Paper 027-2009

TIPS AND TRICKS FOR CREATING THE REPORTS YOUR CLIENTS NEED TO SEE

Michael J Molter, INC Research, Raleigh, NC

ABSTRACT

Advanced reporting tools such as PROC REPORT have progressed by leaps and bounds in terms of options available to programmers for creating tables. Add to that the always-improving Output Delivery System (ODS), and it seems the sky is the limit for creating ready-for-delivery output straight from SAS®. Just when we start believing this, reality exposes limitations. Report templates are designed to maximize readability and to present data in a way that allows the reader to make well-informed decisions –without regard to programming challenges. Unfortunately for programmers, such tables cannot always be generated with the basic tools everyone talks about. In this paper we respond to such challenges by discussing some of PROC REPORT's and ODS's lesser known tools such as the Across variable, temporary variables, style attributes, inline formatting, and destination-specific markup. Along the way we will discover hiding places for some of the documentation of these features. In the end we will have bridged the gap between PROC REPORT 101 output and professional looking reports.

INTRODUCTION

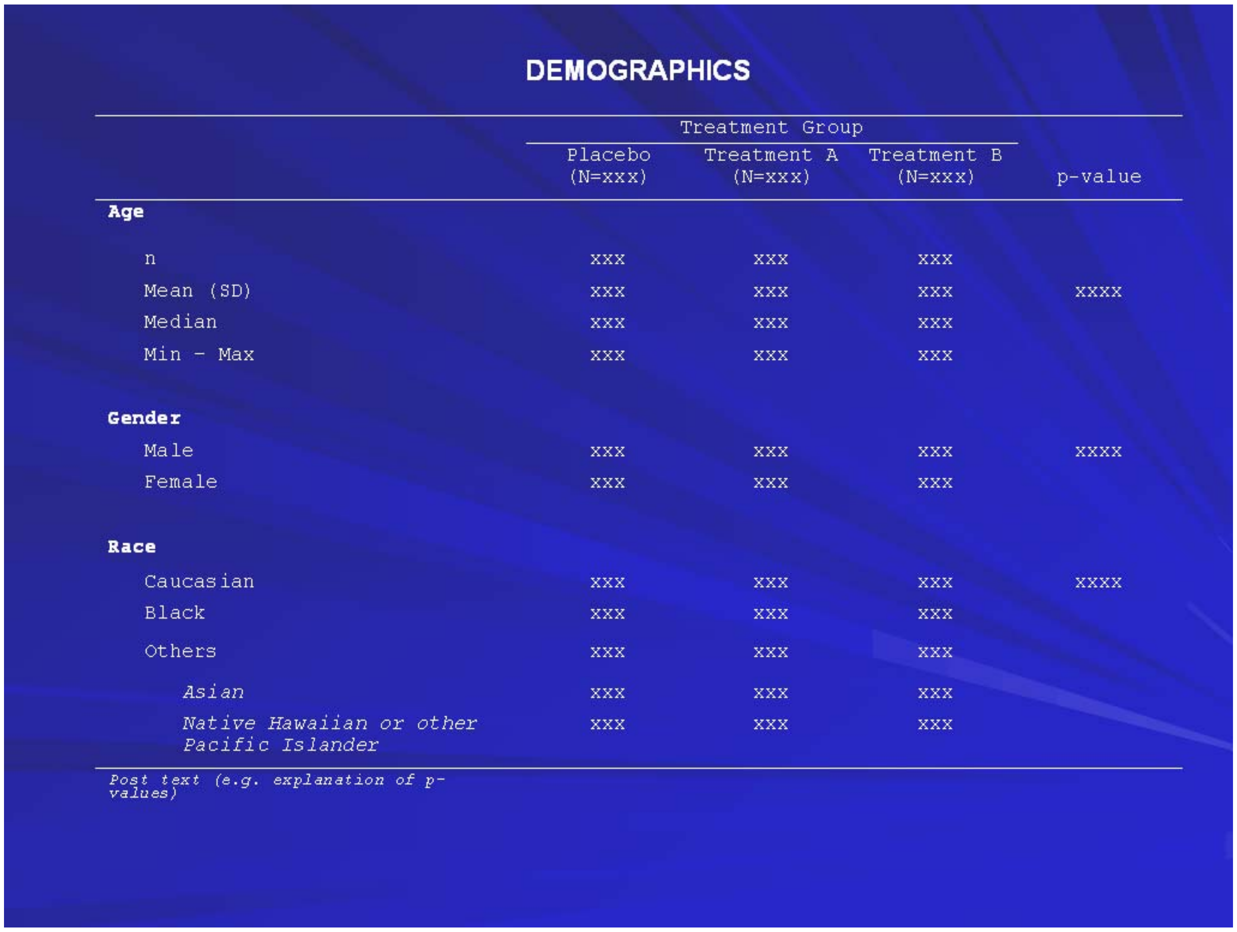
The REPORT procedure is another in a long line of PROCs with features aimed at getting us closer to producing deliverable output with minimal data preparation, manual intervention, and post hoc tweaking. Unlike other PROCs, REPORT has the statistical summary capabilities of the MEANS procedure, while at the same time, the reporting and display features that we sometimes associate with the PRINT procedure. We can supply the names of the variables from the data set that we want included in the table in the order we want as we would with the VAR statement in PROC PRINT, but also define any of them as Group variables, or variables whose values define the level of summarization as we do with the CLASS statement in PROC MEANS. Unlike PROC PRINT, we can "stack" variables on top of each other, we can stack headers and use them to span multiple columns, and we can actually create new variables "on the fly" as a function of other variables in the report. We can display detail rows, plus add summary rows that display either default or customized summary information. Whether we read about them in a book or see them discussed at a talk at a meeting, many of these additional features have made many of us jump on the PROC REPORT bandwagon to produce the reports our clients were looking for.

Having said all this, you might be surprised at how easy it is for a client, a statistician, or anyone else with an interest in the information you're delivering, and not so much the way in which you are producing it, to design a table with features that go beyond the basics of PROC REPORT that peaked our interest as programmers. Even if the initial design of the table is straightforward, even requests like "Could you just add a border here?" can be enough to send us diving into books and white papers, looking for that obscure trick that someone else who had a similar request discovered.

This paper will attempt to simulate such a situation. With a demographics table from a fictitious pharmaceutical study, we will identify features not covered in every PROC REPORT presentation, and develop step-by-step solutions to each. It is well understood that not every demographics table looks the same, that not everyone needs to know how to build a demographics table, and that not all readers are even in the pharmaceutical industry. The goal is not to learn how to create a demographics table. Rather, we are trying to bridge the gap between valuable lessons learned in PROC REPORT 101 and industry needs by significantly adding to our bag of tricks. Each feature discussed will find varying levels of usefulness among different readers, but even if none of the features specific to this demographics table apply to what you do, at the very least, you will have had exposure to multiple resources that may contain answers to the questions you do have.

