Advanced PROC REPORT: Doing More in the Compute Block

Arthur L. Carpenter
California Occidental Consultants

ABSTRACT

One of the unique features of the REPORT procedure is the Compute Block. This PROC step tool allows the use of most DATA step statements, logic, and functions, and through the use of the compute block you can modify existing columns, create new columns, write text, and more! This provides the SAS programmer a level of control and flexibility that is unavailable in virtually all other procedures. Along with this flexibility comes complexity and this complexity often thwarts us as we try to write increasingly interesting compute blocks.

The complexity of the compute block includes a number of column identification and timing issues that can confound the PROC REPORT user. Of course to make matters even more interesting, there can be multiple compute blocks that can interact with each other and these can execute for different portions of the report table.

This tutorial will discuss the essential elements of the compute block, its relationship to the processing phases, and how it interacts with temporary variables and other compute blocks. We will discuss timing issues and naming conventions through a series of examples.

KEYWORDS

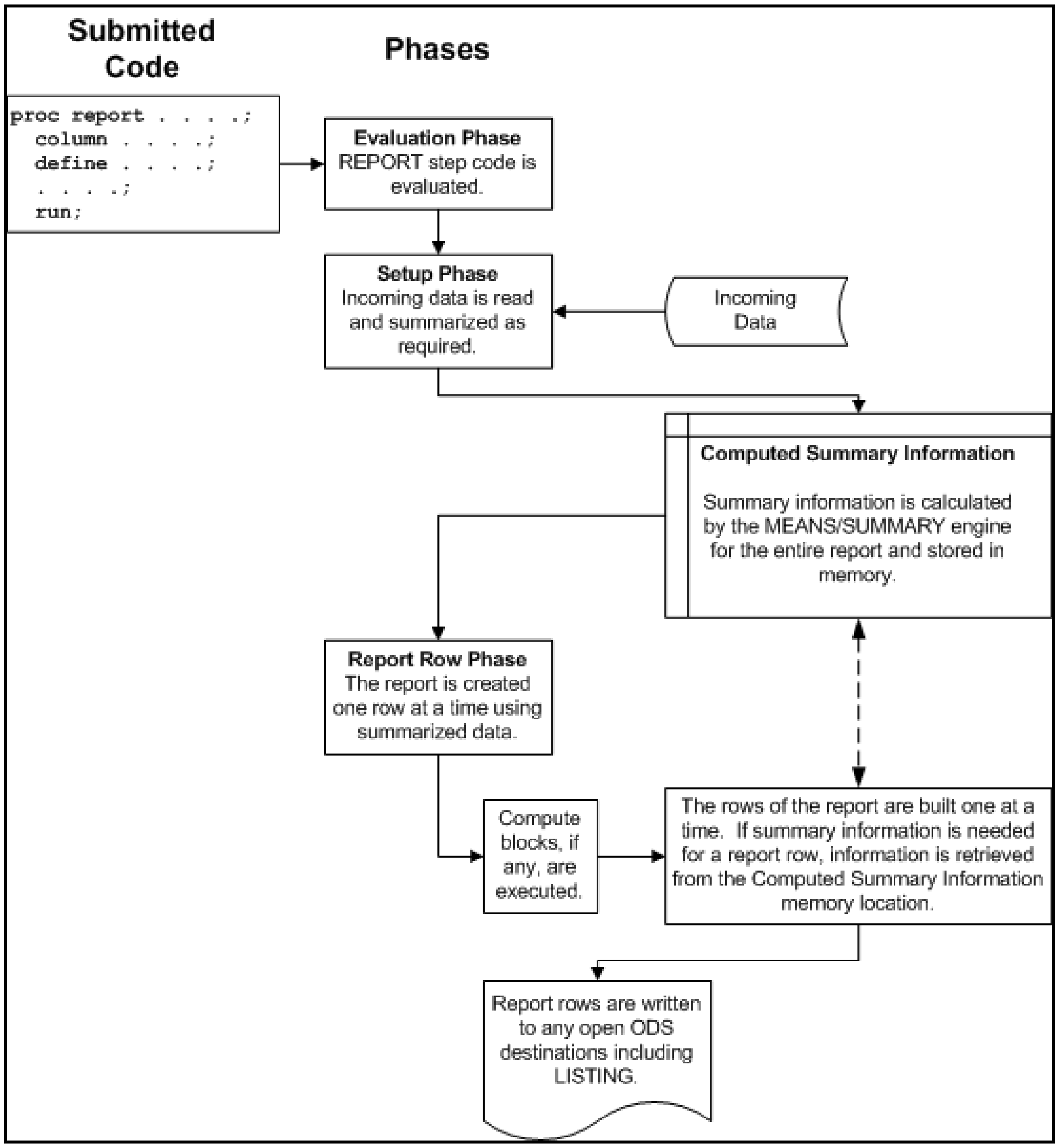
PROC REPORT, Compute Block, Report Row Phase, Temporary variables

REPORT PHASES REVIEW

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.


Submitted Code
Phases
proc report . . . .;
column . . . .;
define . . . .;
. . . .;
run;
Evaluation Phase
REPORT step code is evaluated.
Setup Phase
Incoming data is read and summarized as required.
Incoming Data
Computed Summary Information
Summary information is calculated by the MEANS/SUMMARY engine for the entire report and stored in memory.
Report Row Phase
The report is created one row at a time using summarized data.
Compute blocks, if any, are executed.
The rows of the report are built one at a time. If summary information is needed for a report row, information is retrieved from the Computed Summary Information memory location.
Report rows are written to any open ODS destinations including LISTING.

