement. The result is an accumulated column for `ACCUM_AGE`.

The `_BREAK_` automatic variable will be blank for detail rows. A quick way to determine the value of a `_BREAK_` variable value is to create an output data set with the `OUT=` option in the PROC REPORT statement and examine the `_BREAK_` values in the output data set.

DISCOVERING WAYS TO MOVE COLUMN HEADERS

By default, the column heading values come from the label in the DEFINE statement. If you do not specifically specify a label in your code either in the DEFINE statement or through a LABEL statement, add the LIST option to the PROC REPORT statement, submit your code, and look at the code that is created in the SAS log. Behind the scenes, PROC REPORT will generate the default values it needs to create the output report. One of the default values is the label specified in the DEFINE statement.

All of the column headings from the label option in the DEFINE statement span over a single column with one exception, variables that are defined as across variables. A column heading for an across variable can span over multiple columns. In the COLUMN statement, a comma after the across variable indicates which variable or group of variables are associated with the across variable. An example of PROC REPORT code containing an across variable is shown below:

```

title 'Default Column Headers';
proc report nowd data=sashelp.shoes;
  column Region Product,Sales;
  define Region / group format= $25. "Region";
  define Product / across format= $14."Product";
  define Sales / sum format= DOLLAR12. "Total Sales";
run;

```

Output 6 shows the PROC REPORT example output.

Default Column Headers								
	Product							
	Boot	Men's Casual	Men's Dress	Sandal	Slipper	Sport Shoe	Women's Casual	Women's Dress
Region	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales	Total Sales
Asia	\$62,708	\$11,754	\$119,366	\$8,208	\$152,032	\$2,092	\$25,837	\$78,234
Canada	\$385,613	\$441,903	\$920,101	\$14,798	\$952,751	\$140,389	\$410,807	\$989,350
Pacific	\$123,575	\$662,368	\$426,191	\$48,424	\$390,740	\$26,169	\$219,886	\$399,441
United States	\$448,296	\$1,372,527	\$969,271	\$12,039	\$967,927	\$104,403	\$541,536	\$1,087,987

Output 6. Default Column Heading with an Across Label Spanning over Multiple Columns

Behind the scenes, each unique value of an across variable is transposed from a column to a row. The row data is **not** available for any further processing within the code as it now becomes a column heading. In Output 6, each value of Product becomes a column with the Product value as the column heading. Under each Product column heading value is the Sales variable column heading and data for the particular Product value.

The heading label Total Sales for every column is redundant. The output report would look better if Total Sales were removed from under the Product column heading and placed above the Product column headings. PROC REPORT provides a way to add column heading information that can span over multiple columns by using a SPANNED HEADER. The SPANNED HEADER is used in the COLUMN statement in this way:

```

column ('spanned header text' variable-list)...;

```

The following example code shows three different methods for using the DEFINE statement and SPANNED HEADERS for creating the column heading:

```

proc report nowd data=sashelp.shoes split='*';
  column region ('(1)Total Sales' '(1)Product' '(2)Total Sales*(2)Product'
               product, sales);
  define region / group format= $25. "Region";
  define product / across format= $14. "(3)Total Sales" "(3)Product" ;
  define sales / sum format=DOLLAR12. " " ;
run;

```

You can mix and match the methods. There is no *best practice* for using each method. The method that you choose depends on the look that you want for the column heading.

The output is shown in Output 7.

Moving Column Headers

	(1) Total Sales							
	(1) Product							
	(2)Total Sales (2)Product							
	(3)Total sales							
	(3) Product							
	Boot	Men's Casual	Men's Dress	Sandal	Slipper	Sport Shoe	Women's Casual	Women's Dress
Region								
Asia	\$62,708	\$11,754	\$119,366	\$8,208	\$152,032	\$2,092	\$25,837	\$78,234
Canada	\$385,613	\$441,903	\$920,101	\$14,798	\$952,751	\$140,389	\$410,807	\$989,350
Pacific	\$123,575	\$662,368	\$426,191	\$48,424	\$390,740	\$26,169	\$219,886	\$399,441
United States	\$448,296	\$1,372,527	\$969,271	\$12,039	\$967,927	\$104,403	\$541,536	\$1,087,987

Output 7. Moved Column Headings from Different Methods

The three different methods are numbered in the example code and the output shown in Output 7: method (1) uses multiple SPANNED HEADER text; method (2) uses SPANNED HEADER text with the PROC REPORT SPLIT= character of * to force the text to continue on the next row; method (3) uses multiple labels in the DEFINE statement (you can also use a split character here).

