**MWSUG 2024 – Paper BB-018**

# An Animated Guide to Proc Report Internals

Russ Lavery, contractor, Bryn Mawr, PA

## ABSTRACT

PROC REPORT is an exciting “big file technique” that every programmer should know.

PROC REPORT allows the creation of complicated reports, with many levels of summarization, while only reading a large source file one time. PROC REPORT allows, in just one step, summarization to multiple desired levels, calculation of new variables and the appending of different kinds reports into one complex report. ALL this happens in one read of the input data. Especially interesting is that the internal file for PROC REPORT, a file that holds “the combined multiple report” can be sent to a SAS ® data set and used as a input data.

This ‘static” paper attempts to show the time sequence of the internal actions of PROC REPORT. Knowing the time sequence of actions, especially calculations, is crucial to doing complicated PROC REPORTs. This paper supports a highly animated PowerPoint deck that is on the conference web site where the time sequence can be demonstrated as a sequence of events. This paper must use screen prints from the presentation and words, knowing words are less effective than visuals, to explain the time sequence.

I suggest you use this paper, and the slides that are on the conference web site, and present this talk at your company as a lunch-and-learn. You can tell your company you learned this material at MWSUG and ask to be sent to future MWSUGs. I suggest you PLAY the slides (slide show- from current slide and hit the space bar to have the slide advance). The trick is to hit the space bar, wait for the motion to stop and describe what happened. All the code for the paper is in the appendix.

## INTRODUCTION

Figure 1 is just an illustration of what is meant by combining multiple reports into one – and doing that in only one pass through the data. Remember that a data read is an expensive operation.

I think of Figure 1 as combining two different kinds of reports. Inside the red box I “see” something like a

PROC FREQ – a crossing of variable values. This is powerful because PROC REPORT will add the proper number of “crossing” columns depending on the number of zones that happen to be in the data set.

To the right we see a complex report. Three columns on the right, show averages for three different variables but there is no “crossing” of the variables involved (as is in the number of properties.

An interesting trick is that the internal file for this report can be sent to a SAS table and used, as data, in some other way. The fact that PROC REPORT can produce all of this content in one read of the data is why I call PROC REPORT a “big file technique”

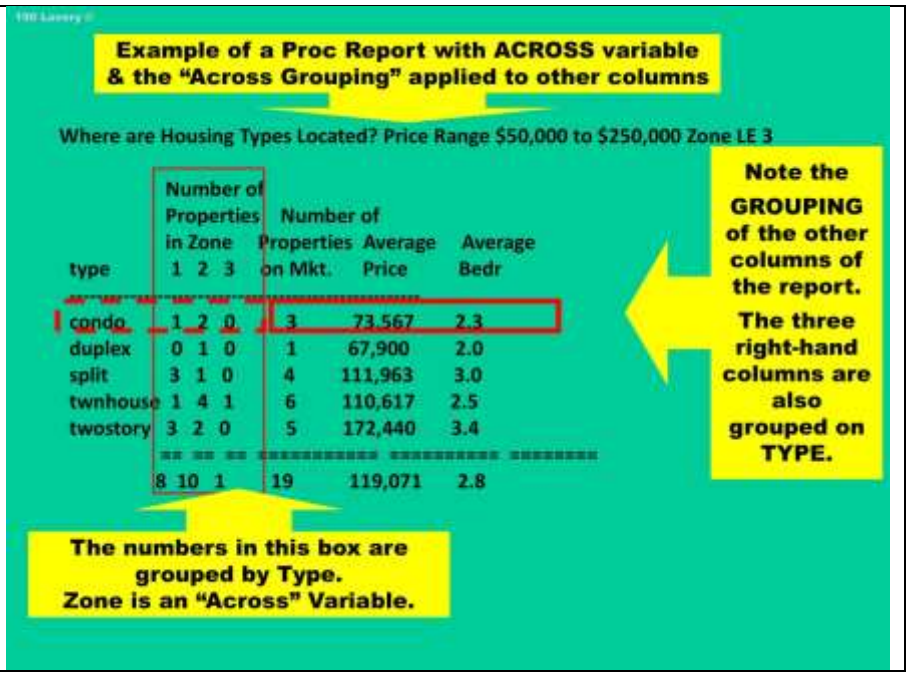


Figure 1

## PROC REPORT INTERNALS

Please look at the important graphic in figure 2.

In the upper upper-left-hand corner we see the data set that used in this example.

In general, the slides wil show the data used in the example, but many examples will have a where clause to limit the data used in the report.

On the left-hand side we see the SAS syntax.


Figure 2

On the right-hand side of figure 2, we see a representation of the computer’s RAM holding the PROC REPORT internal file (Called the Computed Summary Information or CSI). In the bottom right-hand corner, we see the output. Most slides will represent a complete example/report.

## THE THREE MAJOR STEPS INSIDE A PROC REPORT

In figure 2 you can see that part of the RAM holds what looks like a SAS table. The maroon colored table represents an internal file for PROC REPORT and is called the computed summary information or CSI.

The major process of producing a PROC REPORT has three steps:

Step one: The evaluation phase: In this step SAS reads the syntax and creates the ***structure*** for the CSI (as well as creating internal code that PROC REPORT will run). Of crucial important importance is the column statement. Variables in the column statement will be in the CSI.

Step two: The setup phase: In this step SAS reads the data from the source file into the CSI. I think of PROC REPORT is a "big file technique" because it is so very efficient. The source file is only read once. If we were reporting on "sales by state" and had millions of individual sales records, they would be read once into the CSI which would have 50 rows of data.

Since a "state level report" is desired, any calculations required would only be on the 50 rows of data in the CSI and that would be fast. The setup phase really makes PROC REPORT a great tool because, by reading the big file only once, we get the report we want in a short amount of time.

Step three: Report row phase: After reading all the raw data from the source file and summarizing to get to whatever grouping level desired, it is common for a programmer to want to perform a calculation on each, or at least some, of the rows in the CSI.

The CSI is processed from top to bottom, one row at a time, and calculations are performed on each row. After all the calculations have been performed on a row, the row is sent to an output data set to be routed through the ODS and displayed in some format.

## RULES

The major deliverable of this talk will be showing the reader pictures of the internal process of PROC REPORT. This will allow a reader to "see" what the words in the documentation mean. I now want to collect all of the rules in this one place, so that this paper can be used as a reference.

I do not expect just reading the rules will offer much enlightenment. It is suggested that you read this paper in four steps. First, read the rules quickly. Second, read the rest of the paper and see how the rules are applied. Third, come back and read the rules again. Fourth, use the animated PPT deck from the MWSUG website to, explain PROC REPORT to a group. I only expect that the rules will make any sense if you have had a chance to study some of the examples contained in the rest of the paper.

*The Column statement creates variables in the Computed Summary Information. To use a variable, from a data set, you request it via the column statement.

* Variables mentioned in the column statement will appear in the CSI

*If you need a variable in a calculation, but do not want to print it, list the variable in the column statement and define it as "NOPRINT".