REPORT ITEM NAMING RULES

When coding within the compute block, report items will need to be addressed. How they are addressed or named 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
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 addressed directly and is available during the execution of the compute block.

Using a Compound Name
Compound variable names are needed, when a variable with an ANALYSIS define type has been used to calculate a statistic. 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
```

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 column number as an indirect column reference. This pseudo variable name is always of the form

```
_Cxx_
```

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

More specific information and examples of the naming of report items in the compute block can be found in Carpenter (2006a).

USING BEFORE AND AFTER

The BEFORE and AFTER timing location options can be used on the compute statement to determine when the compute block is to be executed. They can be relative to both grouping variables and to the overall report.

In the following example we want to calculate the percentage of patients falling within selected weight groups both by and across gender. We want to generate the following report using the data from a small clinical study.

```
Extending Compute Blocks
Using BEFORE and AFTER

                             Percent   Percent
  Gender         Weight   N    by Sex     Total
  Female   < 100          2       6%        3%
          100-< 200      29      91%       36%
          200-< 300       1       3%        1%
                         ==  ========  ========
                         32     100%       40%

  Male    100-< 200      37      77%       46%
          200-< 300      11      23%       14%
                         ==  ========  ========
                         48     100%       60%

                         ==  ========  ========
                         80       .       100%
```

Percentage calculations are always a bit more interesting since they require a current value and a group total.

This example has two types of percentage calculations and therefore requires two types of totals.

In the clinics data the patient weight is to the nearest tenth of a pound. We would like to consolidate the patients into 100 pound groups and we will use the format PNDS. to define the groups of interest.

```
proc format;
   value pnds
         low-<100 = ' < 100'
         100-<200 = '100-< 200'
         200-<300 = '200-< 300'
         300-high = '300 and over';
      value $gender
        'M'='Male'
        'F'='Female';
   run;

title1 'Extending Compute Blocks';
title2 'Using BEFORE and AFTER';
```

In each of the compute blocks in the following REPORT step notice the usage of the variable WTN. It has a very different meaning in each of the compute blocks. The REPORT step is:

```
proc report data=rptdata.clinics nowd
            out=outpct split='*';
   column sex wt wt=wtn percnt tpercnt;
   define sex     / group format=$gender6. 'Gender' ;
   define wt      / group format=pnds.     'Weight' order=formatted;
   define wtn     / analysis format=2. n   'N'; ❶
   define percnt  / computed format=percent8. ❷
                        'Percent*by Sex';
   define tpercnt / computed format=percent8. ❸
                        'Percent*Total';
   compute before; ❹
      * Total number of patients;
      totcount = wtn;
   endcomp;

   compute before sex; ❺
      * Total number within gender group;
      count = wtn;
```

```
endcomp;

compute percnt; ❻
   * percent within weight group;
   percnt= wtn/count;
endcomp;

compute tpercnt; ❼
   * Total percent within weight group;
   tpercnt= wtn/totcount;
endcomp;

* Percent count for summary after SEX;
compute after sex; ❽
   percnt= wtn/count;
   tpercnt= wtn/totcount;
endcomp;

break after sex / suppress summarize dol skip;

* Percent count for summary after the report
* (across SEX);
compute after; ❾
   percnt= .;
   tpercnt= wtn/totcount;
endcomp;

rbreak after / summarize dol skip;
run;
```

❶ The alias WTN will be used to collect the number of patients with non-missing values of WT (which is being used as a grouping variable).

❷ PERCNT is a computed variable that will hold the percentage of patients within each gender and weight group.

❸The report item TPERCNT is a computed variable that will hold the percent of patients across genders.

❹ During the execution of this compute block, WTN contains the total number of non-missing values of WT (assuming that there are no missing values for WT, this count is also the total number of patients). In order to retain this total, it is saved in the temporary variable TOTCOUNT. This will be the denominator for all percentage calculations based on the overall patient count. This compute block will only be executed once for the entire report.

❺ During the execution of this compute block, WTN contains the number of non-missing values of WT within a gender. BEFORE the value of SEX changes, the number of patients with a weight is saved in the temporary variable COUNT. Since the value of a temporary variable is automatically retained, this value of COUNT will be available for all the detail lines associated with this value of SEX. COUNT becomes the denominator for all percentage calculations within a gender. This compute block will execute only once for each value of SEX.

❻ The value of PERCNT is calculated for the detail lines of the report. During the execution of this compute block, WTN contains the number of non-missing values of WT within a specific weight group. The assignment statement in this compute block calculates a computed variable (PERCNT) using the temporary variable COUNT and a report item variable (WTN). This block is also executed for summary rows.

❼ The value of TPERCNT is calculated during the execution of this compute block for the detail lines of the report. Although PERCNT and TPERCNT are both calculated for each detail line both the PERCNT and the TPERCNT compute blocks must appear. This is because for detail lines, the report item's computed value is only transferred to the report row from its own compute block. During the execution of this compute block, WTN contains the number of non-missing values of WT within a specific weight group. This block is also executed for summary rows.

❽ This compute block and the BREAK statement both have the location of AFTER SEX, consequently here the values for

PERCNT and TPERCNT are calculated only for the group summaries for each gender. During the execution of this compute block, WTN contains the number of non-missing values of WT within this value of gender. Because of the presence of the compute blocks at ❻ and ❼ this compute block is redundant in this step.

❾ Since no variable is on this COMPUTE statement, it will be executed after the entire report along with the RBREAK statement. It is at this time that the values for PERCNT and TPERCNT are calculated for the final summary line of the report. During the execution of this compute block, WTN contains the number of non-missing values of WT across the entire table. Of course WTN and TOTCOUNT will have the same value at this point. This compute block will be executed only once, at the end of the report.

Sometimes it can help to understand the relationship of the compute blocks and the report item variables by looking at the data set created using the OUT= option.