Let's choose method (1) for the column heading and move the column heading to the top row. You can remove the label from the DEFINE statement by replacing the Region text with a blank " " and moving the Region text to a SPANNED HEADER in the COLUMN statement. There are three rows of headers. This means that the text of Region will need to be pushed up to the top row. You can do this by adding blank SPANNED HEADER text after the Region text in the COLUMN statement. Here is the modified PROC REPORT code with method (1) and the column heading text of Region:

```
proc report nowd data=sashelp.shoes split='*';
  column ('Region' ' ' ' ' ' ' ' Region) ('Total Sales' 'Product' Product , Sales);
  define Region / group format= $25. " " ;
  define Product / across format= $14. " " ;
  define Sales / sum format=DOLLAR12. " " ;
run;
```

Output 8 shows the output.

Moving Column Headers

Region	Total Sales							
	Product							
	Boot	Men's Casual	Men's Dress	Sandal	Slipper	Sport Shoe	Women's Casual	Women's Dress
Asia	\$62,708	\$11,754	\$119,366	\$8,208	\$152,032	\$2,092	\$25,837	\$78,234
Canada	\$385,613	\$441,903	\$920,101	\$14,798	\$952,751	\$140,389	\$410,807	\$989,350
Pacific	\$123,575	\$662,368	\$426,191	\$48,424	\$390,740	\$26,169	\$219,886	\$399,441
United States	\$448,296	\$1,372,527	\$969,271	\$12,039	\$967,927	\$104,403	\$541,536	\$1,087,987

Output 8. Moving Column Headings Using Blank SPANNED HEADERS

Behind the scenes, when there is a blank header row and the output is routed to an ODS destination, the blank row is removed automatically. This does not affect the LISTING output. If you want to preserve the blank row, change the blank label on one of the DEFINE statements that is not an across variable to some value. Then add a style

statement for the header, assigning the foreground color to the background color. For example, if your column heading background is purple, then the style statement for the DEFINE statement would look something like this:

```
style(header)=[background=purple foreground=purple]
```

With the background and the foreground assigned to the same color, any text in the label will blend into the background color.

CHANGING DEFAULT ATTRIBUTES WITH STYLES

Beginning with SAS 9.3, the default output destination is HTML. Behind the scenes, PROC REPORT is using the HTMLBLUE style. All the output in this paper all uses this default destination. What if you are not fond of the HTMLBLUE style? Then, what do you do if you want to change the default style of your output report?

If you want to change the style of HTMLBLUE to another style that is supplied in the *Sashelp.Tmplmst* template store, you can run the following code to create a list of all the styles that are available:

```
proc template;
  list styles;
run;
```

You can apply the styles by adding an ODS statement with the specified style before the PROC REPORT statement. For example, if you want to use the FESTIVAL style instead of the default HTMLBLUE style, the ODS statement would look similar to this:

```
ods html style=festival;
```

PROC REPORT also provides the ability to change the styles of the different report locations. Here are the style location values and a description for each that indicates which part of the report is affected:

- REPORT—the report as a whole
- HEADER|HDR—the column headings
- COLUMN—the column cells
- LINES—the lines generated by LINE statements
- SUMMARY—the summary rows created from BREAK and RBREAK statements
- CALLDEF—the cells identified by a CALL DEFINE statement

All of the style locations are valid in the PROC REPORT statement. These styles apply to the entire location that is specified. The style locations can also be combined if the same attribute is being applied to multiple locations. This is the correct syntax:

```
style<(location(s))>=<style-element-name>[<style-attribute-specification(s)>]
```