*Define statements assign characteristics to variables (group, order, across, sum, NOPRINT).

*Rbreak before/after, or compute before/after will create a "report level" summary line in the CSI.

*Break before/after var, or compute before/after var, create a "variable level" summary line in the CSI.

*Summary lines in CSI **are not printed** *unless you code a break line* with a / summarize option.

*Compute variable blocks execute on every line in the CSI. Please note
1) that they execute in the order in which variables appear in the column statement and
2) that all compute var. blocks execute before the Compute Before/Compute After blocks.

*Compute before Var blocks create lines in the CSI and then execute when SAS is processing that line, in the CSI, that is before/after a new value of the variable.

*Compute before blocks (no variable mentioned) create lines in the CSI and execute when SAS is processing that line, in the CSI, that is before/after all/any other lines in the CSI

* The CSI always contains a _break_ column that is used by PROC REPORT to identify the level of summarization for that row and to "trigger" internal processes. If you do not code a break statement, the columns will be blank.

* Summary lines in CSI are not printed unless there is an associated Break statement with a /summarize option. This can seem a bit confusing but giving programmers control of the process is a good thing and allows the creation of complicated reports.

## EXAMPLE 1: AN ILLUSTRATION OF BASIC PROC REPORT STEPS

Figure 2 shows all the steps "at the same time" and can be confusing. It's difficult to learn a time sequence from a printed page.

Stage 1: In the evaluation stage, the column statement is read and used to create the CSI structure. At this point, the CSI is a SAS table, and has metadata information we might see from a PROC CONTENTS, but with no rows of data.

The CSI will always have an extra column called _break_. This is created by PROC REPORT and is used, by PROC REPORT, to identify rows that are summary rows as opposed to rows containing detailed data. The first row in this CSI is of type _R break_. This tells PROC REPORT that this line, in the CSI, is a report level summary. Any numbers in this row will be describing the whole input data set (or at least the rows from the input data set that made it through the where clause – here we coded where zone LE 2 just to make the output fit on a slide).

PROC REPORT is fast and here is one trick that makes it fast. In this example, as each observation is read into the CSI ***it affects two rows in the CSI.*** Each observation will affect the summary row and also affect the row for its zone. Having observations change multiple totals is very powerful and a great speed trick. It is easy to see that the total number of bathrooms where zone is less than or equal to two is, in fact, 17.

At this stage, because zone is defined as "group", PROC REPORT knows that it will group data by zone.

Price and baths are defined as type "sum". We could ask for other types (Min, Max average and others).

By asking for type "sum" we are going to get the totals of the prices and bathrooms, not only on rows for zone one and zone two – but also for the report level row. We will explain this in more detail later.

In step three, the Report row phase, the CSI is read from top to bottom, one row at a time, and appropriate calculations are performed.

The code has a "compute before" block and SAS knows that a "compute before" statement should only be executed on the row "that is before any other rows in the CSI" and is type _Rbreak_. "Compute before" and "Compute after" statements only execute on rows of the type _R break_ but the individual compute blocks are smart enough to know whether they should execute on the first, or on the last, row in the CSI.

In this example, a “Compute before” statement is used to make the output more understandable. It changes the zone, on the first line, to “ALL” and that makes the report easier to read. You should note that zone was defined as character and was also defined as being wide enough to hold the character string “ALL”. If the zone were numeric, or was character, but not wide enough to hold three letters, we would have a problem.

## EXAMPLE 2: AN ILLUSTRATION OF TIMEING IN THE REPORT ROW PHASE

To make various “cut and pastes” of output fit on these slides I had to use unusual abbreviations for the variable names. STY stands for style of house. REGN stands for the region of the city in which the house is located. DETL is a “made up” variable that has no logical connection to selling houses. It is a detail variable and, by that I mean, it will show up on every row in the CSI.

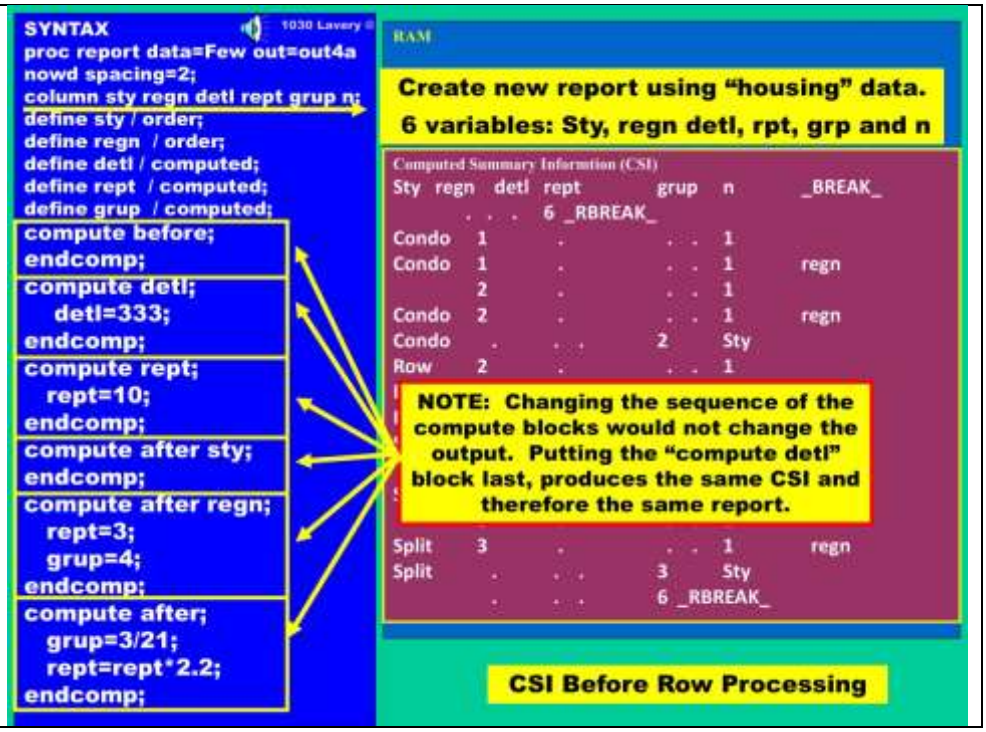


Figure 3

REPT stands for report and this variable has no logical connection to selling houses. It is just a name. GRUP is a variable that has no logical connection to selling houses. The variable name is pronounced like group and I just want to show this variable values to “the group of people attending the seminar”.

Please start by looking at the code on the left-hand side of figure 3. The column statement lists the variables that will be in the CSI. The rightmost variable is n, a PROC REPORT reserved word that causes PROC REPORT to count the number of rows. STY and REGN are “order” variables (sort by variables) and DETL, REPT and GRUP are “computed”. In this example, we will compute most of the values in the CSI. This is how the CSI would look at the end of stage II – the setup stage.

As figure 3 says, changing the sequence of the “code blocks” does not affect the order in which blocks execute. Blocks are aligned to variable names and execute in the order in which the variables appear in the column statement.

