Paper 031-2008

PROC REPORT: Compute Block Basics – Part I Tutorial

Arthur L. Carpenter
California Occidental Consultants

ABSTRACT

One of the unique features of the REPORT procedure is the Compute Block. Unlike most other SAS procedures, PROC REPORT has the ability to modify values within a column, to insert lines of text into the report, to create columns, and to control the content of a column. Through compute blocks it is possible to use a number of SAS language elements, many of which can otherwise only be used in the DATA step.

While powerful, the compute block can also be complex and potentially confusing. This tutorial introduces basic compute block concepts, statements, and usages. It discusses a few of the issues that tend to cause folks consternation when first learning how to use the compute block in PROC REPORT.

This paper is being presented in conjunction with the Hands-on Workshop PROC REPORT: Compute Block Basics –Part II Practicum (Paper 188-2008)

KEYWORDS

PROC REPORT, Compute Block, LINE, Report Row Phase

INTRODUCTION

While the topic of compute blocks in the REPORT step is not particularly difficult or advanced, the discussion of compute blocks must necessarily assume that the reader has a basic understanding of the REPORT step itself. In the discussion that follows, an understanding of the COLUMN, DEFINE, BREAK, and RBREAK statements and their relationships to each other is assumed. The reader should also be aware of the different define usages of report items.

There are two basic types of compute blocks; those that are associated with a location (the option BEFORE or AFTER follows the COMPUTE keyword), and those associated only with a report item. While the structure and execution of these two types of compute blocks is similar, how they are used and the timing of their execution can be quite different.

The compute block starts with the COMPUTE statement and terminates with the ENDCOMP statement. Usually the compute block is placed in the REPORT step after the DEFINE statements. The syntax of the compute block looks something like:

```
compute <location> <report_item> </ options>;
      one or more SAS language elements
endcomp;
```

The components of the COMPUTE statement include:

- location — Specifies when the compute block is to execute and ultimately what is to be done with the result of the compute block. Accepted values include BEFORE and AFTER. When a location is specified without also specifying a *report_item*, the location will be at the start (BEFORE) or at the end (AFTER) of the report.
- report_item — When the result of the compute block is associated with a variable or report item, its name is supplied here. This *report_item* variable can be any variable on the COLUMN statement. When *report_item* is a variable that either groups or orders rows (usage of GROUP or ORDER) you may also use BEFORE and AFTER to apply the result at the start or end of each group.

- options — Several options are available that can be used to determine the appearance and location of the result of the compute block.

- SAS language elements
  Any number of SAS language elements can be used within the compute block. These include the use of executable statements, logical processing (IF-THEN/ELSE), and most of the functions available in the DATA step.

The compute block can be placed anywhere within the REPORT step, however generally compute blocks are grouped following the DEFINE statements.

USING THE LINE STATEMENT

One of the more interesting programming statements within the compute block is the LINE statement. This statement is roughly analogous to the PUT statement in the DATA step and can be used to introduce lines of text into the report.

Inserting a Blank Line

In the following example a blank line has been inserted AFTER each level of the grouping variable (REGION).

```
* Blank Line using COMPUTE;
title1 'Using Proc REPORT';
title2 'Blank Line After Region';
proc report data=rptdata.clinics
                (where=(region in('1' '2' '3' '4')))
            nowd;
  column region sex wt,(n mean);
  define region / group format=$6.;
  define sex    / group format=$6. 'Gender';
  define wt     / analysis;
  compute after region;
    line ' ';
  endcomp;
  run;
```

```
Using Proc REPORT
Blank Line After Region

                      weight in pounds
  region  Gender           n        mean
  1       M                4         195

  2       F                6   109.66667
          M                4         105

  3       F                5       127.8
          M                5       163.8

  4       F                4         143
          M               10       165.6
```

The LINE statement has been used to insert a blank line AFTER each regional group.

Because the COMPUTE statement contains the location specification AFTER and also designates the report item REGION, we have effectively requested that the results of the compute block (a blank space) be written **after** each group of the grouping variable REGION.

Adding Lines of Text

In the previous example the LINE statement added a blank line, the LINE statement is more commonly used to add lines of text at specific locations in the report.