THE SITUATION

The meeting is over, and you, the programmer, having not been in attendance, are about to see the results for the first time. Statisticians, client representatives, and administrative personnel, each of whom has read the statistical analysis plan (SAP), a document describing exactly what statistical summaries are needed to illustrate the safety and efficacy of the drug under development, have just met to collaboratively lay out "shells" or templates for the tables discussed in the SAP. Administrative personnel will now use point-and-click table making tools available in Microsoft Word to create the templates. Your job is to duplicate each of the templates, filled in of course with data, using PROC REPORT. Figure 1 illustrates what was decided on for the Demographics table.



DEMOGRAPHICS

	Treatment Group			
	Placebo (N=xxx)	Treatment A (N=xxx)	Treatment B (N=xxx)	p-value
Age				
n	xxx	xxx	xxx	
Mean (SD)	xxx	xxx	xxx	xxxx
Median	xxx	xxx	xxx	
Min - Max	xxx	xxx	xxx	
Gender				
Male	xxx	xxx	xxx	xxxx
Female	xxx	xxx	xxx	
Race				
Caucasian	xxx	xxx	xxx	xxxx
Black	xxx	xxx	xxx	
Others	xxx	xxx	xxx	
Asian	xxx	xxx	xxx	
Native Hawaiian or other Pacific Islander	xxx	xxx	xxx	

Post text (e.g. explanation of p-values)

Figure 1 – Demographics table shell

The following is a list of decisions regarding this table that were made during the meeting and are reflected in Figure 1 above.

- The table will concisely summarize multiple parameters (variables), lined up vertically and separated with a blank line, with the name of each parameter in bold face, the levels of each categorical variable indented under the name of the variable, and the names of the descriptive statistics indented under each numeric variable. Each comparison group will have its information summarized as described in the first column in a subsequent column, and the final column will contain information regarding a statistical comparison between the groups. Meeting attendees have in the past found this layout to be a convenient way to summarize multiple related but different pieces of commonly accessed information in one area while at the same time minimizing "clutter".
- A breakdown of the groups that make up the "Others" level of Race will be provided underneath "Others", with the text being indented further and italicized. Meeting attendees felt that this would help emphasize to the reader that these were, in fact, the groups that made up "Others" and not additional Race groups.
- Text would be added immediately following the table, not necessarily at the bottom of the page, to indicate the name of the statistical test(s) that generated the p-values.
- Horizontal borders would be added, not only for aesthetic purposes, but also to avoid text running together, such as the border that separates the name of each treatment group from the text "Treatment Groups."

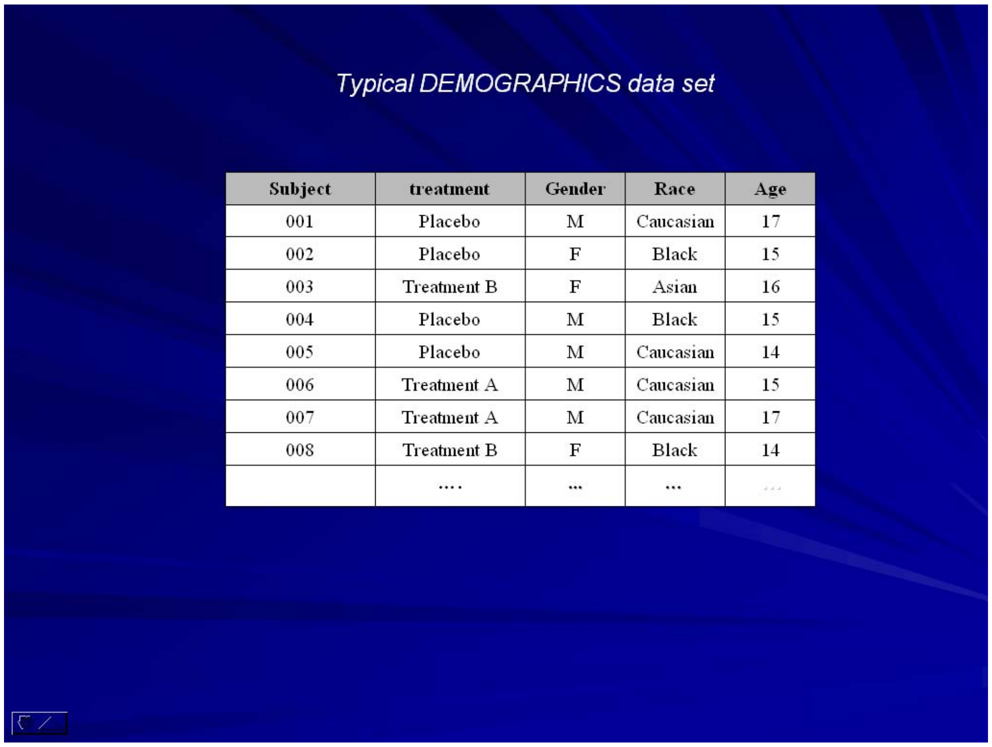
Other decisions such as font, font size, the presentation order of the parameters and their levels (or order of descriptive statistics), and text alignment were also made. As you can see, decisions were made, as they should be, by non-programmers based on readability, and the need to quickly and easily identify relevant information. It is the job of the programmer to say "yes, I can do that," and then go back to his/her desk and if necessary, research ways to get it done.

PREPARING THE DATA

It's often the case that the raw data must undergo some kind of preparation or manipulation before feeding it to PROC REPORT. Each of one hundred programmers may have their own unique way of preparing the sample data in this paper. The approach used in the following discussion is not claimed to be any better or worse than any other. The discussion that follows, while not a focus of this paper, is necessary to have something to work with.

WHAT'S WRONG WITH THE RAW DATA?

As the programmer, your first step is to compare the layout of the final table to the structure of the data set(s) you have to work with. An excerpt of the Demographics data set is illustrated below in Figure 2.



Typical DEMOGRAPHICS data set

Subject	treatment	Gender	Race	Age
001	Placebo	M	Caucasian	17
002	Placebo	F	Black	15
003	Treatment B	F	Asian	16
004	Placebo	M	Black	15
005	Placebo	M	Caucasian	14
006	Treatment A	M	Caucasian	15
007	Treatment A	M	Caucasian	17
008	Treatment B	F	Black	14
	….	…	…	…

Figure 2 – Demograhics data set layout

As expected, the data set contains raw data, while the table displays summarized data. For the moment this doesn't bother you because of the summary capabilities of PROC REPORT. What does bother you is that the column and row definitions of the table are exactly opposite of the analogous variable and observation definitions found in the data. For example, while the data set contains a variable called TREATMENT whose values are the different treatments administered in the study, the table contains a column for each treatment. Conversely, the parameters being analyzed and displayed vertically in the table each represent their own variables in the data set. This poses a problem because PROC REPORT is a column-driven PROC, meaning that each column in a PROC REPORT table corresponds in some way to a variable in the data set that feeds it. It appears that rather than feeding this data set to PROC REPORT, some amount of transposing of this data set will be required.