```
Extending Compute Blocks
Using BEFORE and AFTER
Final Report Rows

Obs     sex      wt     wtn      percnt     tpercnt     _BREAK_

  1               .      80      .             .        _RBREAK_ ❹
  2      F        .      32      .          0.4000      sex      ❺
  3      F       98       2     0.06250     0.0250               ❻ ❼
  4      F      105      29     0.90625     0.3625               ❻ ❼
  5      F      201       1     0.03125     0.0125               ❻ ❼
  6      F        .      32     1.00000     0.4000      sex      ❽
  7      M        .      48     1.50000     0.6000      sex      ❺
  8      M      105      37     0.77083     0.4625               ❻ ❼
  9      M      201      11     0.22917     0.1375               ❻ ❼
 10      M        .      48     1.00000     0.6000      sex      ❽
 11               .      80      .          1.0000      _RBREAK_ ❾
```

CREATING COMPUTED CHARACTER VARIABLES

When creating a computed character variable its attributes must be declared. To do this the COMPUTE statement supports the CHARACTER and LENGTH options. The CHARACTER option can be abbreviated as CHAR and the following example creates a computed variable that contains the concatenated first and last names of the patient ❶.

```
title1 'Extending Compute Blocks';
title2 'Defining Character Columns';

proc report data=rptdata.clinics(where=(region='2'))
            nowd split='*';
   column lname fname name wt ht edu;
   define lname / order noprint;
   define fname / order noprint;
   define name  / computed 'Patient Name';
   define wt    / display  'Weight';
   define ht    / display  'Height';
   define edu   / display  'Years*Ed.';

   compute name / character length=17; ❶
      name = trim(fname) || ' ' ||lname;
   endcomp;
   run;
```

```
Extending Compute Blocks
Defining Character Columns

                                                 Years
  Patient Name            Weight    Height         Ed.
  Teddy Atwood               105        64          14
  Linda Haddock              105        64          14
  Samuel Harbor              105        64          14
                             105        64          14 ❷
  Marcia Ingram              115        64          14
  Zac Leader                 105        64          14
  Sandra Little              109        63          12
  Margot Long                115        64          14
  Linda Maxwell              105        64          14
  Liz Saunders               109        63          12
```

The attributes of the computed variable NAME have been specified through the use of options on the compute statement.

❶ The computed variable NAME is declared to be CHARACTER with a length of 17. The default variable type is numeric and the default length for character variables is 9.

❷ Samuel Harbor has two visits, and since last and first names are order variables, the compute block is only executed for the first occurrence, consequently the computed variable NAME is missing for the second occurrence.

When you specify a character format using the FORMAT= option on the DEFINE statement, you **must** also use one or both of these two COMPUTE statement options. If both a format and a LENGTH= option are present, the length will be taken from the LENGTH option.

CHANGING GROUPING VARIABLE VALUES ON SUMMARY LINES

The Problem

In the following example regions have been used to form groups based on the user defined format $REGNAME ❶. In addition a request for an overall summary line has been made through the use of the RBREAK statement ❷. Remember that SUPPRESS is not used with the RBREAK statement, as there is no value of the grouping variable to suppress ❸.

```
proc format;
   value $regname ❶
     '1','2','3' = 'No. East'
     '4'         = 'So. East'
     '5' - '8'   = 'Mid West'
     '9', '10'   = 'Western';
   run;

title1 'Extending Compute Blocks';
title2 'RBREAK Does not Create Summary Text';

proc report data=rptdata.clinics nowd split='*';
  column region edu ht wt;

  define region  / group width=10 'Region' ❶
                   format=$regname. order=formatted;
  define edu     / analysis mean  'Years of*Education'
                   format=9.2 ;
  define ht      / analysis mean format=6.2 'Height';
  define wt      / analysis mean format=6.2 'Weight';

  rbreak after   / summarize dol; ❷
  run;
```

The resulting output table shows that for the summary line, the REGION is blank. ❸

```
Extending Compute Blocks
RBREAK Does not Create Summary Text

                Years of
  Region        Education  Height  Weight
  Mid West          14.31   66.85  172.54
  No. East          13.25   67.33  138.17
  So. East          15.00   69.00  159.14
  Western           12.63   67.25  182.00
                =========  ======  ======
    ❸               13.78   67.45  161.78
```

By default there is no text ❸ under the grouping variable for the RBREAK summary line.

The examples that follow show some techniques that can be used to supply text for this and other summary rows.

Specifying Text in a Compute Block

This is simplest method for adding text to the summary line, however it does have limitations. Add the following compute block to change the value of REGION from MISSING.

```
* Text for the report summary line;
compute after;
   region = 'Combined';
endcomp;
```

After adding the compute block, the following report is generated.