The following code shows how to apply the styles in the PROC REPORT statement:

```
ods html style=festival;
title 'Styles on the PROC REPORT statement';
proc report nowd data=sashelp.class(obs=5) split='*'
  style(report)=[outputwidth=7in]
  style(column)=[background=lavender]
  style(header)=[foreground=green]
  style(summary)=[background=purple foreground=white]
  style(lines)=[background=lime]
  style(calldef)=[background=yellow foreground=black];
  column name age sex weight height;
  define name / display;
  define age / order;
  define sex / display;
  define height--weight / sum;
  break after age / summarize;
  rbreak after / summarize;
  compute before;
    line 'this is the beginning';
  endcomp;
```

```

compute age;
  if _break_ ne ' ' then
    call define('age', 'style', 'style=[pretext="total"]');
endcomp;
run;

```

The `STYLE<(location)>` options in the preceding PROC REPORT statement are formatting the output in this way:

- `style(report)` sets the report output width to 7 inches.
- `style(column)` sets the background for all of the columns to lavender.
- `style(header)` applies a green foreground to all of the headers.
- `style(summary)` sets all of the summary rows created from BREAK and RBREAK statements with a background of purple and a foreground of white.
- `style(lines)` sets the line statements to a background of lime.
- `style(calldef)` sets the foreground to black and background to yellow for the CALL DEFINE locations.

The resulting report output is shown in Output 9.

Styles on the PROC REPORT statement

Name	Age	Sex	Weight	Height
this is the beginning				
Alice	13	F	84	56.5
Barbara		F	98	65.3
	total 13		182	121.8
Alfred	14	M	112.5	69
Carol		F	102.5	62.8
Henry		M	102.5	63.5
	total 14		317.5	195.3
	total		499.5	317.1

Output 9. Changing Default Styles in the PROC REPORT Statement

The DEFINE statement supports two types of styles: `STYLE(COLUMN)` and `STYLE(HEADER)`. `STYLE(COLUMN)` applies to the entire column but will not override any styles that are applied to other locations in the column. Using the same code in this section, you can modify the DEFINE statement for the NAME variable that creates the Name column like this:

```

define name / display style(column header)=[background=plum];

```

The background of the HEADER and COLUMN locations for the NAME variable is set to plum. Because styles were applied already to the SUMMARY location, only the header and detail cells for the NAME column are changed to plum. A CALL DEFINE statement is used to override the SUMMARY style for the NAME column. The CALL DEFINE statement is discussed more in the next section.

Output 10 is the resulting report output.

Name	Age	Sex	Weight	Height
this is the beginning				
Alice	13	F	84	56.5
Barbara		F	98	65.3
	total 13		182	121.8
Alfred	14	M	112.5	69
Carol		F	102.5	62.8
Henry		M	102.5	63.5
	total 14		317.5	195.3
	total		499.5	317.1

Output 10. Changing the Default Styles for the NAME Column Using a DEFINE Statement

The BREAK and RBREAK statements support style changes for summary lines, customized lines, or both. A summary line is created from the BREAK or RBREAK statements. A customized line is created from a LINE statement within a COMPUTE BEFORE <target> or a COMPUTE AFTER <target> COMPUTE block. The <target> is a *break-variable* that is defined as either GROUP or ORDER or the _PAGE_ location.

A style on the BREAK and RBREAK statements will not override a cell style that is created by a CALL DEFINE statement or the STYLE(CALLDEF) option in the PROC REPORT statement. A CALL DEFINE statement will be used to make the style changes in this case. Using the same code in this section, you can modify the RBREAK statement like this:

```
rbreak after / summarize style=[background=pink foreground=black font_weight=bold];
```

The COMPUTE BEFORE <target> or a COMPUTE AFTER <target> supports a style option in the COMPUTE statement. A forward slash '/' precedes the style option in the COMPUTE statement. The style option only applies to the LINE statement and will override any previous STYLE(LINES) requests. The style applies to all of the LINE statements within the COMPUTE block. Using the code from this section, a COMPUTE AFTER AGE block is added to show a style modification to the foreground of the LINE statement output.

```
compute after age/ style=[foreground=red];
  line ' this is after age';
endcomp;
```