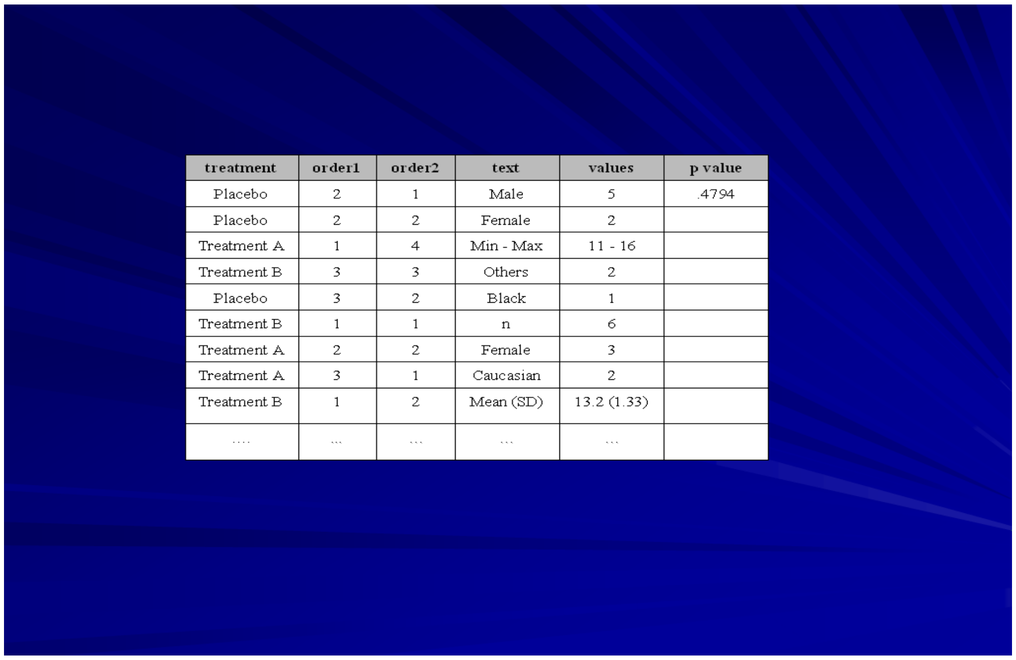
You're willing to compromise your database structure ethics by creating a new data set that has values of RACE and values of GENDER in one variable, but your ability to use the summary features of PROC REPORT run into jeopardy when considering how to generate and display the descriptive statistics of the numeric variable AGE. For starters, while PROC REPORT allows us to calculate multiple summary statistics of any one variable, placing their results in consecutive rows rather than columns can be quite difficult. Secondly, unlike the case with GENDER and RACE where the rows represent the different levels of the respective categorical variables and the data in each treatment column represents a frequency count for that level, the rows and the data in each of the treatment columns represent something different for the AGE parameter. That means that we can't define a column in the table with one unique definition, as is required by the COLUMN and DEFINE statements of PROC REPORT. Furthermore, the p-values in the last column are not accessible with PROC REPORT. You're now at the point you were hoping you wouldn't have to face – the summary features of PROC REPORT are useless for this purpose. The data in the table will have to be calculated as part of the data preparation process, and PROC REPORT will be used strictly for display purposes.

CREATING THE SUMMARY DATA SET

Now that we've come to grips with the fact that summary statistics will have to be calculated ahead of time, we'll hold off on any data set transpositions for now and work with each parameter independently. At this point it's a matter of personal choice deciding on which PROCs to use to analyze each of these parameters, and beyond our scope to discuss the details of any (including stat PROCs to get p values), so we'll skip ahead to the structure of the output data sets produced by these PROCs that contain treatment-specific summaries. We'll come back to the p value column later. Since parameters must be lined up vertically in the table, it makes sense to append the output data sets, but before doing that, we have to make them compatible – same variables, each with the same attributes. Since the table is displaying summaries of each parameter by treatment, a Treatment variable should be present in each output data set. We will refer to this variable as TREATMENT. The Gender and Race output data sets should each have a variable whose values are the different levels of these variables. We will refer to this variable as TEXT. As noted earlier, frequencies of each value of AGE is not part of the table, but multiple statistics of AGE are, and their descriptions are placed in the column that also holds the levels of the categorical variables. With a little extra work on the output data set from the analysis of AGE, these become values of TEXT. With the “grouping” variables in place, each output data set now only needs a variable to hold results. The character variable VALUE holds each statistic of AGE and frequency counts for each level of GENDER and RACE.

Two new numeric variables will be added for ordering purposes. The first, ORDER1, has a value of 1 for each observation in the AGE analysis data set (the first parameter displayed in the table), 2 for each observation from the Gender output data set and 3 for each from the Race output data set. We can use this to ensure that the parameters are displayed in the order specified in the template. A format is also added to the format catalog that maps values of ORDER1 to the text that describes them, found in the table at the beginning of each parameter. ORDER2 is added as a way to order the rows within each parameter. Within each value of ORDER1 the range of values for ORDER2 begins at 1 and increases by 1 with each row to be displayed for that parameter, forming a one-to-one correspondence between ORDER2 and TEXT.

Finally, a P-value variable is added to each output data set. In each of these data sets, the value of this variable will be missing for all observations, except one. This one observation will have a value of ORDER2 whose value corresponds to the row within the parameter on which the p value is to be displayed. The value of TREATMENT can be any one of the possible values. We’ll see soon that by choosing only one observation instead of all observations with the desired value of ORDER2 will allows us to define the P-value column in PROC REPORT as an analysis variable using the Sum statistic. An excerpt of the data set DEMO, constructed by appending the three fully modified output data sets described above is illustrated in Figure 3 below.



treatment	order1	order2	text	values	p value
Placebo	2	1	Male	5	.4794
Placebo	2	2	Female	2	
Treatment A	1	4	Min - Max	11 - 16	
Treatment B	3	3	Others	2	
Placebo	3	2	Black	1	
Treatment B	1	1	n	6	
Treatment A	2	2	Female	3	
Treatment A	3	1	Caucasian	2	
Treatment B	1	2	Mean (SD)	13.2 (1.33)	
….	…	…	…	…	

Figure 3 – DEMO data set

ACROSS VARIABLES – AN ALTERNATIVE TO DATA TRANSPOSING

Now that the output data sets have been modified and appended, it appears that the only step left is to transpose the data set so that values of TREATMENT define columns. While PROC TRANSPOSE or another DATA step would work, PROC REPORT’s ACROSS variable feature has the same effect without restructuring the data set. Whereas TREATMENT would play the role of the ID variable and VALUES would be the VAR variable in PROC TRANSPOSE, the COLUMNS statement would simply list TREATMENT, followed by a comma, followed by VALUES. The comma in the COLUMNS statement in PROC REPORT has the effect of stacking the variable to its left above the variable to its right in the table. TREATMENT is defined as an Across variable by using the keyword ACROSS to the right of the slash on the DEFINE statement. With this specification the values of the variable define columns, presented from left to right in the order of the formatted values.

```
/* The PROC TRANSPOSE way… */
proc transpose ;
id treatment ;
var values ;
```

```
   /* … vs. the Across variable */
   columns … treatment,values … ;
   define treatment / across ;
```