I’d like to discuss the different compute blocks that were coded in this example. Combined they created a CSI (that we can examine using the out = option) that will allow us to deduce the timing of the execution of compute blocks.

Compute block one creates a row in the CSI but does not cause any calculations to execute. This might seem a bit weird but is a characteristic of a PROC REPORT that a programmer can exploit to create percents of totals.

Compute block two is a “compute var” block and it computes a variable in the CSI. It will set the value of detail to 333. Important rules concerning “compute var” blocks are 1) “compute var” blocks execute on every line in the CSI and 2) execute before any “compute before” or “compute after” blocks

Compute block three is a “compute var” block and sets the value of REPT to 10.

Compute after style is a “compute after var” block it executes only on the lines in the CSI that occur after STY changes value. This block does not actually request that any calculations be done. It was intended to show that the “compute before var” and’ compute after var” blocks create rows in the CSI.

We should remember that STY was defined as an order variable and that caused all the rows of a particular style of house to be “bunched” together and also to have the groups sorted.

Block five is also a “compute after var” block. It causes REPT to have a value of 3 and GRUP to have a value of 4. These are useful in exploring compute block execution timing. Block five was created to allow a reader to see, on which lines, those kinds of statements execute.

Block six is a “compute after” block. We should remember that, the CSI is processed from the top row to the bottom row and that PROC REPORT performs “appropriate” calculations on each row of the CSI. The statements in block six will only execute on the last row of the CSI – where the value of _break_ is _Rbreak_ and the row is the last row of the CSI.

Figure 4 focuses on the issue of compute blocks and **when** they execute.

If you are going to do complicated calculations in a PROC REPORT you must know the timing of the different compute blocks.

Figure 4 shows the CSI after all the compute blocks have executed and so it can be used to illustrate the rules listed above.

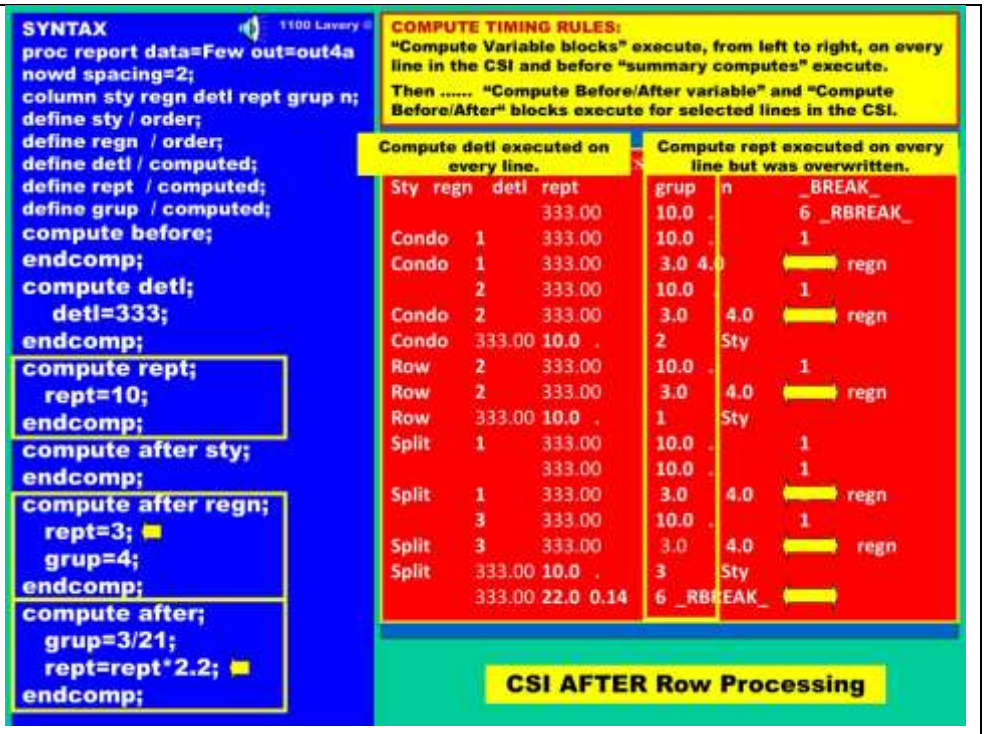

Figure 4

As a start, let’s take a look at the column _break_. It has values of _R break_, regn and sty. These values identify rows that contain summary level information. They identify rows where grouping variables have changed values and, with proper coding, these rows in the CSI can occur before, or after, the variable changes value.

If you look at previous figures you can compare the raw data with the value of the n variable that occurs on summary lines under the variable n. A reader can confirm that n, in these rows, is holding summary

level data, generally, for the rows above. The first row in the CSI was created by the “compute before” block and contains data (n is six) that summarizes rows that follow.

DETL has a value of 333, even on the first and last rows of the CSI, because “compute var” blocks execute on every line of the CSI – and they execute before any other compute block.

REPT has a value of 10 on most rows, but not all. On some rows the value of REPT is three. One must ask if the values of REPT were, at one time, 10 and then were changed to three or if they were always valued at three. Whenever REPT has a value of three GRUP has a value of four and this suggests that the value of three came from the “compute after regn” block.

The “compute after sty” block just adds a line to the CSI. A programmer will often want to create lines in the CSI that she does not print and we will explore this more in later examples.

The last compute block, the “Compute after” block, gives strong evidence as to the timing of computes. It instructs to take the current value of REPT and multiply it by 2.2. It also overwrites the value of GRUP. If you look in the CSI, REPT has a value of 22. The only way this could occur as if REPT had been set to 10 by a “compute var” block and then multiplied by 2.2. This is strong evidence that “compute var” statements execute on every line and execute before other compute blocks execute.

Understanding the timing of execution of compute blocks is critical to programming complicated reports.

## EXAMPLE 3: IF STATEMENTS AND HOW DUMPING THE CSI IS CONFUSING

“If statements” can be useful when creating a report.

Coding if statements is a bit tricky and I use a three step process.

Step 1: Before writing any if statements, I write the program without if statements and “dump” the CSI to a file - which I print. I think it is good to see the CSI because not all rows in the CSI are printed to the SAS listing and I like to see the “missing” rows.