A CALL DEFINE is a statement within a COMPUTE block. To change a style using a CALL DEFINE statement, the STYLE attribute is specified for the *attribute-name* and the style option is specified as the *value*. The following is the syntax for a CALL DEFINE statement:

```
call define (column-id | _ROW_ , 'attribute-name', value);
```

Here is the code with all of the style modifications:

```
ods html style=festival;
title 'Changing Default Attributes with Styles';
proc report nowd data=sashelp.class(obs=5) split='*'
  style(report)=[outputwidth=7in]
  style(column)=[background=lavender]
  style(header)=[foreground=green]
  style(summary)=[background=purple foreground=white]
  style(lines)=[background=lime]
  style(calldef)=[background=yellow foreground=black];
  column name age sex weight height;
  define name / display style(column header)=[background=plum];
  define age / order;
  define sex / display;
  define height--weight / sum;
  break after age / summarize;
  rbreak after / summarize style=[background=pink foreground=black
                                font_weight=bold];

  compute before;
    line 'this is the beginning';
  endcomp;
  compute age;
    if _break_ ne ' ' then
      call define('age', 'style', 'style=[pretext="total"]');
  endcomp;
  compute after age/ style=[foreground=red];
    line ' this is after age';
  endcomp;
run;
```

The updated output is shown in Output 11.

Changing Default Attributes with Styles

Name	Age	Sex	Weight	Height
this is the beginning				
Alice	13	F	84	56.5
Barbara		F	98	65.3
	total 13		182	121.8
this is after age				
Alfred	14	M	112.5	69
Carol		F	102.5	62.8
Henry		M	102.5	63.5
	total 14		317.5	195.3
this is after age				
	total		499.5	317.1

Output 11. Final Report Output with Changes to Default Attributes Using Style Options

You also can change styles by using inline formatting. *Inline formatting* is a feature of the Output Delivery System that enables you to insert simple formatting text into ODS output by using the ODS ESCAPECHAR statement. For example, here is a TITLE statement and the resulting output:

```
title 'This is ^{style [color=red font_weight=bold] RED}';
```

This is **RED**

The inline formatting in the TITLE statement changes the text of RED to the color of red. The caret (^) in the TITLE statement is the declared ODS ESCAPECHAR. The ODS ESCAPECHAR statement has to be submitted before any inline formatting will take place. The caret (^) can be any unique character that would not normally be in your code.

USING THE CALL DEFINE STATEMENT

The previous section discussed using the CALL DEFINE statement as a way to change a style by specifying the STYLE attribute for the *attribute-name* and the STYLE= option for the *value*. As mentioned earlier, this is the syntax for the CALL DEFINE statement:

```
call define (column-id | _ROW_ , 'attribute-name', value);
```

The *column-id* is the column name or the column number. The *column-id* can be specified as one of the following:

- a character literal (in quotation marks) that is the column name
- a character expression that resolves to the column name
- a numeric literal that is the column number
- a numeric expression that resolves to the column number
- a name of the form *_Cn_*, where *n* is the column number
- the automatic variable *_COL_*, which identifies the column that contains the *report-item* to which the compute block is attached

ROW is an automatic variable that indicates that the *value* is to be applied to the entire row. Currently, the *_ROW_* variable is applicable only with the STYLE attribute name.

Behind the scenes, all of the COLUMN statement *report-items* are used to create the report. The columns created from the COLUMN statement *report-items* are placed in the same order, left to right. Each created column has a column number, beginning with '1' for the left-most column on the report. All *report-items* have a column number, even if there are NOZERO, NOPRINT, and COMPLETECOLS options specified, because these options are applied after the report is created in memory. The following code shows the column number:

```

data test;
  input type $ color $ counter;
  cards;
  aaa purple 1
  aaa orange 1
  bbb purple 2
  ccc orange 2
  ;
run;
proc report nowd data=test missing ;
  col counter type,color,counter=num;
  define counter / group ' ';
  define type / across ' ';
  define color / across ' ';
  define num / sum ' ' nozero;
  compute num;
  call define(4,'style','style=[background=purple]');
endcomp;
run;

```

Output 12 shows the output.