Circumstances dictate just how much of an advantage, if any, the Across variable is over data transposing. At the very least, it's one less step of data preparation. Options on the DEFINE statement that defines the Across variable such as ODS styling will affect all the columns that represent values of the variable. Most of the time this will be an advantage, since you only have to type them once. On occasion a need to treat one of the columns differently would force you into data transposing. Maybe the most significant advantage though is the case when each value of the Across variable is to be stacked on more than one variable. For example, a typical Adverse Events table often displays a patient count and an event count underneath each treatment. Assuming PATIENTS and EVENTS are two separate variables in the data set, without the Across variable, much more modification would be needed (e.g. two PROC TRANSPOSEs, one in which PATIENTS is the VAR variable, the other in which EVENTS is, followed by a merge). With the Across variable, TREATMENT can be stacked on two variables such as PATIENTS and EVENTS that are to be displayed side by side by placing them side by side in parentheses to the right of the comma.

```
   columns … treatment,(patients events) /* Without headers, or …*/


   columns … treatment,((“Patient count” patients) (“Event count” events)) /* … with
headers */
```

BUILDING OUR PROC REPORT

With our data set now in place as represented in Figure 3, and with our goal in mind as represented in Figure 1, we're ready to start building our report. Keep in mind that much of what we did to prepare the data was in preparation for the PROC. Let's start with a simple PROC REPORT that takes advantage of the structure of the data set, and then assess our status by comparing the result to the goal.

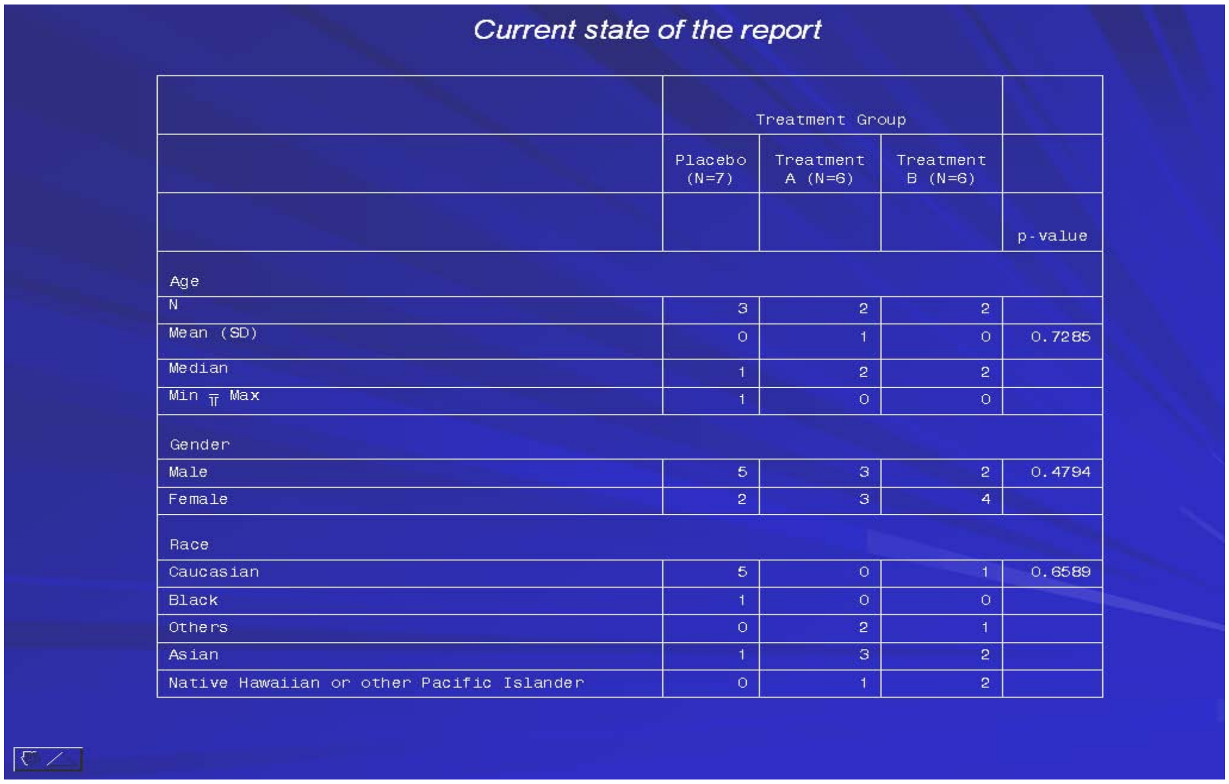
```
   proc format ;
   value head
   1 = ‘Age’
   2 = ‘Gender’
   3 = ‘Race’ ;
   run ;

   /* Commented numbers are in reference to the numbered items below the code */
   ods rtf file='demo1.rtf' /*1*/ style=minimal /*2*/;
   proc report nowindows data=demo split=”`” ;
   column order1 order2 /*3*/ text treatment,values /*5*/
   define order1 / group noprint order=internal ;
   define order2 / group noprint order=internal ;
   define text / group ‘ ' ;
   define treatment / across ‘Treatment Group’ /*8*/
       order = internal format = treat. ; /*6*/
   define values / group ‘ ‘ ;
   define pvalue / sum ‘p value’ /*8*/ format=pval. ; /*7*/

   compute before order1 ;
     line @1 ‘ ‘ ;
     line @1 order1 head. ; /*4*/
   endcomp;
   run;
   ods rtf close;
```

Let's begin with a few observations.

1. Note that this is creating an RTF file. Most of the content of this paper does not depend on the destination, and the small parts that do have analogous functionality with other destinations. This point will be reiterated when we get to those parts.
2. Note that the Minimal style is used. This will give us more opportunity to add styling options to our toolbox.
3. Note how ORDER1 and ORDER2 are used. Remember that these were created strictly for the purpose of ordering parameters and rows within parameters respectively. Their values themselves don't mean much to the reader which is why the NOPRINT option is specified on each of their DEFINE statements, but by placing them in the front of the COLUMNS statement, the rows are ordered exactly how we want.