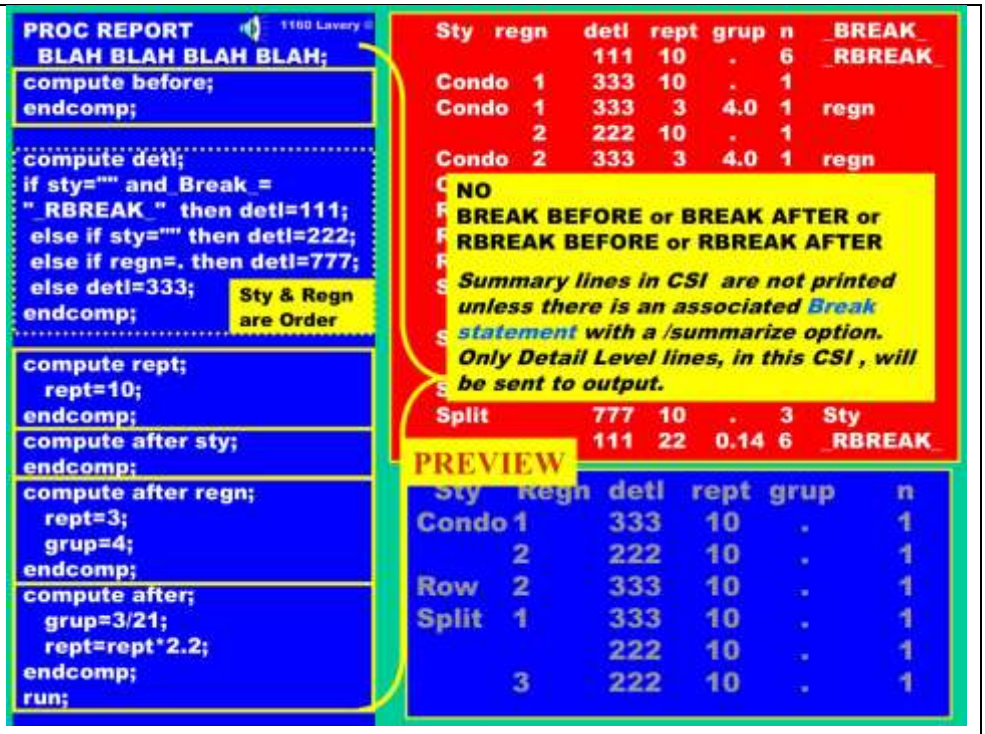


Figure 5

However; there is a problem with “dumping” the CSI. When we use the out = option to “dump” the CSI to a SAS file. **SAS fills in any missing values that had been caused by values repeating over several lines** (please see the STY variable in figure 5). In the CSI, these values are missing and so one cannot

code if statements by simply looking at a “dump” of the CSI. If you look in figure 4, the fourth row does not show the word “condo”. That value is missing in the CSI and in the listing but not in the “dump” of the CSI.

Step 2: I print the listing/output from the PROC REPORT. This does not show all of the rows in the CSI but does show where repeated values have been set to missing.

Step three: After comparing what I’ve seen, in the previous two steps, I’m ready to take a try at coding the if statements. If statements require a little bit of thinking because, if one were to get a data refresh with more rows and more repetitions, the pattern of missing values might change. We will examine a few rows in the CSI in figure 5.

The first row came from a “compute before”.

The second row is for one house that is a condo in region one.

The third row is a summary row, because we have reached the end of condos in region one. Note that _RBreak_ has the value of regn,

The fourth row, with the missing value for STY, is actually a row describing a condo in region two.

“Condo” is suppressed on this linre because it would be a repetition of the word “condo” in the row above. Putting underlines on summary lines makes reports a lot more readable.

The fifth row is a summary for condos in region two note that the column _break_ is valued as regn. This is a REGN level break line.

The sixth row is a summary for all the condos. REGN is missing on this row and that makes sense. This

row summarizes condos for more than one region. It summarizes condos over region 1 and region 2. It makes sense that REGN is blank on this row. Notice that _break_ has the value of STY (style of house) on this row.

I also ask you to look at the output at the bottom of figure 5. Compute statements cause the creation of rows in the CSI *but they do not cause those rows to print*. Only break statements with a /summarize option cause summary rows in the CSI to print. Because of this, the printed output, shown in the bottom right-hand corner of this figure, is much smaller than the CSI. This looks weird but is a bit of brilliant programming logic.

Programmers have good reason to have rows in the CSI that they do not print. These rows allow a programmer to compute running percentages as we will see in the next example.

## EXAMPLE 4: PERCENTAGES WITHIN GROUPS OR “RETAINING” IN PROC REPORT

This example covers a pretty complicated example and will require several slides.

It is so complicated that the code for this example would not fit on one slide. Figure 6 only shows part of the code for this example.

This report does not make a very compelling *business* story. People might say that managers would not want this report, and might be right.


Figure 6

However; while a bit nonsensical, this report does fit on a slide and demonstrates several important concepts in PROC REPORT.

We want to use PROC REPORT to compute the total sales inside a zone. The report (lower right corner) looks useful.

We also want to compute, for each style of house in a zone, two different percentages of sales. We wish to see, for each row, the row's percent of the total dollars in the data set and the percent of the total dollars in the zone.

To calculate these percentages, we need the denominators for the calculations to be stored somewhere in RAM. The denominators will be stored in ram, but separate from the CSI, in what are called "temporary variables". Think of them as similar to the memory buttons that you have on your calculator. You can store values in temporary variables and recall the values when you want to use them in a calculation.

Figure 7 shows the rest of the code for this example.

AllPr (all prices summed) and ZnTot (Zone Price tota) are temporary variables.

Temporary variables ***cannot be named in the column statement*** – because variables in the column statement become part of the CSI.

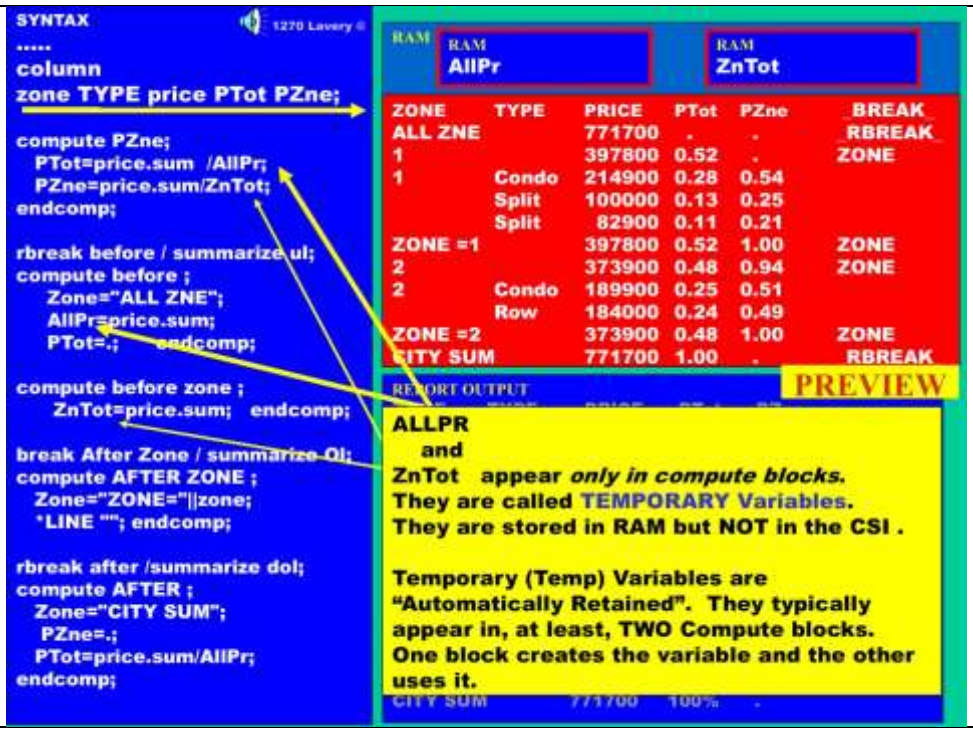


Figure 7

Temporary variables appear in at least two compute blocks. At least one of the compute blocks will move data from the CSI into the temporary variable, as "the proper" row in the CSI is being processed. At least one of the compute blocks will recall values from the temporary variable so that it can be used in a computation. In figure 7, you can see that AllPR (All Prices summed – yeah, not a great name) and ZnTot (Zone price total) do not appear in the column statement and do appear in compute blocks.