	aaa		bbb	ccc
	orange	purple	purple	orange
1	1	1	.	.
2	.	.	2	2

Output12. PROC REPORT Output with the Incorrect Column Number Used in a CALL DEFINE Statement

In the code above, the CALL DEFINE statement applies a purple background to the fourth column. There is a NOZERO option in the DEFINE statement for NUM, which instructs the report to not print that column if all the column values are zero or missing. By adding the SHOWALL option to the PROC REPORT statement and resubmitting the code, the resulting output in Output 13 shows the fourth column with a purple background. The SHOWALL option displays all of the NOPRINT option and NOZERO option columns in the output report. This option, with the LIST option, is good to use when debugging PROC REPORT code.

```

proc report nowd data=test missing showall;

```

	aaa		bbb		ccc	
	orange	purple	orange	purple	orange	purple
1	1	1		.	.	.
2	.	.		2	2	.

Output 13. Resulting Output When the SHOWALL Option Is Applied to the PROC REPORT Statement

If the intention is to change the background of the fourth column that is shown in Output 13, then here is the correct CALL DEFINE statement:

```

call define(5,'style','style=[background=purple]');

```

There is no limit to the number of CALL DEFINE statements that can be used in a COMPUTE block. If there are duplicate styles that need to be applied to different cells, you might want to consider consolidating the CALL DEFINE statements. Behind the scenes, PROC REPORT calls on the SAS DATA step compiler when a COMPUTE block is used. Most of the SAS DATA step code functionally is available to you when you create code for a COMPUTE block. One consolidation technique is to use a DO loop with a CALL DEFINE to loop through the column number to apply a style. Using the code in this section, here is a modification to the COMPUTE NUM block:

```

compute num;
  call define(_row_, 'style', 'style=[background=wheat]');
  do purple_column= 3 to 5 by 2;
    call define(purple_column, 'style', 'style=[background=purple foreground=white
      font_weight=bold]');
  end;
endcomp;

```

The output is shown in Output 14.

	aaa		bbb	ccc
	orange	purple	purple	orange
1	1	1	.	.
2	.	.	2	2

Output 14. Output Using Modified Code from the COMPUTE NUM Block

We have seen examples of using the attribute name of STYLE. There are other attribute names that can be used. For example, if you want to make the contents of each cell a link to a specified Uniform Resource Locator (URL), you can use the URL attribute as the *attribute-name* and the link as the *value*.

Before ODS, and yes, there was a time before ODS, there was the Output Window (known now as the LISTING destination). The only attribute that is specified in a CALL DEFINE statement for use in the Output Window is the FORMAT attribute. Once ODS was introduced in SAS® 7, the ability to use the FORMAT attribute included all output destinations. _ROW_ cannot be used when the FORMAT attribute name is specified in the CALL DEFINE statement. The best use of the FORMAT attribute can be illustrated by using the output from a PROC MEANS using the default statistics. The following PROC MEANS code creates an output data set and a PROC PRINT to print the output:

```

proc means data=sashelp.class nway;
  where age=15;
  class age;
  var weight height;
  output out=means_output;
run;
proc print;
run;

```

The output is shown in Output 15.

Obs	Age	_TYPE_	_FREQ_	_STAT_	Weight	Height
1	15	1	4	N	4.000	4.0000
2	15	1	4	MIN	112.000	62.5000
3	15	1	4	MAX	133.000	67.0000
4	15	1	4	MEAN	117.375	65.6250
5	15	1	4	STD	10.419	2.0966

Output 15. PROC PRINT Output

In looking at the output in Output 15, it really does not make sense for the N statistic for the WEIGHT and HEIGHT variables to have decimals. PROC REPORT allows an easy way to change the format for these two cells by using the CALL DEFINE statement within a COMPUTE block. The following PROC REPORT shows the CALL DEFINE with the FORMAT attribute.

```

proc report nowd data=means_output;
  col age _stat_ weight height;
  define age / order;
  define _stat_ / display;
  define weight / sum format=8.2;
  define height / sum format=8.2;
  compute height;
    if _stat_='N' then do;
      call define('Weight.sum', 'format', '8. ');
      call define('Height.sum', 'format', '8. ');
    end;
  endcomp;
run;

```

The results are shown in Output 16.