4. Despite being a NOPRINT variable, which simply means that it won't take up a column in the report, the fact that the values of ORDER1 are in one-to-one correspondence with the parameters can be useful. Since the name of each parameter must appear on a non-data line immediately before its corresponding summary data lines, we can create a format (HEAD.) that maps values of ORDER1 to parameter names. Before each new value of ORDER1, we'll write out this parameter name with a LINE statement using the format.
5. Note the use of the comma to stack the Across variable TREATMENT on top of VALUES as discussed above.
6. Note the use of ORDER=INTERNAL on the DEFINE statement for TREATMENT. Recall that Across variables are ordered by their formatted values, but this option overrides that. The FORMAT=TREAT. option makes sure that the formatted values show up as column headers. This approach controls the order of the treatment columns in a way similar to the use of NOPRINT Order variables controlled the order of the rows.
7. Note that PVALUE is used as an analysis variable. This is strictly a function of the way the data was set up. Of course adding p values would never result in anything statistically meaningful, but because each unique p value shows up on only one observation, and each of those three observations represent different combinations of grouping variable values, no more than one p value will be added.
8. Note that TREATMENT and PVALUE have column headers specified in their respective DEFINE statements. Though the formatted values of TREATMENT will serve as column headers for their respective columns, the table template specifies "Treatment Group" as a spanning header over all three columns. TEXT is a self-explanatory column and needs no column header.

Figure 4 illustrates the initial results.



Current state of the report

	Treatment Group			
	Placebo (N=7)	Treatment A (N=6)	Treatment B (N=6)	
				p-value
Age				
N	3	2	2	
Mean (SD)	0	1	0	0.7285
Median	1	2	2	
Min ╥ Max	1	0	0	
Gender				
Male	5	3	2	0.4794
Female	2	3	4	
Race				
Caucasian	5	0	1	0.6589
Black	1	0	0	
Others	0	2	1	
Asian	1	3	2	
Native Hawaiian or other Pacific Islander	0	1	2	

Figure 4 – Results of initial PROC REPORT

Outside of several obvious needs for formatting, the good news is that almost everything is where it is supposed to be – with one exception. Note that the column headers are not lined up consistently. The column headers that represent formatted values of TREATMENT appear on a row above the header for the p value column that was specified in the p value's DEFINE statement. Technically, the formatted values of TREATMENT are considered by PROC REPORT to be spanning headers, whereas the column header specified in DEFINE statements is not. We can fix this by moving this column header out of the DEFINE statement and into the COLUMNS statement where headers are considered spanning. This will move "p-value" into the same row with the rest of the headers and eliminate the row from which this header moved.

```
columns … ("p-value" pvalue) ;
define pvalue / sum format=pval. " " ;
```

ODS STYLES IN PROC REPORT

While figure 4 looks like it has a long way to go before it starts looking like figure 1, adding some styling to our PROC REPORT will get us much closer. The REPORT, FREQ, PRINT, and TABULATE procedures are the only PROCs that allow us to implement ODS styles on the fly without having to resort to PROC TEMPLATE to build a new style template. This can save a lot of time as long as you have an idea of the style attributes and values that are available, and how they can be controlled within the PROC. Of course with so many attributes it's easy to forget all the names and possible values available, even if you do remember the functionality. For that reason, the SAS Online Documentation can be your best friend, as long as you know where to look.

Information on ODS styles is scattered throughout the Online Documentation. Before jumping into the specifics of this table, let's first see where we can find information specific to PROC REPORT, and then take a closer look at how to use it. We start by entering SAS's support website at www.support.sas.com. We then choose Product Documentation on the left side of the screen, followed by SAS 9.1, and then SAS OnlineDoc in the middle of the page. We're now in the Online documentation. Now expand Base SAS, Base SAS Procedures Guide, and Procedures, scroll down and click The Report Procedure. We now click on the Concepts: REPORT Procedure link at the top. This takes you to a long web page that begins with Laying Out a Report and Planning the Layout. Scroll down about two thirds of the page to Using Style Elements in PROC REPORT. This is what we want.

Near the beginning of this section is the syntax using typical SAS conventions with optional text enclosed in angle brackets. We'll take a moment here to discuss each part of this option.

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

The optional argument *location* refers to which part of the table you want to affect. Valid values specifically for PROC REPORT, found in the table in this section of the web page labeled Location Values, are REPORT, CALLDEF, COLUMN, HEADER, LINES, and SUMMARY. *Location* is optional because defaults are in place depending on the statement in which the STYLE option is used, as illustrated in the next table labeled Locations and Default Style Elements for Each Statement in PROC REPORT. In this table you'll notice that all values of *location* can be specified on one statement plus the PROC REPORT statement. The text near the bottom of this section explains that an attribute specification found on the PROC REPORT statement will always be overridden when the same attribute with a different value is found on the other statement. The advantage of making the specification in the PROC statement is that you only have to type it once and it will apply to all headers, lines, summaries, columns, or whatever location you have specified. If, for example, you want a green background color for ALL headers except one, specify green in the PROC statement and then override it with another color in the DEFINE statement corresponding to the column where the exception is to occur. Style-*element-name* is optional for the same reason, and default values are found in the same table. Unlike *location* whose values are specific to the PROC, style elements are specific to the style definition being used. Most of the time the defaults will be what you want, but elements may need to be specified at times when you're using custom style templates with custom elements. Finally, *style-attribute-specification* is a name-value pair of the form *attribute* = *value*. Scrolling down a bit further, you see two lists – the first of which contains attributes valid for the Report location, the second, attributes valid for all other locations. By clicking on the Style Attributes link in the paragraph that describes these lists, scrolling down to the bottom of the page and clicking Next Page, you can see the definition of each attribute and if applicable, valid values.

We're now ready to return to the Demographics table, and we'll start by repairing the Report location. Recall that these attributes can be specified on the PROC statement without specifying a location (since Report is default on this statement). One of the differences that stands out the most is all the borders. These can be controlled with the FRAME and RULES attributes, both of which only apply to the Report location. Following the links that define the attributes as described above, we see that both of these attributes have a discrete list of possible values. Among those for FRAME is HSIDES which puts borders only at the top and bottom of the table. Among values for the RULES attribute is GROUPS which places a border between table headers and the body of the table.

POSTTEXT is an attribute that's valid in any location. When using it in the Report location, we can add text to be displayed immediately after the table. Note that this is not a footnote, which is displayed at the bottom of the page. We'll use this to add information about the p values.

FONT_FACE and FONT_SIZE are also attributes allowed in all locations, but the list of Report location attributes has an asterisk next to them. Following this asterisk you'll see that when used in the Report location, these specifications only apply to pretext and posttext. As with the rest of the table, we'll use Times New Roman, but a smaller font size of 8 pt.

The PROC statement from above is now modified to include these attributes.

```
proc report nowindows data=demo split=”`”
style=[frame=hsides rules=groups posttext=”Explanation of p values”
    font_face=”times new roman” font_size=8 pt] ;
```

Again, because the desired location, Report, is the default for the STYLE option in the PROC statement, we didn't need to provide it. Keep in mind that we can also globally provide styling options for other locations from the PROC statement as long as we provide the name of the location. In our case we would like Times New Roman throughout the table. We'll also specify a cell width of 1.25 inches and center justify the data. The new augmented PROC statement is as follows.

```
proc report nowindows data=demo split="`"
style=[frame=hsides rules=groups posttext="Explanation of p values"
    font_face="times new roman" font_size=8 pt]
style(header lines)=[font_face="times new roman"]
style(column)=[font_face="times new roman" cellwidth=1.25 in just=center] ;
```

Note in the second STYLE option that when a set of attribute specifications applies to multiple locations, all can be listed in the location specification. You may also notice that JUST=CENTER was not applied to the Header location. That's because this is default in this location. Similarly the default font size, 10 pt, was acceptable and so was not necessary to specify.