In the following example the text is written as a footnote at the end of the report. We can specify the options BEFORE and AFTER to indicate the location, and since in this case no grouping variable appears on the COMPUTE statement, the AFTER applies to the whole REPORT.

```
* Text Line using COMPUTE;
title1 'Using Proc REPORT';
title2 'Footnote Using LINE';
proc report data=rptdata.clinics
                (where=(region in('1' '2' '3' '4')))
            nowd;
  column region sex wt,(n mean);
  define region / group format=$6.;
  define sex    / group format=$6. 'Gender';
  define wt     / analysis;
  compute after region;
    line ' ';
  endcomp;
  compute after;
    line @20 'Weight taken during';
    line @20 'the entrance exam.';
  endcomp;
  run;
```

```
Using Proc REPORT
Footnote Using LINE

                     weight in pounds
  region  Gender           n        mean
  1       M                4         195

  2       F                6   109.66667
          M                4         105

  3       F                5       127.8
          M                5       163.8

  4       F                4         143
          M               10       165.6

                    Weight taken during
                    the entrance exam.
```

This report was generated with the system option NOCENTER in effect (notice the titles). Even so the results of the LINE statement will be centered unless a column specification is provided.

In these LINE statements the @ is used, as it is in the DATA step PUT statement, to designate the column number. If a specific column is not specified with the @, and no justification options are specified, text generated by the LINE statement will be centered. This is a different default behavior than the PUT statement in the DATA step.

When writing to ODS destinations other than LISTING, proportional fonts may make exact placement of values difficult, and may require you to use a trial-and-error approach, and to make things more interesting some destinations ignore the @ altogether.

Writing Formatted Values

Formatted variable values can be placed on the report using the LINE statement, and these values can be placed BEFORE or AFTER each group. In the following example the user defined format $REGNAME provides a text name for the first four regions. These names are than added to the report through the use of a LINE statement and a compute block.

```
proc format;
  value $regname
```

```
    '1' = 'New England'
    '2' = 'New York'
    '3' = 'Maryland'
    '4' = 'South East';
  run;

* Text Line using COMPUTE;
title1 'Using Proc REPORT';
title2 'Formatted Values';
footnote1 'at the bottom';
proc report data=rptdata.clinics(where=(region in('1' '2' '3' '4')))
            nowd;
  column region sex wt,(n mean);
  define region / group format=$6.;
  define sex    / group format=$6. 'Gender';
  define wt     / analysis;
  compute before region;
    line @3 region $regname8.; ❶
  endcomp;
  compute after region;
    line ' '; ❷
  endcomp;
  compute after;
    line @20 'Weight taken during'; ❸
    line @20 'the entrance exam.';
  endcomp;
  run;
```

```
Using Proc REPORT
Formatted Values

                      weight in pounds
  region  Gender             n        mean
  New England ❶
  1       M                  4         195

  New York
  2       F                  6   109.66667
          M                  4         105
❷
  Maryland
  3       F                  5       127.8
          M                  5       163.8

  South East
  4       F                  4         143
          M                 10       165.6

                    Weight taken during ❸
                    the entrance exam.
```

Three compute blocks are used to generate the three types of text. A formatted region name BEFORE each region ❶, a blank line after each region summary ❷, and footnote text at the end of the report ❸.

The format $REGNAME could have also been used in the DEFINE statement, however in this case we wanted to show the unformatted value as well as the formatted group header.

USING SAS LANGUAGE ELEMENTS

Much of the power of the DATA step is available within the compute block. This radically increases the flexibility of the REPORT step as most of the SAS language elements, such as, routines, functions, arithmetic operations, and executable

statements can also be used within the compute block. These include DO loops, assignment and sum statements, ARRAYs, and IF-THEN/ELSE processing.

Most of the functions and routines that do not work in the compute block are those that utilize either the DATA step's Program Data Vector, PDV, or the processing of the DATA step itself. Examples include the LAG function and the RETAIN statement. As was already discussed, the LINE statement replaces the PUT.

The use of %INCLUDE statements, macro variables, and macro invocations work the same in compute blocks as they do in other parts of SAS. In the case of %INCLUDE statements and elements of the macro language, it is less a matter of these items working in a compute block, and is more accurately simply a property of Base SAS and how these elements apply to all procedures.