```
Extending Compute Blocks
Using COMPUTE to Supply Summary Text

                Years of
  Region        Education  Height  Weight
  Mid West          14.31   66.85  172.54
  No. East          13.25   67.33  138.17
  So. East          15.00   69.00  159.14
  Western           12.63   67.25  182.00
  ==========    =========  ======  ======
  Co                13.78   67.45  161.78
```

Because REGION is a character variable of length 2, the text 'Combined' has become truncated to 'Co'.

A problem can also occur if you try to assign a value of the wrong type to the grouping variable in the compute block . You cannot assign text to a numeric variable or numbers to a character variable.

Using a Formatted Value

By including an 'other' line in the format definition, we can accommodate the summary line which otherwise has a missing value ❶. In order to make this work we must change the value of REGION from missing ❷ on the summary line.

```
proc format;
   value $regname
     '1','2','3' = 'No. East'
     '4'         = 'So. East'
     '5' - '8'   = 'Mid West'
```

```
       '9', '10'    = 'Western'
       other        = 'Combined'; ❶
     run;

  title1 'Extending Compute Blocks';
  title2 'Using a Format to Rename Summary Text';

  proc report data=rptdata.clinics nowd split='*';
    column region edu ht wt;
    define region  / group width=10 'Region'
                     format=$regname. order=formatted;
    define edu     / analysis mean 'Years of*Education'
                     format=9.2 ;
    define ht      / analysis mean format=6.2 'Height';
    define wt      / analysis mean format=6.2 'Weight';
    rbreak after   / summarize dol;
    * Text for the report summary line;
    compute after;
       region = 'x'; ❷
    endcomp;
    run;
```

❶ Values other than missing and the 10 region numbers will be displayed as ‘Combined’.

❷ REGION is assigned a value *other than* missing or one of the acceptable region numbers.

```
Extending Compute Blocks
Using a Format to Rename Summary Text

               Years of
  Region      Education  Height  Weight
  Mid West        14.31   66.85  172.54
  No. East        13.25   67.33  138.17
  So. East        15.00   69.00  159.14
  Western         12.63   67.25  182.00
  ==========  =========  ======  ======
  Combined        13.78   67.45  161.78
```

‘Combined’ is not truncated here, because it is actually a formatted value.

In the DATA step a missing value will be picked up by the OTHER in the format definition. That will not be the case in the REPORT step when you are formatting a value on a summary line. This means that you will have to assign a non-missing value to REGION to make this technique work.

Creating a Dummy Column

Another approach to adding the text is to build a computed column specifically designed to hold the text of interest. In this example a new column, REGNAME, has been added to the COLUMN statement ❶.

```
  proc report data=rptdata.clinics nowd split='*';
    column region regname edu ht wt; ❶

    define region  / group noprint format=$regname.; ❷
    define regname / computed 'Region'; ❸
    define edu     / analysis mean 'Years of*Education'
                     format=9.2 ;
    define ht      / analysis mean format=6.2 'Height';
    define wt      / analysis mean format=6.2 'Weight';

    rbreak after   / summarize dol suppress;
```

```
* Determine the region name;
compute regname/char length=8;
   if _break_='_RBREAK_' then regname = 'Combined'; ❹
   else regname = put(region,$regname.); ❺
endcomp;
run;
```

❶ Both REGION and REGNAME appear on the COLUMN statement.

❷ The column REGION will be used for grouping but will not be printed.

❸ The computed column REGNAME is created to hold the actual text (row headers) to be displayed.

❹ This compute block will be executed once for each report row, including the summary row.

❺ Use the PUT function to assign the text to REGNAME using the $REGNAME. format.

```
Extending Compute Blocks
Using COMPUTE to Create a Text Column

             Years of
  Region     Education  Height  Weight
  Mid West       14.31   66.85  172.54
  No. East       13.25   67.33  138.17
  So. East       15.00   69.00  159.14
  Western        12.63   67.25  182.00
  ========   =========  ======  ======
  Combined       13.78   67.45  161.78
```

Although not apparent from an inspection of the table, the column labeled REGION is actually a computed character variable.

USING LOGIC AND SAS LANGUAGE ELEMENTS

Within the compute block we can use most of the power of the DATA step. This includes the use of functions, assignment statements, SUM statements, and other executable statements. Some of the more commonly used statements include:

- ARRAY
- CALL routines and functions
- comments (all types are recognized)
- DO block, iterative DO, DO WHILE, DO UNTIL
  (as in the DATA step, all DO loops and DO blocks expect the END statement)
- IF-THEN/ELSE and SELECT
- %INCLUDE
- SUM and assignment statements

Within these statements you can use SAS Language functions in the same manner as you would in the DATA step. Most DATA step functions are available for use in the compute block. Functions that are not available include those (like LAG) that operate against the Program Data Vector, PDV.

The macro language can be used in the REPORT step and the compute block. The use of macro variables and macro calls is essentially the same in the compute block as it is in other locations within SAS.

Using the SUM Statement with Temporary Variables

The SUM statement is as handy in the compute block as it is in the DATA step. In the following example we would like to have a consecutive counter for items within a group.

```
proc report data=rptdata.clinics
                  (where=(region in('1' '2' '3' '4')))
            nowd split='*';
   column region cnt clinnum ('   Patient' wt=wtn wt);
   define region / group format=$regname.;
   define cnt     / computed ' '; ❶
   define clinnum/ group 'Clinic*Number';
   define wtn     / analysis n 'N';
   define wt      / analysis mean
                   format=6. 'Mean*Weight';

   compute before region;
      clincount=0; ❷
   endcomp;
   compute before clinnum;
      clincount+1; ❸
   endcomp;
   compute cnt;
      cnt=clincount; ❹
   endcomp;

   break after region/ suppress skip;
   run;
```

❶ The computed variable CNT will be used to display the item numbers.

❷ CLINCOUNT is initialized to 0 BEFORE each new REGION.

❸ The SUM statement is used to accumulate the count of the clinics in CLINCOUNT.

❹ The cumulative count in CLINCOUNT is written to the report item variable CNT.