We've accomplished quite a bit thus far in the PROC statement, but there are some exceptions that we need to take care of. For starters, the TEXT column needs to be wider than the rest. Also unlike the other columns, the text in this column needs to be left-justified. By specifying these attributes in the DEFINE statement for TEXT, we override the global COLUMN specifications made in the PROC statement.

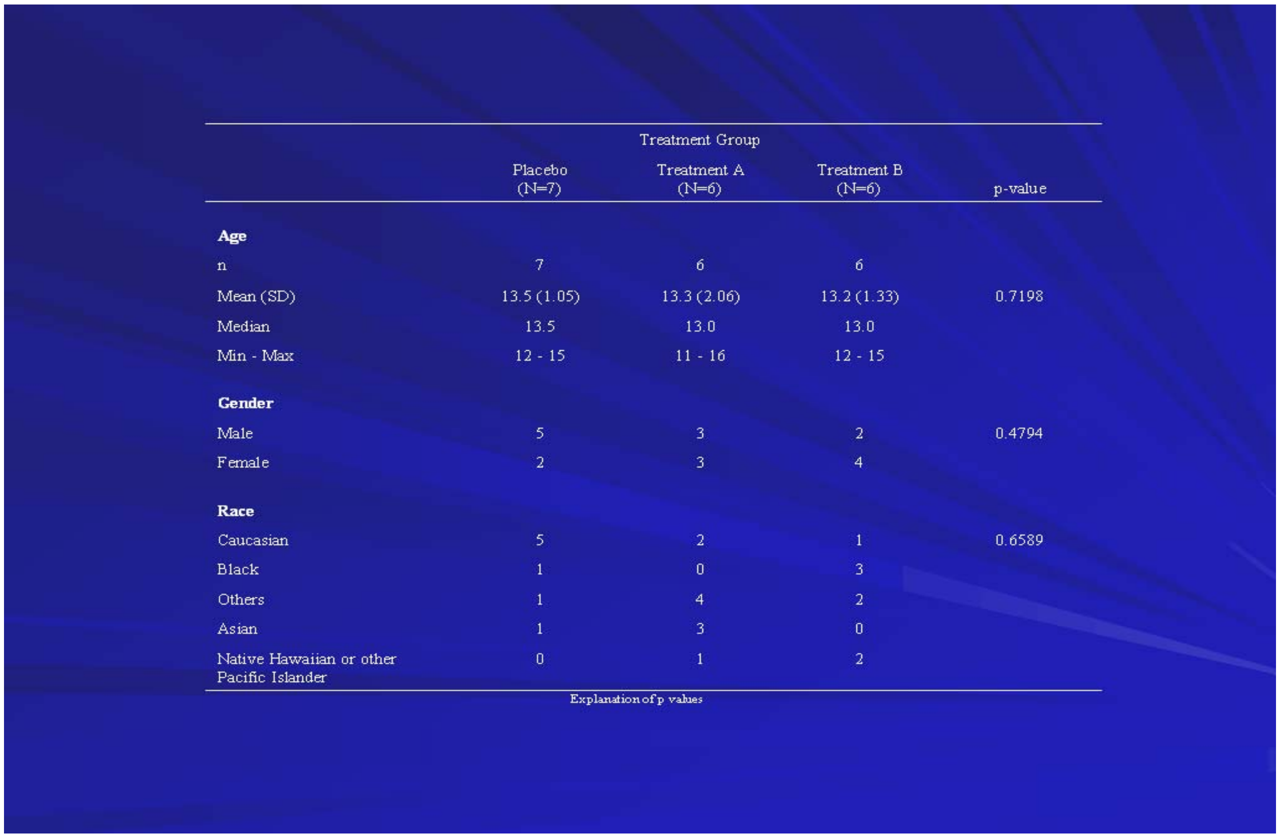
```
define text / group " " style=[cellwidth=2 in just=left] ;
```

We also want the parameter names on the non-data lines to be in bold. We make this change by modifying the COMPUTE statement.

```
compute before order1 / style=[font_weight=bold] ;
```

Alternatively, since all the non-data lines came from this one COMPUTE block, this attribute specification could also have been made in the PROC statement.

With all of these changes in place, figure 5 illustrates our current status.



	Treatment Group			
	Placebo (N=7)	Treatment A (N=6)	Treatment B (N=6)	p-value
Age				
n	7	6	6	
Mean (SD)	13.5 (1.05)	13.3 (2.06)	13.2 (1.33)	0.7198
Median	13.5	13.0	13.0	
Min - Max	12 - 15	11 - 16	12 - 15	
Gender				
Male	5	3	2	0.4794
Female	2	3	4	
Race				
Caucasian	5	2	1	0.6589
Black	1	0	3	
Others	1	4	2	
Asian	1	3	0	
Native Hawaiian or other Pacific Islander	0	1	2	

Explanation of p values

Figure 5 – Results with PROC REPORT style attributes

You'll notice that the posttext is centered underneath the table, whereas the template has it under the left side of the table. The reason for this is that the default value for the JUSTIFY attribute in the Report location is CENTER. One way to fix this is by changing that value.

```
proc report nowindows data=demo split=”`”
style=[frame=hsides rules=groups posttext=”Explanation of p values”
    font_face=”times new roman” font_size=8 pt just=left]
```

While this fixes the posttext problem, it does also have the effect of moving the whole table to the left side of the page. We'll offer another alternative in the next section.

DESTINATION LANGUAGE

We've come a long way from where we started, but we've just about gone as far as we can with ODS styles – it's time to find some other tricks. Our next two tasks will be to put the border underneath "Treatment Groups", and to indent the text in the data rows in the left columns. We'll do this by inserting text into the document that has meaning to the program used to open it.

When creating RTF and HTML files with ODS, we're actually creating a text file that contains ***markup***. What this means is that surrounding all of the text that we see when we open the file (e.g. titles, footnotes, headers, cell contents, etc.) is more text that we don't see upon opening the file unless we open it in a plain text editor. This "extra" text, or markup, serves to provide instructions to the program regarding how to display the file. Both RTF and HTML are considered markup languages, which means that each has their own set of instructions. Microsoft Word is a program that knows how to interpret RTF instructions; web browsers like Internet Explorer are programs that interpret HTML (though different browsers may interpret certain instructions a little differently than others).