The following example demonstrates, in a simple way, some of this power. This compute block performs a transformation of weight from pounds to Kilograms. The conversion is done in a compute block with the same name as the variable (report item) that is being modified. The assignment statement in the compute block is of the same form as you would expect to find in the DATA step.

```
title1 'Using Proc REPORT';
title2 'Converting Weight to Kg';
proc report data=rptdata.clinics(where=(region in('4')))
            nowd;
  column lname fname sex wt;
  define lname  / display;
  define fname  / display;
  define sex    / display format=$6. 'Gender';
  define wt     / display 'Weight in Kg'
                  format=6.2;

  compute wt;
     * Convert pounds to KG;
     wt = wt/2.2;
  endcomp;
  run;
```

```
Using Proc REPORT
Converting Weight to Kg

              first            Weight
  last name   name     Gender   in Kg
  Halfner     John     M         70.45
  Johnson     Randal   M         91.36
  Rodgers     Carl     M         81.36
  Cordoba     Juan     M         60.45
  Baron       Roger    M         72.73
  Adams       Mary     F         70.45
  Rymes       Carol    F         59.55
  Most        Mat      M         70.45
  Jackson     Ted      M         91.36
  Maxim       Kurt     M         81.36
  Perez       Mathew   M         60.45
  East        Clint    M         72.73
  Batell      Mary     F         70.45
  Rumor       Stacy    F         59.55
```

The label for the variable WT contains the units (pounds), so a new column header is also required.

REPORT ITEM NAMING RULES

When coding within the compute block, report items and temporary variables will need to be addressed. How they are addressed or named in the compute block depends on what the report item is and how it is being used. There are four different ways of naming report items within a compute block.

Explicitly by Name

While you can have a compute block for any report item (variable on the COLUMN statement), the naming conventions of the report items that are addressed within the compute block are not nearly as straight forward as they would seem to be in the previous two examples. These are about the simplest cases in which the name of the column is used directly (explicitly).

The variable name can be used directly when the variable has a define type of GROUP, ORDER, COMPUTED, or DISPLAY. Temporary variables, variables that are created and only used in a compute block, are also always addressed directly by variable name.

The REPORT step also creates an automatic temporary variable named _BREAK_. This variable is also addressed directly and is available during the execution of the compute block.

Using a Compound Name

Compound variable names are needed, when a variable has a define usage of ANALYSIS. Whether requested or not, a statistic is always calculated for analysis variables. The compound name is a combination of the variable name and the statistic that it has been used to calculate. The general form is *variablename.statistic*, and an example of a compound name might be:

```
wt.mean
```

When a statistic is not otherwise specified it is assumed to be SUM.

By Specifying an Alias

An alias can be specified in the COLUMN statement when you want to use a single analysis variable in more than one way - generally to calculate more than one statistic. When used in a compute block, aliases are named explicitly. The following COLUMN statement generates a series of aliases for the HT analysis variable.

```
columns region ht ht=htmin  ht=htmax
                  ht=htmean ht=htmedian;
```

In the compute block the alias is addressed directly as in:

```
compute after;
   line @3 'Minimum height is ' htmin 6.1;
endcomp;
```

Indirectly Using the Absolute Column Number

Sometimes as the report is constructed a given column may not have a specific name. This is especially the case when a variable, with the define type of ACROSS, creates a series of columns. These, and indeed any column in the report, can be referenced by using the absolute column number as an indirect column reference. This column number is included in a pseudo variable name which is always of the form

```
_Cxx_
```

where the *xx* is the column number as read from left to right on the report. Keep in mind that the column count includes any columns that may ultimately not be printed *e.g.* those columns defined with NOPRINT or NOZERO.

In the following example SEX is an across variable with WT nested within each distinct value of SEX. With REGION as

the only other variable in the COLUMNS statement, WT appears in _C2_ and _C3_.

```
Title1 'Mean Weight in Kg';
proc report data=sasclass.clinics nowd;
  columns region sex,wt;
  define region / group f=$6.;
  define sex / across '  patient Sex';
  define wt / mean 'Weight' f=6.2;
  compute wt;
    _c2_ = _c2_/2.2;
    _c3_ = _c3_/2.2;
  endcomp;
  run;
```

```
Mean Weight in Kg

            patient Sex
                F        M
  region  Weight   Weight
  1            .     88.64
  10       33.68    80.45
  2        22.66    47.73
  3        26.40    74.45
  4        29.55    75.27
  5        30.21    80.45
  6        38.64    93.33
  7            .     68.64
  8        72.73        .
  9        36.57    86.59
```

In the compute block _C2_ refers to the mean weight of females, while _C3_ refers to the mean weight of males. The order is based on the ORDER= option which has a default value of INTERNAL.