In figure 8 we start to process the first row of the CSI.

The "rbreak before / summarize ul" will make this row print and underline it in the listing.

In the "Compute before" statement we change the value of zone to be a more useful string.

NOTE: Download the animated slides and PLAY the slides to see several additional steps here.


Figure 8

We take the value of price for this row, which is the sum of the dollars in zones 1 and 2, and move it from the CSI to the temporary variable. This is how you store of value in a temporary variable.

I also set Ptot to be missing, though this is not required. Ptote is missing already and I was just being cautious (or/and confused) when I wrote that code.

| | |
|---|---|
| "Compute Var" blocks execute on every CSI row.<br><br>The statements in the "Compute PZne" block execute on this row but return missing because AllPr and ZnTot are missing.<br><br>Figure 9 shows us processing a "zone total row". It is not printed because there is no break line with a /summarize option. This code loads the total dollars for zone 1 into ZnTot |  |

Figure 9

Notice that there are two gold arrows from this compute block. This statement will execute a second time – just as we start processing rows for zone two.

In figure 10, we see an example of processing rows of data inside a zone.

The “Compute PZne” block executes and the two percentages are calculated.

This pattern repeats until the end of the block.


Figure 10

Figure 11 shows the processing of the last row in zone one.

We changed some text (Zone=) to make the report easier to read. We show the total dollars for this zone.

The “Compute PZne” block calculates dollars as percentage of the total (.52) and of its own (1.00).


Figure 11

The next row of data will allow us to change the value of ZnTot.

In figure 12 we see the processing for the *compute before zone* block.

The code just moves the value of price (which has been defined as a sum variable) from the CSI to the temporary variable ZnTot.


Figure 12

We now have a divisor we can use in calculating percentages for zone two. Please note that we did not have to change the divisor for the Ptot which is stored in AlllPr.

Figure 13 shows the calculations for a typical row inside of zone two.

Calculations are done because the "Compute PZne" block executes ad executes on every row.

A similar pattern is followed until we hit the summary row for zone two.


Figure 13

Figure 14 shows the processing for the summary row for zone two.

Two blocks of code execute.
They are:
"Compute PZne"

and "Compute After Zone".

The "break after Zone / summarize;" will cause the line to print.


Figure 14

Figure 15 shows the processing for the final row in the CSI. To make this final row of output, two sections of code must execute.

First the "Compute PZne" block executes to calculate the percentages.

Pzne is calculated and is calculated wrong (but we are saved by the timing of statement execution). SAS divides 771700 by 373900.


Figure 15

SAS resets this to missing in the "Compute after block". Then the "Compute after" block of code executes and this block deserves a bit of discussion.
We changed the text in the variable "zone" to "CITY SUM" to make the report easier to read. PZne is then set to missing. PZne was calculated incorrectly and, besides, does not make logical sense on this

row. We calculate PTot again, but this was not required. PTot had been calculated correctly in the "Compute PZne" block

## REFERENCES

It is useful to know the timing of calculations when coding complex PROC REPORTs.This talk – the PowerPoints and my voice, in its full 2 ½ hour length- was burned onto a CD and included in the back of the hardcover (only) version of Art Carpenter's excellent book on PROC REPORT.
If you have a chance to buy this book I would do so – especially if you can get the hard cover version. I appreciate the chance I had to work with Art on this project.

## ACKNOWLEDGMENTS

Thanks to all the helpful folks at SAS and especially those at Tech Support.

## RECOMMENDED READING

Carpenter's Complete Guide to the SAS REPORT Procedure (SAS Press)