Recall that the spanning header is present because we included it on the DEFINE statement for TREATMENT. With one more style attribute specified, we should be able to add the RTF text that instructs Word to insert borders, as below.

```
define treatment / across ‘Treatment Group \brdrb\brdrs’
style=[protectspecialchars=off] order = internal format = treat. ;
```

New text has now been added to the header. The 'brdr' stands for border. The 'b' at the end of the first one stands for bottom, so 'brdrb' means put a border on the bottom of the cell. The 's' on the end of the second one means to make it single thickness.

By default, ODS "protects" characters from being interpreted by the program. This is so that in the unusual case where you want \brdrb to be part of the text in the table, it won't disappear. However, in uses like the one above, we want this text to be interpreted by the program, so we must turn off this protection with the PROTECTSPECIALCHARS=OFF atttribute specification.

For the case of indenting, we want to place indenting instructions in front of the text of each cell in the left column. Of course none of this text, unlike the spanning header "Treatment Group", appears in our PROC REPORT. For situations like this we have the PRETEXT attribute. Recall that we used POSTTEXT earlier in the Report location to insert text at the bottom of the table. PRETEXT works the same with the obvious difference being that the text will appear before rather than after. In the Report location, prettext would appear before the table. In the Column location, it appears before the data cell content.

The RTF command for indenting from the left side is "li", followed by a numeric parameter that indicates how far to indent. RTF interprets all numeric parameters to be in a unit called twips. One inch is equal to 1,440 twips, which of course means that a half inch is 720 twips, and one quarter inch is 360 twips. The new DEFINE statement for TEXT follows.

```
define text / group ‘ ‘ style = [cellwidth=2 in just=left
pretext='\ql\li360 ' protectspecialchars=off]  ;
```

The "ql" command means to left justify. The PRETEXT attribute makes sure that every value of TEXT in this column is preceded by this text, which tells Word to left justify the text, then indent it one fourth of an inch. Once again we have to tell ODS not to protect these characters.

Using "ql" and "li" is another way to get the p value information under the left side of the table, but without having to left justify the table. If, by controlling the cell width of each column, you know the width of the table, then you should know how much white space falls between the edge of the margin and the beginning of the table (which will also depend on if you're using portrait or landscape). Convert this to twips and add the commands above to the beginning of your posttext specification in the Report location. See the full program in Appendix A for an illustration.

These are just a few of many commands available. For more information, visit support.sas.com, on the left side, click Focus Areas, Base SAS, ODS, SAS notes for ODS, and under the ODS RTF section, click Concepts. You can also find information on Microsoft's website at http://msdn2.microsoft.com/en-us/library/aa140277(office.10).aspx. Many resources are also available for the equivalent HTML commands, such as www.w3schools.com.

INLINE FORMATTING

Experimental in version 8.2, inline formatting adds just a few features to our bag of tricks, a few of which are unique and a few that overlap with features we've seen before. The term "inline", also expressed as "on the fly", implies something that we can use for customizing a specific file with a specific piece of code without having to create a general template. PROC REPORT styling syntax is a form of inline styling, as is inserting destination-specific language. However the term Inline Formatting here refers to a specific set of commands, each with their own syntax, that can be used to format specific parts of output. The advantage is that unlike the use of destination-specific language, most commands are driven by SAS syntax, and so in most cases, no knowledge of any markup language is required.

The necessary components for using inline formatting include an escape character, the command, and usually an argument. An escape character is defined with a global statement (not necessarily inside a DATA step or a PROC) in the follownig manner.

```
   ods escapechar="*" ;
```

Once defined, the presence of this character means that an inline function and its argument follow. Though the escape character can be any character, it is best to use a character that isn't otherwise needed for any other purpose (e.g. don't use the same character that you are using to define your split character). Some of the more useful functions include the following.

```
   ods escapechar="^" ;
   ^n /* Inserts a new line */
   ^{super text} /* text is inserted as a superscript */
   ^{sub text} /* text is inserted as a subscript */
   ^{dagger} /* inserts a dagger */
   ^R"text" /* insert raw text – very similar to examples involving  destination
specific language, but PROTECTSPECIALCHARS=OFF is not necessary */
```

These functions became experimental in version 8.2 but are supposed to be in production in 9.2. Additionally, more commands should become available and the syntax will be made more consistent. Currently the documentation is scarce, but for more information, type the URL http://support.sas.com/rnd/base/topics/expv8/inline82.html into your web browser, or enter support.sas.com, on the left side, click Focus Areas, Base SAS, ODS, then scroll to the bottom under the Archive section. Click *In-Line Formatting For ODS in SAS 8.2*. Also, in the SAS notes for ODS area (where you went for destination-specific language documentation), visit the Inline Formatting FAQs link under the section labeled ODS PRINTER family.

For our example, notice that no blank line is inserted between the border and the AGE header, whereas blank lines do separate the parameters from each other. The code we've used up until now included the following COMPUTE block.

```
   compute before order1 ;
     line @1 ' ' ;
     line @1 order1 head. ;
   endcomp;
```

The first line served to insert that blank line, but of course that also inserts an undesirable blank line before AGE. In order to get what we want, let's remove this LINE statement and slightly change the format we're using to write these headers.

```
   ods escapechar="^" ;
   proc format ;
   value newhead
       1 = "Age"
       2 = "^n Gender"
       3 = "^n Race" ;
   run ;
```

The new COMPUTE block now has the following form.

```
   compute before order1 / style=[font_weight=bold] ;
```

```
     line @1 order1 newhead. ;
   endcomp;
```

Note that the new format now includes the inline formatting command that inserts a blank line, but only for the second and third parameters to be displayed.

CONDITIONAL STYLING AND PROC REPORT'S TEMPORARY VARIABLE

We're now in the home stretch. Our final task is to apply special formatting to the values of TEXT that represent a breakdown of the data displayed in the OTHER row under the Race parameter. Recall that each row of the table represents a unique value of ORDER2 within a value of ORDER1. Also recall that we have already applied styling options to the TEXT column. What we would like is to be able apply custom styling options for these rows. In other words, we would like the ability to say "if ORDER1=3 and ORDER in (4,5), then apply these styling options." This can be achieved with the CALL DEFINE statement.

In short, CALL DEFINE allows us to set attributes such as formats and ODS styles on a cell by cell basis. The column is determined by the COMPUTE block to which the CALL DEFINE belongs, and the row(s) is determined by the conditions under which the statement is executed. On the surface, it would seem that the following statement should give us what we need.

```
   compute text ;
   if order1 eq 3 and order2 in (4,5) then
   call define(_col_,'style','style=[pretext="\ql\li720" font_style=italic]');
   endcomp;
```

Note the arguments of CALL DEFINE. The first argument represents the column to be affected. The automatic variable _COL_ means the column that the current COMPUTE block is defining. References to other column numbers can also be made. The second argument names the attribute and for the most part, the third is the value to be assigned to that attribute, though as can be seen, the value of a style attribute can include multiple attribute specifications. For more information, follow the CALL DEFINE link under PROC REPORT in the online documentation. Recall that the following style specifications were added to the DEFINE statement for the TEXT variable.

```
   define text / group ' ' style = [cellwidth=2 in just=left
   pretext='\ql\li360 ' protectspecialchars=off]  ;
```

From this we can see that the CALL DEFINE attempts to override the one-fourth-inch (pretext='\ql\li360') indentation with a one-half-inch indentation (pretext='\ql\li720'), and add italics.