COMPUTE BLOCK EVENT SEQUENCING

An understanding of the sequencing of events when processing a REPORT step become even more important when the step includes one or more compute blocks.

When you have multiple compute blocks they can appear in any order in your REPORT step. The order that you include the blocks does **not** effect when they are executed. I usually try to group my compute blocks after my BREAK and RBREAK statements, and I also try to order them nominally in the same order as they will execute. This arrangement helps me with my coding, but does not change how the compute blocks work.

Report Step Phases

The REPORT step executes in three distinct phases. Having at least a passing understanding of these phases helps the compute block programmer understand why some things work while others do not.

Evaluation Phase

REPORT begins by evaluating all of the REPORT step statements. If any compute blocks are present the SAS language elements and LINE statements are set aside for later.

Setup Phase

Next, after the statements have been evaluated, the Setup Phase uses the MEANS/SUMMARY engine to sort the input data for ORDER and GROUP variables and computes any summarizations. When summarizations or statistics are calculated, the results are held in the computed summary information which is stored in memory.

Report Row Phase

In this phase, REPORT builds each report row using data from the input data set and/or, when needed, the computed summary information. If any compute blocks are present they are executed during this phase. REPORT sends each completed row (one row at a time) to all the ODS destinations (LISTING, PDF, etc.) that are currently open.

Compute Block Events

As the report is generated (one row at a time - top to bottom and left to right - during the report row phase), the various compute blocks are executed at the appropriate time. Remember that compute blocks are tied both vertically and horizontally to locations in the report, and the location controls the timing of when the compute block will be executed. These ties are specified on the COMPUTE statement. Vertical ties are established by using the timing options BEFORE and AFTER, and horizontal ties are made by the specification of report items in the COLUMN statement.

When a compute block is tied to a specific report item (like a variable name) on the COLUMN statement, it will be executed for each report row. The execution will take place when REPORT processes that specific column, and any given row is processed from left to right based on the order of the items on the COLUMN statement. In the following step, because SEX is to the left of WEIGHT on the COLUMN statement, the compute block for SEX will be executed before the compute block for WEIGHT.

```
proc report .......;
  column region sex weight;
  .....
  compute weight;
      .....
  endcomp;
  compute sex;
      .....
  endcomp;
  .....
  run;
```

Although it would make no difference to the processing of the step, I would reorder the code to place the compute blocks in the order that they will be processed. The step would become:

```
proc report .......;
  column region sex weight;
  .....
  compute sex;
      .....
  endcomp;
  compute weight;
      .....
  endcomp;
  .....
  run;
```

Compute blocks that are defined with BEFORE or AFTER on the COMPUTE statement will be executed at the time that the specified event takes place. When BEFORE or AFTER appears on the COMPUTE statement and there is no report item, the compute block will be executed only once - BEFORE or AFTER the report.

Including a BEFORE or AFTER on a compute statement is sufficient to generate a row in the computed summary information during the setup phase. That report row may or may not be written out to the final report depending on the BREAK and RBREAK specifications. In either case as the report rows are processed during the report row phase, the compute block is executed as its associated report row is processed.

The COMPUTE BEFORE statement

```
compute before;
```

will generate what will be the first row in the computed summary information, while the COMPUTE AFTER statement

```
compute after;
```

would generate the last row in the computed summary information and would be the last compute block to execute.

When you have a report item that is a grouping/ordering variable, you can specify the COMPUTE statement with both the BEFORE/ AFTER and the report item. This allows you to execute the compute block when values of the grouping variable change. The processing opportunities are roughly similar to FIRST. and LAST. processing in the DATA step. The following COMPUTE statement sets up a compute block that will be executed just before each new value of REGION.

```
compute before region;
```

It is not at all unusual to have compute blocks that execute both BEFORE and AFTER in the same REPORT step. Compute blocks that execute before the report or group are very useful for initializing variables, while compute blocks that execute after the report or group are more generally used for summaries.