## CONTACT INFORMATION

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

Russ Lavery Russ.lavery@verizon.net



APPENDIX A – code for the talk

```
/*************************************************************************************
*  Program name     : Exampled for Proc Report W Art Carpenter
*  Project          :
*  Written by       :
*  Date of creation :
*  Description      :
*  Macros called    :
*  Input file       :
*  Output file      :
*  Revision History :
*  Date      Author   Description of the change
*
*************************************************************************************/
Footnote "Prog Name: %sysget(SAS_EXECFILENAME) **By: &SYSUSERID ** Run on:
%sysfunc(datetime(),datetime16.)";

options nocenter ps=20;
ods listing;
ods html close;
```

```
/*****************************************************************
Section _A_: Create data
*****************************************************************/

data A01_House1;
infile datalines Missover firstobs=2;
Input @1 Type $char5.
      @6 Zone $Char6.
      @14 price
      @21 Bath;
datalines;
12345678901234567890123456789O
Split     3  179950 3
Condo     1  214900 3
Split     1  100000 4
Split     1   82900 3
Condo     2  189900 4
Row       2  184000 3
;
run;



/*****************************************************************
Section _B_: Sample Report 1
      detail level report with report break
*****************************************************************/

Proc report data= a01_house1 nowd headline  out=B01_out1;
title "Section B Sample Report 1 -- detail level report with report break";
column zone type price bath;
define zone / order    width=5;
define type / order ;
define price/ display  ;
define bath / sum width=5;

/*compute before ;*/
/* Zone="All";*/
/* Type="All";*/
/*EndComp;*/

Rbreak before
   /summarize dul;
run;

proc print data = B01_out1;
run;
```

```
/*****************************************************************
Section _C_: Sample Report 2
Detail Level Report with a: break after variable    RBREAK after and a compute
*****************************************************************/

Proc report data= A01_house1 NOWD headline   out=C01_out2;
title "Sample report 2 Section C  Report with a: break after variable        RBREAK after and a compute";
where zone GE '     2' ;
column
   zone type price bath;
define zone/order width=5;
define type / order;
define price/ sum  ;
define bath / mean width=5;
Break after zone   /summarize  ol skip suppress;
Rbreak after /summarize dol;
compute after  ;
    price.sum = .;
    Zone="ALL";
endcomp;
run;


/*****************************************************************
Section _D_: Sample Report 3 Summary Level Report with Rbreak
*****************************************************************/

Proc report data= A01_house1 nowd headline  out=D01_Out3;
title "Sample report 3 Section D Summary Level Report with Rbreak";
column zone  price bath;

define zone / Group width=5;
/*define type / Group;*/
define price/ sum  ;
define bath /mean width=5;

rbreak before /summarize ul;

compute before ;
  Zone = 'ALL' ;
endcomp;

where zone LE   '     2';
run;

**;
Proc report data= A01_house1 nowd headline  out=D01_Out3;
```

```
title "Sample report 3 Section D Summary Level Report with Rbreak";
column zone type price bath;

define zone / Group width=5;
define type / Group;
define price/ sum  ;
define bath /mean width=5;

rbreak before /summarize ul;

compute before ;
  Zone = 'ALL' ;
endcomp;

where zone LE   '     2';
run;

**;
Proc report data= A01_house1 nowd headline  out=D01_Out3;
title "Sample report 3 Section D Summary Level Report with Rbreak";
column zone type price bath;

define zone / Group width=5;
define type / order;
define price/ sum  ;
define bath /mean width=5;

rbreak before /summarize ul;

compute before ;
  Zone = 'ALL' ;
endcomp;

where zone LE   '     2';
run;

**;
Proc report data= A01_house1 nowd headline  out=D01_Out3;
title "Sample report 3 Section D Summary Level Report with Rbreak";
column zone type price bath;

define zone / order width=5;
define type / group;
define price/ sum  ;
define bath /mean width=5;

rbreak before /summarize ul;
```

```
compute before ;
  Zone = 'ALL' ;
endcomp;

where zone LE   '     2';
run;

**;
Proc report data= A01_house1 nowd headline  out=D01_Out3;
title "Sample report 3 Section D Summary Level Report with Rbreak";
column zone type price bath;

define type / group;
define zone / order width=5;
define price/ sum  ;
define bath /mean width=5;

rbreak before /summarize ul;

compute before ;
  Zone = 'ALL' ;
endcomp;

where zone LE   '     2';
run;

/*****************************************************************
Section _E_:  Sample Report 4a  When do different types of computes happen
*****************************************************************/

data E01_few;
infile datalines missover FIRSTOBS=3;

input @1  Sty  $char5. /*Style Of House*/
      @11 Regn $char1. /*region of the city*/
      ;
datalines;
12345678901234567890
Sty      regn
Condo     1
Condo     2
Row       2
Split     1
Split     1
Split     3
;
RUN;
```

```
proc report data=E01_Few out=E01_out4a nowd spacing=2;
title "Section _E_:  Sample Report 4a  When do different types of computes happen";

  column sty regn detl rept grup n;

  define sty   / order;
  define regn  / order;
  define detl  / computed; /*use how this is valued to determine the order of statement execution*/
  define rept  / computed;
  define grup  / computed;

  compute before;
  endcomp;

  compute detl;
    detl=333;
  endcomp;

  compute rept;
   rept=10;
  endcomp;

  compute after sty;
  endcomp;

  compute after regn;
   rept=3;
   grup=4;
  endcomp;

  compute after;
   grup=3/21;
   rept=rept*2.2;
  endcomp;
run;

options ps=30 ls=120 nocenter;;
proc print data=E01_out4a;
run;

/*****************************************************************
Section _F_: Sample Report 4b IF STATEMENTS IN COMPUTE BLOCKS
*****************************************************************/

proc report data=E01_Few out=F01_out4b nowd spacing=2;
title "Section _F_:  Sample Report 4b IF STATEMENTS IN COMPUTE BLOCKS";
```

```
 column sty regn detl rept grup n;

  define sty   / order;
  define regn  / order;
  define detl  / computed; /*use how this is valued to determine the order of statement execution*/
  define rept  / computed;
  define grup  / computed;


 compute before;
 endcomp;

 compute detl;
   if sty="" and _Break_= "_RBREAK_"  then detl=111;
   else if sty=""                     then detl=222;
   else if regn=.                      then detl=777;
   else                                 detl=333;
  endcomp;

  compute rept;
     rept=10;
  endcomp;

  compute after sty;
  endcomp;

  compute after regn;
     rept=3;
     grup=4;
  endcomp;

  compute after;
     grup=3/21;
     rept=rept*2.2;
  endcomp;
run;

proc print data=F01_out4b;
run;


/*****************************************************************
Section _G_: sample report 5 percentages and temp variables
*****************************************************************/
Proc Report data=A01_House1 out=G01_out5b NOWD;
column  zone TYPE price PTot PZne;
```

```
where zone LE   '     2';

define zone / order width=15;
define TYPE / DISPLAY width=8;
define price/ sum  ;
define PTot/computed ;
define PZne/computed ;

compute PZne;
   PTot=price.sum  /AllPr;
   PZne=price.sum/ZnTot;
 endcomp;

rbreak before / summarize ul;
compute before ;
     Zone="ALL ZNE";
     AllPr=price.sum;
     PTot=.;
 endcomp;

compute before zone ;
      ZnTot=price.sum;
endcomp;

break After Zone / summarize Ol;
compute AFTER ZONE ;
   Zone="ZONE="||zone;
   *LINE "";
endcomp;

rbreak after /summarize dol;
compute AFTER ;
   Zone="CITY SUM";
    PZne=.;
   PTot=price.sum/AllPr;
endcomp;
run;


/*****************************************************************
Section _H_: sample report 6 ACROSS Variables
*****************************************************************/

proc report data=A01_house1 out=H01_Example6_across nowd split="/";
column  type zone
        price=count
        price
        bath;
```

```
Define type    /group  width=12;
define zone    /across '# Prop./in  Zone'    width=2;
define count   /n '# Prop/on Mkt.'   width=11;
define price   /analysis mean format=6.1  'Avg./Price' width=10;
rbreak after   / dol  summarize skip;
run;


/*****************************************************************
Section _I_: Example report 7
*****************************************************************/
proc format ;
 value $ZonUL
       "     1" = "_1_"
       "     2" = "_2_"
       "     3" = "_3_";
run;

options ls=150 nocenter;
proc report data=A01_House1 NOWD out=I01_Compute_under_an_across
  split="/" spacing=1 ;
column  type ('_ZONE IN CITY_' zone),(price=count price bath doll_bath)  LAST ;

Where zone in('     1','     2');

Define type      /group      width=8;
define zone      /across     width=2    " " format=$ZonUL.;
define bath      /analysis sum "total/Baths" width=7;
define count     /n "# Prop/on Mkt."    width=7;
define doll_bath /computed "$ per/Bath" format=8.1;
define price  /analysis sum format=8.1      "Total/Price" width=10;
define LAST /computed noprint;
/*compute LAST;                                                                    */
/*  _c5_=_c3_/_c4_;                                                                */
/*  _c9_=_c7_/_c8_;                                                                */
/*endcomp;                                                                       */
rbreak after    / dol   summarize skip;
RUN;
QUIT;

/*****************************************************
example 8 use proc report instead of a transpose
*****************************************************/

options mlogic mprint symbolgen mcompilenote=all;


%macro rename_vars(dsn =
```

```
                ,var=
                                        ,first_across=
                                        ,prefix=
                );
proc sql;
  select distinct &var
        into :ValueList separated by " "
  from &dsn;

%global TheString;
%let TheString= ;
%let counter   =1;
%let ThisAcross=&first_across;
%do  %while(%scan(&ValueList,&counter,%str( )) NE );
 %let ThisValue=%scan(&ValueList,&counter,%str( ));

   %let TheString= &TheString _C&ThisAcross._=&prefix.&ThisValue;
 %let counter     = %eval(&counter+1);
 %let ThisAcross  = %eval(&ThisAcross+1)
;

 %put &TheString;
%end;

%mend rename_vars ;

%rename_vars(dsn =SASHelp.class
           ,var=Age
           ,first_across=2
                       ,prefix=Age_
           )

options nosymbolgen nomlogic;
proc report data=sashelp.class nowd
 out=Cool(rename=(&&TheString));
column sex age;
define sex/group;
define age / across;
run;

proc print data=cool;
run;
```