```
Extending Compute Blocks
Using the SUM Statement
                                    Patient
                       Clinic               Mean
  region               Number          N  Weight
  No. East          1  011234          2     195
                    2  014321          2     195
                    3  023910          4     105
                    4  024477          4     107
                    5  026789          2     115
                    6  031234          4     179
                    7  036321          4     130
                    8  038362          2     112

  So. East          1  033476          4     156
                    2  043320          4     178
```

A counter has been added for each detail row (within region) of the report.

Remember that CLINCOUNT is a temporary variable and while it can be used to accumulate the count, it cannot appear on the report. On the other hand the computed variable CNT will appear on the report, however values of computed variables are not retained from one row to the next and cannot be used with the SUM statement. This means that we need to use to variables in the REPORT step where only one would be needed in the DATA step.

Repeating GROUP and ORDER Values on Each Row

One of the characteristics of GROUP and ORDER variables is that only the first item of a series of repeated values is shown. Usually this is a preferred behavior, however you may want to show the group value on each row. In the following example the GROUP variable's value is repeated on every line.

```
proc report data=rptdata.clinics
                  (where=(region in('1' '2' '3' '4')))
            nowd split='*';
   column region regname clinnum
         ('Patient Weight' wt=wtn wt);
   define region / group format=$regname. noprint; ❶
   define regname/ computed  'Region'; ❷
   define clinnum/ group     'Clinic*Number';
   define wtn    / analysis n
                  format=5. 'N';
   define wt     / analysis mean
                  format=4. 'Mean';

   compute before region;
      * Load the formatted region into a temporary variable;
      rname = put(region,$regname.); ❸
   endcomp;
   compute regname / character length=12;
      * Move region from temporary variable to a report item;
      regname = rname; ❹
   endcomp;
   break after region/ suppress skip;
   run;
```

❶ REGION is used to form the groups, but it is not printed.

❷ The value of the repeated grouping variable will actually be displayed through REGNAME.

❸ The current formatted value of the grouping variable REGION is stored in RNAME. Since the value of the grouping variable is only available on the first row of the group, the value must be assigned to RNAME once (BEFORE REGION) rather than for each report row.

❹ The value of the temporary variable RNAME is assigned to REGNAME.

```
Extending Compute Blocks
Repeating a Group Name

                  Clinic  Patient Weight
  Region          Number        N   Mean
  No. East        011234        2    195
  No. East        014321        2    195
  No. East        023910        4    105
  No. East        024477        4    107
  No. East        026789        2    115
  No. East        031234        4    179
  No. East        036321        4    130
  No. East        038362        2    112

  So. East        033476        4    156
  So. East        043320        4    178
  So. East        046789        2    160
  So. East        049060        4    143
```

DOING MORE WITH THE LINE STATEMENT

The LINE statement is unique to the REPORT procedure step, and although it is roughly analogous to the PUT statement in the DATA step there are some important behavioral differences. These differences become more apparent as we write more complex compute blocks.

Primarily the LINE statement is used to write text to the report, however it can also be used to write computed values.

Creating Group Summaries

In the clinical study that we have been following, the manager of each of the four primary areas (or groups of regions) is responsible for recruiting 8 clinics with an average of 5 patients for each clinic. The following report assesses the status for each area as the study progresses.

```
Extending Compute Blocks
Using LINE for Group Totals

            Clinic  Patient
  region    Number    Count
  Mid West  051345        2
            054367        2
            057312        2
            059372        2
            063742        4
            063901        4
            065742        4
            066789        2
            082287        2
            084890        2
    Total of  10 clinics is  125.0%  of target
    Patient enrollment is   26
    Per clinic this is   52.0%  of target


  No. East  011234        2
            014321        2
```
. . . . . portions of the report are not shown . . . .

The summary following each set of clinics within regional area is written using LINE statements in a COMPUTE AFTER block.

```
proc report data=rptdata.clinics nowd split='*';
   column region clinnum n; ❶
   define region  / group   format=$regname8.;
   define clinnum / group   'Clinic*Number';
   define n       / width=7 'Patient*Count';

   compute before region; ❷
      clincnt = 0;
      patcnt  = 0;
   endcomp;

   compute n; ❸
      if _break_= ' ' then do; ❹
         * Within region patient count;
         patcnt  + n; ❺
         * Clinic count;
         clincnt + 1; ❻
      end;
   endcomp;

   compute after region;
      * Fraction of target (8);
      clinpct = clincnt/8; ❼
      * Fraction of target (5);
      patpct = patcnt/clincnt/5; ❽
      line @5 'Total of ' clincnt 3. ' clinics is ' ❾
               clinpct percent8.1 ' of target';
      line @5 'Patient enrollment is ' patcnt 4.;
      line @5 'Per clinic this is ' patpct percent8.1 ' of target';
      line ' ';
      line ' ';
   endcomp;
   run;
```

❶ The N statistic is added to the COLUMN statement.

❷ Before processing each region, reset the counters to 0.

❸ Set up the compute block that will be used to execute the SUM statement accumulators.

❹ This compute block will be executed for each report row, and for clinic summary rows we need to update the patient and clinic counters.

❺ Each row represents the number of patients within the clinic.

❻ Each row in the final report represents one clinic.

❼ CLINCNT is the total number of clinics within the region.

❽ The fraction of patients per clinic relative to the target value (5) is calculated.

❾ The LINE statement is used to write out the calculated values.

Variables used on the LINE statement are **always** followed by a format. While a format is not required on the PUT statement, it **is** required on the LINE statement.

Adding Repeated Characters

The LINE statement can be used to generate repeated characters. To create a string of 30 dashes is as simple as specifying the LINE statement as:

```
line @2 '------------------------------------------';
```

```
line @2 30*'-';
```

```
str = repeat('-',29);
line @2 str $30.;
```

The following example, which builds on the previous example, adds repeated text after the summary of each region.

```
Extending Compute Blocks
Using LINE for Group Totals

            Clinic  Patient
  region    Number    Count

  ________________________________________________
  Mid West  051345        2
            054367        2
            057312        2
            059372        2
            063742        4
            063901        4
            065742        4
            066789        2
            082287        2
            084890        2
    Total of  10 clinics is  125.0%  of target
    Patient enrollment is   26
    Per clinic this is   52.0%  of target
  ------------------------------------------------

  No. East  011234        2
            014321        2
    . . . . Portions of the table not shown . . . .
```

Understanding LINE Statement Execution

Although the LINE statement initially seems to be very similar to the DATA step PUT statement, there are important differences that can, at the very least, cause consternation on the part of the programmer. The primary difference is in the way that REPORT executes the LINE statements.

The SAS language elements (statements, functions, etc.) are executed in sequence within the compute block in essentially the same way as they are in the DATA step. However the LINE statements are **not** executed in sequence. After **ALL** of the SAS language elements in the compute block have been executed, REPORT then executes all the LINE statements in the order in which they appear in the compute block. This means that you can **NEVER** conditionally execute a LINE statement with an IF-THEN/ELSE statement, nor can you use a LINE statement inside of a DO loop.

It is generally considered a good programming practice to always put all the LINE statements at the end of the compute block as a visual reminder of this behavior.

CAVEAT:
For LINE statements that contain a temporary character variable, the variable is evaluated with the current value of the variable.

ADDRESSING REPORT ITEMS IN THE PRESENCE OF NESTED ACROSS VARIABLES

The ACROSS define type is used when we want values of classification variables to be next to each other horizontally. In this example we want to nest ACROSS variables, and we want to concatenate the values of two variables into one report item. Because of the nested ACROSS variables, we are going to need to use the absolute column numbers, which use the _C*xx*_ column designations.

```
proc format;
   value $regname
     '1','2','3' = 'No. East'
```

```
      '4'         = 'So. East'
      '5' - '8'   = 'Mid West'
      '9', '10'   = 'Western';
   value $gender
      'm', 'M'  = 'Male'
      'f', 'F'  = 'Female';
   value birthgrp
      '01jan1945'd - '31dec1959'd = '   Boomer' ❶
      other                       = 'Non-Boomer';
   run;

title1 'Extending Compute Blocks';
title2 'Combining Values in ACROSS Columns';

proc report data=rptdata.clinics
            out=outreg ❷
            nowd;
   column region n sex,dob, ❸ (wt=wtmean wt wtval);
   define region / group format=$regname. 'Area';
   define n      / format=3. ' N';
   define sex    / across format=$gender. ' ';
   define dob    / across format=birthgrp. ' ';
   define wtmean / analysis mean noprint; ❹
   define wt     / analysis std  noprint;
   define wtval  / computed ' Mean (SD)';

   * Combine the WT values;
   compute wtval / char length=12; ❺
      _c5_  = cats(put(_c3_,5.1),' (', ❻
                   put(_c4_,7.1),')'); ❼
      _c8_  = cats(put(_c6_,5.1),' (',
                   put(_c7_,7.1),')');
      _c11_ = cats(put(_c9_,5.1),' (',
                   put(_c10_,7.1),')');
      _c14_ = cats(put(_c12_,5.1),' (',
                   put(_c13_,7.1),')');
   endcomp;
   run;

proc print data=outreg;
run;
```

❶ Leading spaces have been added to the label to help center the text.

❷ When working with columns during the program development phase, it is sometimes helpful to print out the output data set of the report so that the column numbers can be accurately determined.

❸ The comma has been used to nest the three weight variables within date of birth (DOB), which is in turn nested within SEX. Notice that parentheses have been placed around the three weight variables to form a group that can be nested within DOB.

❹ The statistics for weight (WT and WTMEAN) are calculated, but the NOPRINT prevents their display.

❺ Notice that although we use the compute block for WTVAL, this report item is never actually created. Instead, because it is nested under ACROSS classification variables, we assign the computed values directly into the appropriate columns using the absolute column numbers.

❻ The mean, _C3_, is added to the computed character string that will contain the statistics. We cannot address the mean using the alias WTMEAN because of the nesting across age groups. This solution only works because we KNOW what is contained in each of the columns. Notice that although WTMEAN is not displayed on the final report (because it has been defined with the NOPRINT option ❹), it still occupies columns (_C3 _, _C6 _, _C9 _, and _C12 _) on the output data set.

❼ The standard deviation, _C4_, is added to the computed value. The variable WT.STD would not be available for use, because it is nested under an ACROSS variable.

The output data set WORK.OUTREG, can provide you a "snapshot" of the end result of the processing of the compute blocks during the report row phase, and it can also be used to help us determine the column numbers that are used in the WTVAL compute block. The absolute column numbers themselves are available for our use because they have already been determined during the Setup Phase.