Using a compute block with a BEFORE or AFTER does not require a corresponding BREAK or RBREAK statement. However when both are present in the same step the compute block and its corresponding BREAK or RBREAK statement share a common row in the computed summary information. Remember that the compute block itself does not transfer information to the report (unless it contains a LINE statement). The compute block can be used to modify information on the computed summary information, and if there is a corresponding BREAK/RBREAK statement with a SUMMARIZE option then that information can also be written to the final report.

CREATING AND MODIFYING COLUMNS

One of the great strengths of PROC REPORT is its ability to create columns based on calculations carried out during the execution of the procedure. Columns that are not on the incoming data set can be created and displayed within the PROC REPORT step. Often this can eliminate one or more DATA steps. The compute block provides the power and flexibility to mold the column in a variety of ways.

Compute blocks can be used to create both numeric and character variables, and an extensive number of the SAS language elements that give the DATA step much of its power and functionality are also available for use in the compute block.

Coordinating with the COLUMN and DEFINE Statements

Variables on the data set that is to be processed by PROC REPORT are named on the COLUMN statement and the way that these variables are to be used is specified on the DEFINE statement. This is also true for report item columns created through the use of the compute block. In addition, the name of the computed variable will also appear on the COMPUTE statement.

When a column is created through a compute block, the new column is named and that column name is used on both the COLUMN statement and on a DEFINE statement. Since a new column with have a define usage of COMPUTE, it cannot also be a grouping or order variable, therefore the BEFORE/AFTER locations cannot be specified on the COMPUTE statement. Since the name of the computed report item will appear on the COLUMN, DEFINE, and COMPUTE statements, it is this name that is used to create and coordinate the link between the three statements.

In the following example the patients weights are to be displayed in both pounds and kilograms (the earlier example converts to kilograms without creating a new column). Since the units for the variable WT are in pounds, a new column containing the weight in kilograms will need to be created. This is accomplished in a compute block which creates the computed variable WTKG.

```
* Creating a new column with a compute block;
title1 'Using The COMPUTE Block';
title2 'Adding a Computed Column';
proc report data=rptdata.clinics nowd split='*';
  column lname sex (' Weight *--' wt wtkg); ❶
  define lname   / order   width=18 'Last Name*--';
  define sex     / display width=6  'Gender*--';
  define wt      / display format=6. 'Pounds*--';
  define wtkg ❷  / computed ❸ format=9.2 'Kilograms*--';
  compute wtkg; ❹
    wtkg = wt / 2.2; ❺
  endcomp; ❻
```

```
run;
```

❶ The name (WTKG) for the new column is added to the COLUMN statement.

❷ The DEFINE statement associated with the new column (WTKG) has a define type of COMPUTED ❸.

❹ The COMPUTE statement contains the name of the computed variable.

❺ The new variable (WTKG) is calculated by dividing the weight (WT) in pounds by 2.2.

❻ Compute blocks are terminated with an ENDCOMP statement.

The resulting table shows both the weight in pounds and kilograms.

Using The COMPUTE Block
Adding a Computed Column

		Weight	
Last Name	Gender	Pounds	Kilograms
Adams	F	155	70.45
Adamson	F	158	71.82
Alexander	M	175	79.55
Antler	M	240	109.09
Atwood	M	105	47.73
Banner	M	175	79.55
Baron	M	160	72.73
Batell	F	155	70.45

. . . . portions of the table not shown . . . .

Calculations Based on Statistics

In the previous example explicit names were used for the variables in the compute block. WT had a usage of DISPLAY, and WTKG was computed. In the following example the mean for each region is calculated and it is from this mean that we create a column in Kilograms. Since statistics are being calculated, WT must by necessity have a define usage of ANALYSIS. This means that a compound column name (WT.MEAN ❷) is used in the compute block.

```
* Calculations based on statistics;
title1 'Using The COMPUTE Block';
title2 'Calculations Based on a Statistics Column';
proc report data=rptdata.clinics nowd split='*';
  column region (' Weight *--' wt wtkg);
  define region  / group   width=7
                           'Region*--';
  define wt      / analysis mean
                           format=8.1
                           'Pounds*--'; ❶
  define wtkg    / computed format=9.2
                           'Kilograms*--';
  compute wtkg;
    wtkg = wt.mean / 2.2; ❷
  endcomp;
  run;
```

❶ The mean weight (in pounds) is calculated for WT across the group variable (REGION).