Age	STAT	Weight	Height
15	N	4	4
	MIN	112.00	62.50
	MAX	133.00	67.00
	MEAN	117.38	65.63
	STD	10.42	2.10

Output 16. PROC REPORT Output with a Cell Format Change

The first row under the headers in Output 16 shows the N statistic for both the WEIGHT and HEIGHT columns without decimals. Any time there is a need to change the format of a cell within a column, the CALL DEFINE with the FORMAT attribute is the best method to use. The other choice would be to create a computed character variable version of the value with the desired format. But what fun would that be?

EXPLORING MULTILABEL FORMATTING TO CREATE SUBGROUP COMBINATIONS

You might be asking yourself, what is multilabel formatting? Admittedly, the concept of multilabel formatting baffled me at first. I knew other procedures such as PROC TABULATE and PROC MEANS worked with multilabel formatting, and therefore could not envision it with PROC REPORT. *Multilabel formatting* enables PROC REPORT to use a format label or labels for a given range or overlapping ranges to create a combination of subgroups. The multilabel formats are applied to either group or across variables.

It was not until I had a scenario where I needed to create a report with various subgroupings that I began to appreciate using multilabel formatting. Unfortunately, because multilabel formatting was not available for PROC REPORT in the version of SAS that I was using, my only choice was to slice and dice the data prior to the PROC REPORT step. Multilabel formatting is new for PROC REPORT in SAS 9.3.

The multilabel format is created with PROC FORMAT. The option of multilabel within parentheses is applied to the VALUE statement after the format name. A syntax error, such as the following, will occur in the SAS log if the multilabel option is added without the parentheses:

```

ERROR 22-322: Syntax error, expecting one of the following: a quoted string, a
numeric constant, a datetime constant, a missing value, ;, (, LOW, OTHER.
ERROR 202-322: The option or parameter is not recognized and will be ignored.

```

If there are overlapping ranges on the labels of the VALUE statement, error messages such as the following will be created in the SAS log for each overlapping range:

```

ERROR: These two ranges overlap: LOW-16 and 11-13 (fuzz=1E-12).
ERROR: These two ranges overlap: 11-14 and 11-15 (fuzz=1E-12).

```

In the following example PROC FORMAT code, the multilabel option within parentheses is listed after the format name of AGEFMT in the VALUE statement:

```
proc format;
  value agefmt (multilabel)
    11-13  = ' 11 to 13'
    11-14  = ' 11 to 14'
    11-15  = ' 11 to 15'
    11-high = '11 and above'
    low-16  = '16 and below' ;
run;
```

You might have noticed that some of the labels contain leading blanks. Behind the scenes, PROC REPORT applies the format before creating groups and the formatted values are used for ordering. Without the leading spaces, the category of '11 and above' will be the first group printed because an 'a' in 'and' precedes a 't' in 'to' for an ascending ordering schema. Adding leading spaces is a way to ensure the desired grouping order.

In the example PROC REPORT code below, AGEFMT format is added to the DEFINE AGE statement. Notice that there is also the option of MLF. The MLF option is required when multilabel formatting is desired.

```
title "Multilabel Formatting";
proc report data=sashelp.class nowd;
  col sex age ('Mean' height weight);
  define sex / group;
  define age / group mlf format=agefmt. 'Age Groups';
  define height / mean format=6.2 'Height (in.)';
  define weight / mean format=6.2 'Weight (lbs.)';
  rbreak after / summarize;
run;
```

The output is shown below in Output 17.