```
Extending Compute Blocks
Combining Values in ACROSS Columns

Obs   region   n     _C3_      _C4_       _C5_         _C6_      _C7_        _C8_

 1      5     24   140.667   36.9504   140.7(37.0)   172.625   34.8668    172.6(34.9)
 2      1     24   119.222   20.6505   119.2(20.7)   112.000    0.0000    112.0(0.0)
 3      4     14   139.000   13.8564   139.0(13.9)   155.000     .        155.0(.)
 4     10     16   177.000    0.0000   177.0(0.0)    163.000    0.0000    163.0(0.0)

Obs    _C9_     _C10_        _C11_        _C12_     _C13_       _C14_       _BREAK_

 1   161.000  20.2287    161.0(20.2)    200.143   33.4187    200.1(33.4)
 2   121.286  27.8132    121.3(27.8)    195.000    0.0000    195.0(0.0)
 3   170.667  27.6381    170.7(27.6)    158.000   18.9209    158.0(18.9)
 4   187.800  37.2451    187.8(37.2)    184.714   13.8770    184.7(13.9)
```

The final REPORT table is:

```
Extending Compute Blocks
Combining Values in ACROSS Columns

                          Female                        Male

                   Boomer     Non-Boomer       Boomer      Non-Boomer
  Area        N   Mean (SD)   Mean (SD)       Mean (SD)    Mean (SD)
  Mid West   24  140.7(37.0)  172.6(34.9)    161.0(20.2)   200.1(33.4)
  No. East   24  119.2(20.7)  112.0(0.0)     121.3(27.8)   195.0(0.0)
  So. East   14  139.0(13.9)  155.0(.)       170.7(27.6)   158.0(18.9)
  Western    16  177.0(0.0)   163.0(0.0)     187.8(37.2)   184.7(13.9)
```

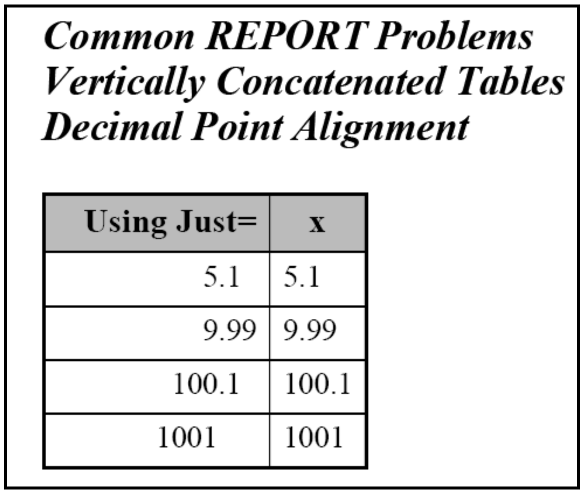
This final report shows how PROC REPORT can provide you some of the same functionality and ability to create cross-tabular reports as PROC TABULATE (albeit with different syntax).

Using JUST=DEC

When the display value contains a decimal point, you can use it to align the numbers directly. The JUST=DEC attribute option can be used to align the decimal points in the values in a column. In the following example the **character** variable X takes on four values (the fourth does not have a decimal point).

```
title1 'Common REPORT Problems';
title2 'Vertically Concatenated Tables';
title3 'Decimal Point Alignment';

proc report data=test nowd;
   column x x=new;
   define x   / display
                style={cellwidth=1in just=dec}
                'Using Just=' ;
   define new / display ;
   run;
```



Common REPORT Problems
Vertically Concatenated Tables
Decimal Point Alignment

Using Just=	x
5.1	5.1
9.99	9.99
100.1	100.1
1001	1001

PDF using the PRINTER style

The numbers in the left column are aligned on the decimal point even though the values are coming from a character variable.

The JUST=DEC style attribute does not work for the HTML destination, but it can be very useful in PDF and RTF.

USING COMPUTE BEFORE/AFTER WITH SUMMARY LINES

The processing steps associated with compute blocks become a bit more interesting when there are both COMPUTE BEFORE or COMPUTE AFTER statements along with BREAK and RBREAK statements. Remember that while the COMPUTE BEFORE / AFTER statements will generate a summary row in the computed summary information, that row will NOT be written to the table unless there is a corresponding BREAK or RBREAK statement with a SUMMARIZE option.

As you read through this example keep in mind that a compute block associated with a summary row *e.g.* COMPUTE BEFORE or COMPUTE AFTER, will ONLY be executed when its corresponding row in the computed summary information is being processed. Compute blocks associated with report items, this includes computed variables, will be executed for **ALL** report rows, **including** summary rows. When two or more compute blocks are to be executed for any given report row, the report item compute blocks are always executed first (left to right). For a summary row any compute blocks associated with that summary row (compute blocks with a BEFORE or AFTER) are only executed after all the report items and their compute blocks have been processed.

The following somewhat contrived example demonstrates both the timing and the relationships between compute blocks and report items. In this report, percentages are calculated for a grouping variable and, although it is a bit silly here, percentages are also calculated across the entire report as well. In order to do this, two compute blocks are used to create two temporary variables which hold the denominators for the two percentage calculations. TOTN ❶ holds the overall number of students while TOTAGE ❷ will hold the total number of students in each AGE group.