❷ The weight in Kilograms is calculated by converting it from the mean weight in pounds. Notice that the compound name, WT.MEAN reflects the calculated statistic (in pounds).

Using The COMPUTE Block
Calculations Based on a Statistics Column

Region	Weight	
	Pounds	Kilograms
1	195.0	88.64
10	172.3	78.33
2	107.8	49.00
3	145.8	66.27
4	159.1	72.34
5	157.8	71.70
6	198.0	90.00
7	151.0	68.64
8	160.0	72.73
9	187.8	85.36

The following, somewhat silly, example demonstrates more fully the need to be able to use compound names. The compute block is used to calculate the standard error, which is based on the standard deviation (STD) and the square root of N. This is silly because the standard error is also available directly as the statistic STDERR and does not really need to be calculated using a formula.

```
* Calculations based on two statistics columns;
title1 'Using The COMPUTE Block';
title2 'Calculations Based on Two Statistics Columns';
proc report data=rptdata.clinics nowd split='*';
  column region (wt,(mean std stderr n) wtse); ❶
  define region / group    width=7 'Region*--';
  define wt     / analysis 'Pounds*--';
  define wtse   / computed format=9.2 'STD Error*--';
  compute wtse;
    wtse = wt.std / sqrt(wt.n); ❷
  endcomp;
  run;
```

❶ A list of statistics have been requested for the single analysis variable WT. The computed variable, which is to be calculated (WTSE), must also appear on the COLUMN statement.

❷ The calculation involves the use of both the standard deviation of WT (WT.STD) and the number of observations (WT.N). The use of compound names allows this refined specification.

In the resulting table the Standard Error has been generated twice. It is first calculated directly (as a named statistic on the COLUMN statement) and secondly the value is calculated in the compute block. For comparison purposes, both have been shown in the table.

Using The COMPUTE Block
Calculations Based on Two Statistics Columns

Region	Pounds				STD Error
	mean	std	stderr	n	
1	195	0	0	4	0.00
10	172.33333	7.2295689	2.9514591	6	2.95
2	107.8	4.1311822	1.3063945	10	1.31
3	145.8	35.197222	11.130339	10	11.13
4	159.14286	23.719052	6.3391832	14	6.34
5	157.75	44.248487	15.644202	8	15.64
6	198	23.776739	7.5188652	10	7.52
7	151	4.6188022	2.3094011	4	2.31
8	160	2.3094011	1.1547005	4	1.15
9	187.8	26.951809	8.5229103	10	8.52

Calculating Percentages within Groups

Since a percentage calculation is the ratio of the current value to a total for the group, the total must be available for the calculation. As a result, the calculation of percentages requires the use of two compute blocks. The first is used to determine the total across the group (COMPUTE BEFORE ❶), and then this total is used on each detail row of the table ❸ to calculate the percentage in the second compute block.

The following example also makes use of the temporary variable TOTWT. A temporary variable does not appear on either the COLUMN statement or on the report itself. The values of temporary variables are automatically retained in memory as the rows of the computed summary information are processed, and are therefore very useful for holding onto intermediate values. Here the variable TOTWT is used to hold the total value of WT.SUM for each group ❷.

Notice in this example that the compound variable name WT.SUM has two different meanings depending on which compute block is being executed. When it is used in a summary across all visits to a clinic (❶ ❺), it will contain the total weight for that clinic. When a detail row is being processed ❸, WT.SUM will contain the value of WT for that row.

```
* Calculating percentages within groups;
title1 'Using the COMPUTE Block';
title2 'Percentages Within Groups';

proc report data=rptdata.clinics nowd split='*';
  where clinnum in('031234', '036321');
  column clinnum lname wt prctwt ;
  define clinnum  / group width=10;
  define lname    / order
                    'Last Name';
  define wt       / analysis format=6.
                    'Weight';
  define prctwt   / computed format=percent8.1
                    'Percent*of Total';

  compute before clinnum; ❶
    totwt = wt.sum; ❷
  endcomp;

  compute prctwt; ❸
```

```
  prctwt = wt.sum / totwt; ❹
endcomp;

break after clinnum  / dol skip summarize suppress; ❺
run;
```

❶ This compute block is executed before the individual rows of the table are processed. This makes the temporary variable (TOTWT) available for use in the second compute block ❸. The keyword BEFORE is available for this compute block because CLINNUM is a GROUP variable.