APPENDIX B -Cool Code Examples

```
/*****************************************************************
Section __Proc Report:
*****************************************************************/
```

/*This is so powerful that it is hard to describe how options interact*/
/*Basics are simple, but management usually wants fancy*/

/*Overviews*/
/*http://www2.sas.com/proceedings/sugi29/122-29.pdf RayP and DanB*/
/*http://www2.sas.com/proceedings/sugi28/071-28.pdf Lauren*/
/*http://www2.sas.com/proceedings/sugi31/142-31.pdf Louise*/
/*http://www2.sas.com/proceedings/sugi31/060-31.pdf Temp Var Molter*/
/*http://www.lexjansen.com/pharmasug/2007/ad/ad16.pdf report Null David, Lanie, Akari*/
/*http://www2.sas.com/proceedings/forum2008/170-2008.pdf compare and contrast keith*/

/*Art Carpenter wrote the best book and wrote great stuff on compute blocks*/
/*http://www.lexjansen.com/pharmasug/2005/handsonworkshops/hw07.pdf basics Art*/
/*http://www2.sas.com/proceedings/forum2008/188-2008.pdf Comput Block Practicum Art*/
/*http://www2.sas.com/proceedings/forum2008/188-2008.pdf Comput Blocks 2 Art*/
/*http://www2.sas.com/proceedings/forum2007/242-2007.pdf Advanced Compute Block Art*/
/*http://www2.sas.com/proceedings/forum2007/025-2007.pdf Naming Issues Art*/

/*http://www.lexjansen.com/pharmasug/2004/CodersCorner/CC07.pdf Compute Blocks Sharon*/

/*Ray and Daphnie - wrote the classic papers*/
/*http://nesug.info/Proceedings/nesug99/bt/bt138.PDF Still Not Using*/
/*http://www.lexjansen.com/pharmasug/2004/handsonworkshops/hw01.pdf Still Not Using*/
/*http://www2.sas.com/proceedings/sugi30/244-30.pdf Is it Pretty*/
/*http://www.lexjansen.com/pharmasug/2006/sasinstitute/sa04.pdf Doing It In Stye;*/
/*http://www.nesug.org/proceedings/nesug99/cc/cc060.PDF Missing Obs Column*/

/*Useful*/
/*http://www2.sas.com/proceedings/sugi29/242-29.pdf Getting up to speed Kimberly*/
/*http://nesug.info/Proceedings/nesug99/cc/cc117.PDF CHRIS ID option*/
/*http://www2.sas.com/proceedings/sugi30/259-30.pdf Gentle Intro Ben*/
/*http://www2.sas.com/proceedings/forum2008/079-2008.pdf Step-by-step David*/
/*http://www2.sas.com/proceedings/sugi27/p059-27.pdf Quick MeiMei, Sandra, Maria*/
/*http://www2.sas.com/proceedings/sugi30/036-30.pdf Statistical Nestor*/
/*http://www2.sas.com/proceedings/forum2008/173-2008.pdf complex Cynthia */
/*http://www.lexjansen.com/pharmasug/2002/proceed/TechTech/tt20.pdf Footnotes Angleina*/
/*http://www2.sas.com/proceedings/sugi24/Coders/p079-24.pdf*/
/*http://www2.sas.com/proceedings/forum2007/056-2007.pdf*/
/*http://www2.sas.com/proceedings/sugi22/HANDSON/PAPER149.PDF*/
/*http://analytics.ncsu.edu/sesug/1999/031.pdf Text flow donals and John*/
/*http://analytics.ncsu.edu/sesug/2007/CC15.pdf ExcelXP Elayne*/
/*http://www.lexjansen.com/pharmasug/2002/proceed/Coders/cc08.pdf Whole page Richard */