```
proc report data=sashelp.class(where=(age in(12,13)))
            out=out11_3_4a nowd;
   column age sex n percent;

   define age     / group;
   define sex     / group 'Gender' format=$6.;
   define n       / 'N' format=2.;
   define percent/ computed format=percent8. 'Percent';

   break after age / summarize suppress skip;
   rbreak after    / summarize;

   compute before;
      totn = n; ❶
   endcomp;
```

```
compute before age;
   totage = n; ❷
endcomp;

compute percent; ❸
   if _BREAK_='_RBREAK_' then percent=n/totn;
   else percent = n/ totage;
endcomp;
run;
```

❶ The overall number of students is stored in the temporary variable TOTN. This compute block is executed only once.

❷ The temporary variable TOTAGE is assigned the value of the number of students in this age group. This compute block is executed at the start of each age group.

❸ This compute block is used to calculate the computed variable PERCENT. Because PERCENT is a report item, this compute block will be executed for EVERY report row.

The incoming data is summarized and stored in the computed summary information during the setup phase. Therefore, at the beginning of the report row phase the summary values are already available to be retrieved from memory when each of the compute blocks execute.

```
Summary Lines with a Percentage

Obs    Age    Sex    n    percent    _BREAK_

  1      .           8       .       _RBREAK_ ❶
  2     12           5       .       Age ❷
  3     12     F     2    0.40000
  4     12     M     3    0.60000
  5     12           5    1.00000    Age
  6     13           3    0.60000    Age ❷
  7     13     F     2    0.66667
  8     13     M     1    0.33333
  9     13           3    1.00000    Age
 10      .           8    1.00000    _RBREAK_
```

During the setup phase the incoming data is read, summarized and stored in the computed summary information area of memory. This information is processed during the report row phase.

By reviewing the output data set, we can see that the presence of the COMPUTE BEFORE block caused the _RBREAK_ information to be captured during the setup phase at the top of the report ❶. During report row phase this compute block is executed, and the previously summarized value of N is retrieved from memory (computed summary information area) and then used to create the temporary variable TOTN. It is at this time that this row is also written to the output data set.

In the same fashion, the COMPUTE BEFORE AGE block ❷ caused the setup phase to summarize age group data into the computed summary information area. During the report row phase this compute block is executed when the second and sixth report rows are processed. It is at this time that N is retrieved from memory and used in the calculation of the temporary variable, TOTAGE.

During the report row phase the COMPUTE PERCENT block ❸ will be executed for each report row, and the value of PERCENT will be computed based on the values of the two temporary variables TOTN and TOTAGE. This of course means that the value for PERCENT will be missing if either of these two temporary variables are missing. When the first report row is processed, two compute blocks will execute (❶ and ❸).

When more that one compute block is executed for a given report row it can sometimes be important to understand their order of execution. For compute blocks associated with report items the compute blocks will always be executed from left to right - in the same order as the variables on the COLUMN statement. It is also important to keep in mind that

COMPUTE BEFORE and COMPUTE AFTER blocks always execute **after** any compute blocks associated with report items. This means that the results of all report item compute blocks will be available for use in COMPUTE BEFORE and COMPUTE AFTER blocks. This also means that these same report item values can be altered and overridden in these compute blocks.

When the first report row (observation 3 in the output data set shown above) is written, the count of 2 for the 12 year old Females was divided by 5 which was the value for the TOTAGE temporary variable. Then the format of PERCENT8. was applied to the result of the calculation and the first report row was completed and sent to all open ODS destinations (in this case the LISTING destination is the only open destination).

The final report as shown in the LISTING destination is:

```
Summary Lines with a Percentage

       Age  Gender    N   Percent
        12  F         2       40%
            M         3       60%
                      5      100%

        13  F         2       67%
            M         1       33%
                      3      100%

                      8      100%
```

Processing and division happens the same way for the next report row (Males' summary information) and their count of 3 is divided by the TOTAGE (5) and the results were formatted with the PERCENT8. format and written to the report row. Finally, the BREAK AFTER AGE statement executed and the total count of 5 was divided by TOTAGE (also 5), and after the format was applied, the calculated value of 100% was written to the report row.

This same processing is repeated for the 13 year old section. Then on the last report row of the report, when it is time to process the report row generated by the RBREAK, the summarized value for TOTN is retrieved from memory and used in the division for PERCENT.

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 coding complexity that can confound the new user. While the compute block supports of number of SAS language elements that are more commonly thought of as a part of the DATA step, programming knowledge and techniques that are useful in the DATA step are not always applicable in the compute block. For other than the simplest of compute blocks, it quickly becomes important for the programmer to understand the differences between the compute block and the DATA step.

It is tempting to want to assign DATA step like sequential processing to the report row phase, however, PROC REPORT holds all the needed summary information in memory, so it is available for both COMPUTE BEFORE and/or COMPUTE AFTER processing during the construction of each report row. The process is different enough from the DATA step that we can confuse what is actually happening if we depend too much on our DATA step processing knowledge. This becomes even more apparent as we add a report item with a define usage of ACROSS.

ABOUT THE AUTHOR

Art Carpenter's publications list includes four books, and numerous papers and posters presented at 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
10606 Ketch Circle
Anchorage, AK 99515

(907) 865-9167
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 draft of the SAS Press book with a working title of *Carpenter's Complete Guide to the SAS REPORT Procedure*.

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