❷ The total weight is calculated as the total weight within the specific clinic number. Since CLINNUM is a GROUP variable, WT.SUM will hold the total weight. During the execution of this compute block WT.SUM is the total for the upcoming CLINNUM, and this value is what is saved in the temporary variable TOTWT.

❸ This compute block is used to calculate the percentages for each row of the table.

❹ In this equation we use the total weight for all the patients that have visited the clinic, TOTWT. This total is used with the individual patient's weight (WT.SUM). Eventually the value is displayed using the percent8.1 format.

❺ The BREAK statement requests a summary line after each clinic. This statement causes an additional summary row to be added to the final report and the compute block for PRCTWT will also be executed for the corresponding row in the computed summary information.

Using the COMPUTE Block
Percentages Within Groups

clinic number	Last Name	Weight	Percent of Total
031234	Candle	195	27.3%
	Henry	162	22.7%
	Panda	195	27.3%
	Smith	162	22.7%
		714	100.0% ❺
036321	Herbal	155	29.8%
	Jones	105	20.2%
	Masters	155	29.8%
	Stubs	105	20.2%
		520	100.0%

The percentage across the clinic is of course 100%. This value is calculated automatically when the row resulting from the BREAK AFTER CLINNUM statement ❺ is processed.

On this row the temporary variable TOTWT and WT.SUM will contain the same value.

USING _PAGE_ WITH BEFORE AND AFTER

In an earlier example the compute block was used with the LINE statement to generate a footnote using a COMPUTE AFTER statement.

```
compute after;
  line @20 'Weight taken during';
  line @20 'the entrance exam.';
endcomp;
```

The two lines of text generated by these two LINE statements will appear at the end of the report, and unlike a footnote generated by a FOOTNOTE statement the text generated by these two LINE statements will not appear at the bottom of each page if multiple pages are required. This is because the compute block is targeted to only execute at the end of the report. Its '*target*' is implied, essentially the `compute after` statement is interpreted as `compute after report`.

For the LISTING destination we can use the _PAGE_ option to give us additional control by explicitly stating _PAGE_ as

the target. The compute block target determines when the compute block executes, and when _PAGE_ is used as a target, it forces the compute block's execution at page boundaries. In a sense _PAGE_ is used as if it was a grouping variable, however the text will instead be written AFTER the page breaks.

```
compute after _page_;
  line @10 'Weight taken during';
  line @10 'the entrance exam.';
endcomp;
```

The same technique can be used to place text at the top of the page (between the titles and the body of the report) by using BEFORE.

```
compute before _page_;
  line @10 'Weight taken during';
  line @10 'the entrance exam.';
endcomp;
```

Of course the concept of a page is not the same for all ODS destinations, and this can be an issue for some destinations, such as HTML and PDF. For other destinations page break determination is not always directly controlled by SAS. For these destinations, which include the RTF destination, _PAGE_ does not work or at best it probably will not work as you might anticipate.

USING THE OUT= OPTION TO VIEW REPORT BREAK INFORMATION

Sometimes when using compute blocks, it is helpful to be able to see the how the break information is being used in the construction of the report rows. This break information can sometimes be visualized by creating an output data set using the OUT= option ❶ on the PROC statement.

After the REPORT step's computed summary information is processed during the Report Row Phase, the data set that is specified using the OUT= option is saved as the phase is completed. When this data set is written for us by PROC REPORT, all the compute block processing has therefore been performed. Although there is really no way to trap or trace the compute block processing, an examination of the output data set created by PROC REPORT can still be informative.

As a bonus the output data set shows all of the report rows and report items (including those that will not be written to the final report). In addition to the other report items, the output data set includes the automatic temporary variable _BREAK_. This variable notes observations in the output data set that are a result of data summarizations, such as those generated by BREAK and RBREAK statements.