/*http://www2.sas.com/proceedings/sugi31/129-31.pdf style thaer*/
/*http://www2.sas.com/proceedings/forum2008/224-2008.pdf Color Style Wendy*/
/*http://analytics.ncsu.edu/sesug/2007/CC13.pdf EMAIL Theresa*/

```
/*Example 1 - basic stuff and dumping the internal file*/
title1 "There are some tricks to this - This is not Exactly the internal file";
title2 "Order supresses reopeating of vlaues in the internal file annd report";
title3 "missing Repeating values are added back in as thius file is created  ";

proc report data=sashelp.class nowd
        panels=2     /*Specify the number of panels on each page of the report*/
        PS=30 ls=120  /*Specify the number of columns/lines in a page*/
        split="*"    /*Specify the split character*/
        MISSING          /*Consider missing values as valid values for group, order, or across variables*/
        SPACING=2 /*number of blank characters between columns*/
        out=InternalFile             /*Holds internal file created by Proc Report*/
                ;
column sex  name NameSex age height;
 /*the Define sttement Supports preloadfmt and Exclusive for production reporting*/
        define sex         / /*order*/ Width=7 "Gender*of*Students";
        define name      / Width=7 "Student*Name";
        define NameSex          / computed Width=20 "Student*Name" FLOW;
        define age         / Sum Width=20 "Age" spacing=3;
        define Height     / mean Width=20 "Age" spacing=3;

compute NameSex / character length=30;
        If _break_ = "_RBREAK_" then
        namesex="Age is Sum & Height is avg.";
   Else namesex=trim(name)||" is a "||Sex||" On our records";
endcomp;
rbreak after /summarize DOL;
run;

proc print data=InternalFile;
title1 "There are some tricks to this - This is not Exactly the internal file";
title2 "Order supresses reopeating of vlaues in the internal file annd report";
title3 "missing Repeating values are added back in as thius file is created  ";
run;
title "";


/*Example 2 - across variables*/
proc report data=sashelp.class nowd
        split="*"    /*Specify the split character*/
        MISSING          /*Consider missing values as valid values for group, order, or across variables*/
        SPACING=2 /*number of blank characters between columns*/
        out=InternalFile             /*Holds internal file created by Proc Report*/
                ;
column sex age,(n=count height);
        define sex /group width=4;
        define age / across "-Count and Mean Height by Sex * Ages of Students-";
        define count / "Count*of*Students" center;
```

```
        define height / mean "Mean*Height" center;
run;


*example 3 From Art Carpenter***/
* Creating a new NUMERIC column with a compute block;
title1 'Using The COMPUTE Block';
title2 'Adding a Computed Column';
proc report data=sashelp.class split='*' NOWD;
column name sex NmSex(' Weight *--' weight wtkg);
        define name / order width=18 'Last Name*--';
        define sex / display width=6 'Gender*--';
        define NmSex  / computed  'Just A Character*Concatination*--';
        define weight / display format=6. 'Pounds*--';
        define wtkg  / computed  format=9.2 'Kilograms*--';

compute NMSEX /character length=20;
        NMSEX = strip(name)||"_"||Sex;
endcomp;
compute wtkg;
        wtkg = weight / 2.2;
endcomp;
run;
title "";


* example 4  - panels  ;
data classlong;
set sashelp.Class sashelp.Class sashelp.Class
        sashelp.Class sashelp.Class sashelp.Class;
spacer="*";
run;

proc report data=ClassLong panels=3 nowd  PSPACE=3;
column name sex age spacer;
define name      / width=10 spacing=1;
define sex                / width=4  spacing=1;
define age                / width =4 spacing=1;
define spacer    / width =1 spacing=3 "";
run;


* example 5 luo proc report
option background color\;
ods html file='C:\temp1\report1.html';


title1 "<a href='C:\temp1\report2.html'
```

```
     >report2</a>";



proc report data=sashelp.class split='*' NOWD style(column)={background=red};
column name sex NmSex(' Weight *--' weight wtkg) ;
          define name / order width=18 'Last Name*--' style(header)={background=yellow};
          define sex / display width=6 'Gender*--';
          define NmSex  / computed  'Just A Character*Concatination*--';
          define weight / display format=6. 'Pounds*--';
          define wtkg  / computed  format=9.2 'Kilograms*--';
compute NMSEX /character length=20;
          NMSEX = strip(name)||"_"||Sex;
endcomp;
compute wtkg;
          wtkg = weight / 2.2;
endcomp;
run;

ods html close;

data classlong;
set sashelp.Class sashelp.Class sashelp.Class
          sashelp.Class sashelp.Class sashelp.Class;
spacer="*";
run;

ods html file="C:\temp1\report2.html" ;
title1 "<a href='C:\temp1\report1.html'
          >Report1</a>";

proc report data=ClassLong panels=3 nowd
    style(column)={background=_undef_};
column name sex age spacer;
define name      / width=10 spacing=1 style(column)={background=red};
define sex                 / width=4  spacing=1 ;
define age                 / width =4 spacing=1;
define spacer    / width =1 spacing=3 "";
run;
ods html close;


/*************************************************************
Section __: Creating statistics
*************************************************************/
options ls=120 nocenter;

proc format;/*examples of creating formats - no use shown for these*/
```

```
    value $Gndr  M                    ='-Male-'  /*character format*/
                 F          ='-Female-';
run;

Proc report data=SASHelp.class nowd
           split="*" Spacing=4;
columns age sex,(N=count height);
define age /group width=10;
define sex  /across "-count and mean height by age & sex -" format=$Gndr.;
           define count / "Count*of*Students" center;
           define height / mean "Mean*Height" center;
run;

proc report data=sashelp.class nowd
                     split="*"    /*Specify the split character*/
                     MISSING           /*Consider missing values as valid values for group, order, or across
variables*/
                     SPACING=2 /*number of blank characters between columns*/
                     out=InternalFile              /*Holds internal file created by Proc Report*/
                     ;
column sex age,(n=count height);
           define sex /group width=4;
           define age / across "-Count and Mean Height by Sex * Ages of Students-";
           define count / "Count*of*Students" center;
           define height / mean "Mean*Height" center;
run;




/**********************/
data grocery;
  input Sector $ Manager $ Department $ Sales @@;
  datalines;
se 1 np1 50    se 1 p1 100   se 1 np2 120   se 1 p2 80
se 2 np1 40    se 2 p1 300   se 2 np2 220   se 2 p2 70
nw 3 np1 60    nw 3 p1 600   nw 3 np2 420   nw 3 p2 30
nw 4 np1 45    nw 4 p1 250   nw 4 np2 230   nw 4 p2 73
nw 9 np1 45    nw 9 p1 205   nw 9 np2 420   nw 9 p2 76
sw 5 np1 53    sw 5 p1 130   sw 5 np2 120   sw 5 p2 50
sw 6 np1 40    sw 6 p1 350   sw 6 np2 225   sw 6 p2 80
ne 7 np1 90    ne 7 p1 190   ne 7 np2 420   ne 7 p2 86
ne 8 np1 200   ne 8 p1 300   ne 8 np2 420   ne 8 p2 125
;
run;
```

```
/*libname proclib 'SAS-library';*/
options nodate pageno=1 linesize=80 pagesize=60
     /*  fmtsearch=(proclib)*/;


proc format /*library=proclib*/;
  value $sctrfmt 'se' = 'Southeast'
                 'ne' = 'Northeast'
                 'nw' = 'Northwest'
                 'sw' = 'Southwest';

  value $mgrfmt '1' = 'Smith'   '2' = 'Jones'
                '3' = 'Reveiz'  '4' = 'Brown'
                '5' = 'Taylor'  '6' = 'Adams'
                '7' = 'Alomar'  '8' = 'Andrews'
                '9' = 'Pelfrey';

  value $deptfmt 'np1' = 'Paper'
                 'np2' = 'Canned'
                 'p1'  = 'Meat/Dairy'
                 'p2'  = 'Produce';
run;


proc report data=grocery nowd
      headline headskip
      ls=66 ps=18;

 column sector manager (Sum Min Max Range Mean Std),sales;

 define manager / group format=$mgrfmt. id;
 define sector / group format=$sctrfmt.;
 define sales / format=dollar11.2 ;
title 'Sales Statistics for All Sectors';
run;


/*************************************************************************/
options nodate pageno=1 linesize=64 pagesize=30
       fmtsearch=(proclib);

proc report data=grocery nowd
          headline headskip;

 title 'Sales for Individual Stores';
 column sector manager department sales Profit;
 define sector / group noprint;
 define manager / group noprint;
```

```
 define profit / computed format=dollar11.2;
 define sales / analysis sum format=dollar11.2;
 define department / group format=$deptfmt.;

 compute profit;
     if department='np1' or department='np2'
        then profit=0.4*sales.sum;
     else profit=0.25*sales.sum;
 endcomp;

 compute before _page_ / left;
     line sector $sctrfmt. ' Sector';
     line 'Store managed by ' manager $mgrfmt.;
     line ' ';
     line ' ';
     line ' ';
 endcomp;

 break after manager / ol summarize page;

 compute after manager;
    length text $ 35;
     if sales.sum lt 500 then
        text='Sales are below the target region.';
    else if sales.sum ge 500 and sales.sum lt 1000 then
        text='Sales are in the target region.';
    else if sales.sum ge 1000 then
        text='Sales exceeded goal!';
    line ' ';
    line text $35.;
 endcomp;
run;
```