Multilabel Formatting

		Mean	
Sex	Age Groups	Height (in.)	Weight (lbs.)
F	11 to 13	57.84	78.80
	11 to 14	59.47	83.79
	11 to 15	60.59	90.11
	11 and above	60.59	90.11
	16 and below	60.59	90.11
M	11 to 13	60.22	95.90
	11 to 14	61.94	99.21
	11 to 15	63.01	104.39
	11 and above	63.91	108.95
	16 and below	63.91	108.95
		62.34	100.03

Output 17. Multilabel Formatting HTML Output

The multilabel formatting is applied only to a group or across variable. If you try to apply the MLF option to any other definition, a warning message will be produced. For example, if the group variable is changed to an order variable for the DEFINE AGE statement, the SAS log will show the following warning:

```
WARNING: The MLF option is valid only with GROUP and ACROSS variables. MLF will
have no effect for the variable age.
```

If you need to create a detailed report instead of a summary report, you can change any other group variable to an order variable or add an order variable. For example, using the code in this section, if the DEFINE SEX/GROUP is changed to DEFINE SEX/ORDER, a detailed report showing a row for every observation from the input data set will be produced.

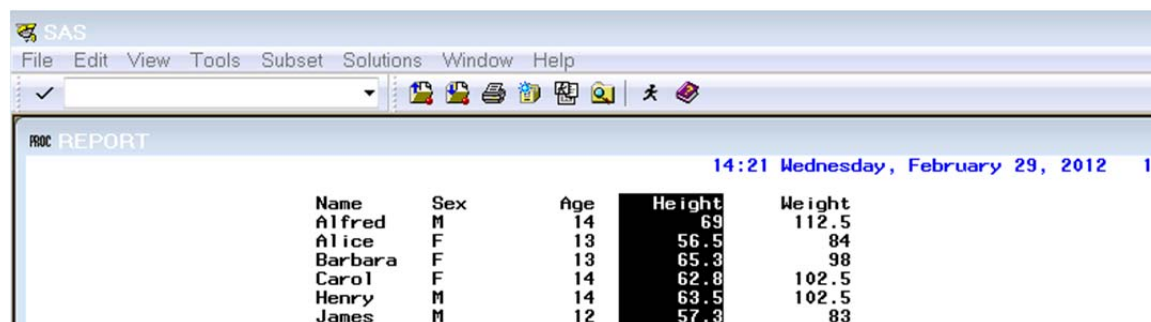
DID YOU KNOW...

Now that you know the behind-the-scenes secrets of PROC REPORT, here are some other little-known facts of interest.

Did you know that PROC REPORT started out as an interactive windowing product and the interactive window is the default environment? Are you not sure what an interactive window is? Most of us have accidentally invoked PROC REPORT code without the NOWD, NOWINDOWS, or the NOFS option and end up in an unfamiliar window. This unfamiliar window is actually the REPORT window. Here is sample PROC REPORT code that invokes the REPORT window:

```
proc report data=sashelp.class;
run;
```

The REPORT window is shown in Display 1.



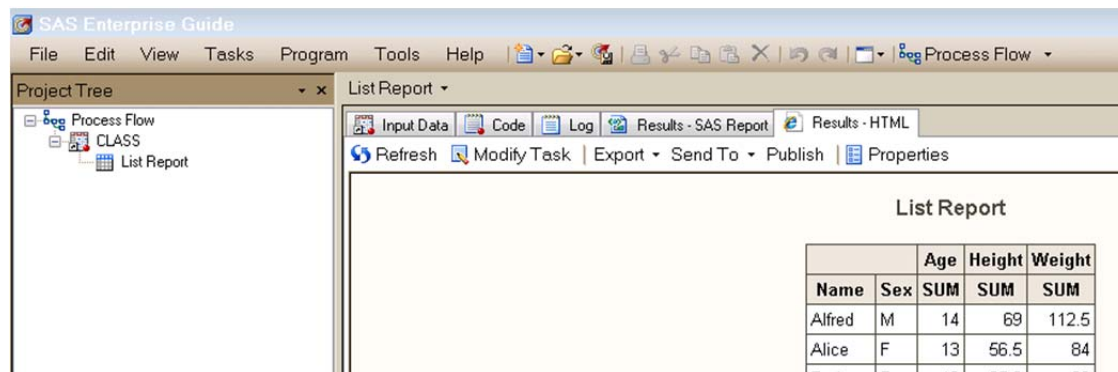
Display 1. The REPORT Window Showing PROC REPORT Code

In fact, the REPORT window can be found in different places of SAS. For example, the **Report Editor** under the **Tools** menu and the **Design Report** selection under **Reporting** in the **Solutions** menu item both invoke the REPORT window. Entering **TREPORT** in the command line box will also invoke the REPORT WINDOW.