The following example repeats the REPORT step used earlier to demonstrate the calculation of percentages, but it adds an OUT= option to the PROC REPORT statement.

```
* Calculating percentages within groups;
title1 'Using the COMPUTE Block';
title2 'Examining the Output Data Set';

proc report data=rptdata.clinics
            out=r5_5out ❶
            nowd split='*';
  where clinnum in('031234', '036321');
  column clinnum lname wt prctwt ;
  define clinnum  / group width=10;
  define lname    / order
                    'Last Name';
  define wt       / analysis format=6.
                    'Weight';
  define prctwt   / computed format=percent8.1
                    'Percent*of Total';

  compute before clinnum; ❷
    totwt = wt.sum;
  endcomp;
```

```
  compute prctwt;
    prctwt = wt.sum / totwt;
  endcomp;

  break after clinnum  / dol skip summarize suppress; ❸
  run;

proc print data=r5_5out;
   title3 Summarized Data Set;
   run;
```

The PROC PRINT of the data set generated by the OUT= option in this example results in:

```
Using the COMPUTE Block
Examining the Output Data Set
Summarized Data Set

Obs    clinnum     lname      wt     prctwt     _BREAK_

  1    031234                714      .         clinnum ❷
  2    031234     Candle     195    0.27311
  3    031234     Henry      162    0.22689
  4    031234     Panda      195    0.27311
  5    031234     Smith      162    0.22689
  6    031234                714    1.00000    clinnum ❸
  7    036321                520    0.72829    clinnum ❷
  8    036321     Herbal     155    0.29808
  9    036321     Jones      105    0.20192
 10    036321     Masters    155    0.29808
 11    036321     Stubs      105    0.20192
 12    036321                520    1.00000    clinnum ❸
```

❷ This compute block generates a summary line BEFORE each distinct value of CLINNUM. This row is only a result of the compute block and does not appear in the final report. Because wt.sum was referenced in a COMPUTE block, we can see how the value of WT.SUM was captured. Because there is no BREAK statement associated with the COMPUTE BEFORE, this row does not appear in the final report. However we can see from the output data set, that WT.SUM was available BEFORE each CLINNUM group in order to be assigned to the temporary variable TOTWT.

❸ AFTER each group of values for CLINNUM, the BREAK statement generates a summary or break row in the final report, and this summary row is also reflected in the OUT= data set..

Notice the inclusion of the automatic column _BREAK_. This column can be used to track break events as the table is calculated. The values of _BREAK_ can be utilized with IF-THEN/ELSE processing in the compute block itself.

SUMMARY

The compute block is unique to the REPORT procedure, and although it adds a great deal of power and flexibility, it also can add a level of coding complexity that can confound the new user. Fortunately it is not hard to get started using the compute block. Much of the programmer's DATA step syntax knowledge can be put to good use in the compute block.

Most SAS language elements can be used in the compute block. In addition the LINE statement can be used to write blank lines, computed and data values, and text to the report.

When using report items in the compute block, the programmer must be aware of the item's usage. A report item's usage will determine how the item is named or addressed within the compute block.

ABOUT THE AUTHOR

Art Carpenter's publications list includes four books, and numerous papers and posters presented at SAS Global Forum, SUGI and other user group conferences. Art has been using SAS® since 1976 and has served in various leadership positions in local, regional, national, and international user groups. He is a SAS Certified Advanced Programmer™ and through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

AUTHOR CONTACT

Arthur L. Carpenter
California Occidental Consultants
P.O. Box 430
Vista, CA 92085-0430

(760) 945-0613
art@caloxy.com
www.caloxy.com

REFERENCES

Some of the tables, text, and examples in this paper have been borrowed with the author's permission from the SAS Press book *Carpenter's Complete Guide to the SAS® REPORT Procedure*.

Carpenter, Arthur L., 2007, *Carpenter's Complete Guide to the SAS® REPORT Procedure*, SAS Institute Inc., Cary NC.

Carpenter, Arthur L., 2006a, "In The Compute Block: Issues Associated with Using and Naming Variables", published in the proceedings of the 14th Annual Western Users of SAS® Software, Inc. Users Group Conference (WUSS), Cary, NC: SAS Institute Inc., paper DPR_Carpenter.

Carpenter, Arthur L., 2006b, "Advanced PROC REPORT: Traffic Lighting - Controlling Cell Attributes With Your Data", published in the proceedings of the 14th Annual Western Users of SAS® Software, Inc. Users Group Conference (WUSS), Cary, NC: SAS Institute Inc., paper TUT_Carpenter.

TRADEMARK INFORMATION