For anyone new to PROC REPORT, using the report in the window mode is a wonderful way to quickly create an immediate report. The code can be found in the **Report Statements** selection located in the **Tools** menu from the REPORT window. For experienced PROC REPORT coders, using the REPORT window to create the code saves time typing. Make sure that the NOWD option is added to the PROC REPORT statement when you are running in an editor. As new options are added to PROC REPORT, most of them will also work in the windowing mode. The exception is with ODS. The windowing mode of PROC REPORT does not support any of the ODS functionality. So check it out!

Also, did you know that for SAS® Enterprise Guide® users, there is a wizard that uses PROC REPORT behind the scenes? It is called the List Report wizard. You can invoke the List Report window through the **Describe** selection under the **Tasks** menu item. The List Report wizard was designed for the user who has little to no SAS or PROC REPORT experience. Only the underlying code reveals that PROC REPORT was used behind the scenes.

Display 2 shows the SAS Enterprise Guide List Report wizard.



Display 2. The SAS Enterprise Guide List Report Wizard

CONCLUSION

So there you have it. We have discovered the secrets behind how PROC REPORT is made by exploring a numeric variable in a COMPUTE block, the difference between an input data set variable and a DATA step variable, and ways to move column headings, change attributes with styles, use the CALL DEFINE statement, and explore the multilabel formatting. We dug deep into example code and even unwrapped some of the little known facts about PROC REPORT. That is all the time we have and thank you for taking part in PROC REPORT Unwrapped!

RECOMMENDED READING

Booth, Allison McMahill. 2011. "Beyond the Basics: Advanced PROC REPORT Tips and Tricks Updated for SAS® 9.2." *Proceedings of the SAS Global Forum 2012 Conference*. Cary, NC: SAS Institute Inc. Available at support.sas.com/resources/papers/proceedings11/246-2011.pdf.

Booth, Allison McMahill. 2010. "Evolve from a Carpenter's Apprentice to a Master Woodworker: Creating a Plan for Your Reports and Avoiding Common Pitfalls in REPORT Procedure Coding." *Proceedings of the SAS Global Forum 2010 Conference*. Cary, NC: SAS Institute Inc. Available at support.sas.com/resources/papers/proceedings10/133-2010.pdf.

Booth, Allison McMahill. 2007. "Beyond the Basics: Advanced PROC REPORT Tips and Tricks." *Proceedings of the SAS Global Forum 2007 Conference*. Cary, NC: SAS Institute Inc. Available at support.sas.com/rnd/papers/sgf07/sgf2007-report.pdf.

SAS Institute Inc. 2012. "Find Your Answer in the SAS Knowledge Base." SAS Customer Support Web Site. Available at support.sas.com/resources/.

SAS Institute Inc. 2012. "REPORT Procedure." *Base SAS® 9.3 Procedures Guide*. Cary, NC: SAS Institute Inc. Available at support.sas.com/documentation/cdl/en/proc/63079/HTML/default/viewer.htm#p0bqogcics9o4xn17yvt2qjbgdpi.htm.

SAS Institute Inc. 2012. "REPORT Procedure Windows." *Base SAS® 9.3 Procedures Guide*. Cary, NC: SAS Institute Inc. Available at support.sas.com/documentation/cdl/en/proc/63079/HTML/default/viewer.htm#p10d8v5dnafqb9n1p35e7kp9q67e.htm.

SAS Institute Inc. 2008. "The REPORT Procedure: Getting Started with the Basics." *Technical Paper*. Cary, NC: SAS Institute Inc. Available at support.sas.com/resources/papers/ProcReportBasics.pdf.

SAS Institute Inc. 2008. "Using Style Elements in the REPORT and TABULATE Procedures." *Technical Paper*. Cary, NC: SAS Institute Inc. Available at support.sas.com/resources/papers/stylesinprocs.pdf.

CONTACT INFORMATION

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

Allison McMahill Booth
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
E-mail: support@sas.com
Web: support.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.