Intuition and lots of DATA step experience tells us that the conditional statement in the COMPUTE block above should be executed for the appropriate rows of the table. The execution of it, however shows otherwise. To understand why, visit the How PROC Report Builds a Report link within the PROC REPORT area of the online documentation. Here we learn the steps PROC REPORT takes to build a temporary file on which conditional logic is based. The documentation doesn't go into any detail about the structure of this temporary file, but experimentation shows that the temporary file is structured similar to the way the table looks. For example, we know that unique values of GROUP (and ORDER) variables appear only once, and when such a value corresponds to more than one row in the table, empty cells appear underneath that value. In the corresponding temporary file, the same situation occurs – unique values of the GROUP variable appear on only one record ("observation"), and the rest of the values for that variable are missing. In our case, ORDER1 is a NOPRINT variable, but it is still part of the temporary file. Looking at the table, we can conclude that ORDER1 has a non-missing value in the temporary file only when ORDER2=1.

The remedy to this problem is PROC REPORT's relatively unknown temporary variable. Introduced in a COMPUTE block, the temporary variable is not found in the data set that feeds PROC REPORT, is not found in the COLUMNS statement or any DEFINE statement, and makes no appearance in the table. What it does do is retain its value until explicitly changed. Illustrated in How PROC Report Builds a Report, one common use is to create percent columns in the table with a denominator of your choice. By grabbing hold of the sum of a numeric variable at the first sight of each new value of a GROUP variable (e.g. COMPUTE before Groupvar), and storing it into a temporary variable, the percent field is defined as a Computed variable in which this sum plays the role of the denominator. In our case, we don't need to compute percents, but we do need to remember the value of ORDER1 on all rows to which it corresponds. We can do this by capturing its value and passing it to a temporary variable as follows. Since we already have a COMPUTE block for ORDER1, we'll simply add to it.

```
   compute before order1 / style=[font_weight=bold] ;
     line @1 order1 newhead. ;
     temp = order1 ;
   endcomp;
```

Now, each time ORDER1 changes value, its new value is passed to TEMP where it's retained. We can now use TEMP in place of ORDER1 as a basis for the conditional execution of the CALL DEFINE.

```
compute text ;
if temp eq 3 and order2 in (4,5) then
call define(_col_,'style','style=[pretext="\ql\li720" font_style=italic]');
endcomp;
```

This concludes the final piece of the puzzle. You have now accounted for all of the formatting features and the layout of the summaries, and are ready to deliver the product. Appendix A at the end of this paper shows the full PROC REPORT.

CONCLUSION

In this paper we've constructed a table that is probably not too far out of the ordinary when compared to some of the tables you may have been asked to construct. With this reasonable request, we've seen how in practice it can be fairly common to have to go beyond the basics of PROC REPORT 101 to accomplish certain tasks. Some of the specific tools we saw may not apply to what you are doing, and there may be tools out there that we didn't discuss that you need to add to your toolbox. To help you address those needs, we looked at different resources available to you – namely, hidden areas of the Online Documentation, different areas of support.sas.com inside the Base community, and websites to help you learn more about the markup that you are asking ODS to create. Hopefully the combination of the tools we looked at here and the tools you may add by spending some time with the documentation we've discussed will help you to bridge your own gap and restore your faith in PROC REPORT as a legitamate tool for building the tables your clients need to see.

REFERENCES

SAS Institute Inc. "Base SAS 9.1.3 Procedures Guide" SAS Online Documentation, Version 9.1.3. <http://support.sas.com/onlinedoc/913/docMainpage.jsp>

SAS Institute Inc. "Base SAS 9.1.3 Output Delivery System: User's Guide" SAS Online Documentation, Version 9.1.3. <http://support.sas.com/onlinedoc/913/docMainpage.jsp>

CONTACT INFORMATION

Please feel free to contact me with questions and comments.

Mike Molter
INC Research,
(919) 926-5710
mmolter@incresearch.com

APPENDIX A

In the following program we have elected to leave the table center-justified and use RTF commands to align the posttext underneath the left edge of the table. The following lists the differences between the code illustrated earlier in the paper and the code in the following program.

- In the PROC statement, the text inside the quotes that makes up the POSTTEXT attribute in the Report location (style=[…]) now includes "ql\li1440 " at the beginning.*
- In the PROC statement, the style attribute specification JUST=LEFT in the report location has been removed and replaced with PROTECTSPECIALCHARS=OFF due to the addition mentioned above.

* Why li1440? Since we know the exact width of each the columns, we know that the total table width is 7 in (TEXT = 2 in, each of the other column is 1.25 in). With the left and right margins set at 1 in each (OPTIONS statement) and because the output will be landscaped (11 total inches of paper), We know that the total amount of white space is whatever is left between the margins outside the table, which is 9 – 7 = 2. Since the table is centered, the 2 inches are split evenly between the left and right side of the table, which tells us that from the edge of the left margin to the beginning of the table, there is 1inch, or 1440 twips.

```
ods escapechar = ‘^’ ;

proc format ;
 value treat
   1 = 'Placebo`(N=7)'
   2 = 'Treatment A`(N=6)'
   3 = 'Treatment B`(N=6)' ;

 value newhead
   1 = 'Age'
   2 = '^n Gender'
   3 = '^n Race' ;
run ;

options orientation=landscape nonumber nodate leftmargin=1 in rightmargin=1 in ;
title “DEMOGRAPHICS” ;
ods rtf file='test.rtf' style=minimal;

proc report nowindows data=demo split=”`”
style=[frame=hsides rules=groups posttext=”\ql\li1440 Explanation of p values”
    font_face=”times new roman” font_size=8 pt protectspecialchars=off]
style(header lines)=[font_face=”times new roman”]
style(column)=[font_face=”times new roman” cellwidth=1.25 in just=center] ;

column order1 order2 text treatment,values (‘p value’ pvalue) ;

define order1 / group noprint order=internal ;
define order2 / group noprint order=internal ;
define text / group ‘ ‘ style=[just=left cellwidth=2 in pretext='\ql\li360‘
  protectspecialchars=off];
define treatment / across ‘Treatment Group \brdrb\brdrs’ format=treat.
  order = internal style=[protectspecialchars=off];
define values / group ‘ ‘ ;
define pvalue / sum ‘ ’ format=pval. ;

compute before order1 / style=[font_weight=bold] ;
  line @1 order1 newhead. ;
  temp = order1 ;
endcomp;

compute text ;
  if temp eq 3 and order2 in (4,5) then
  call define(_col_,'style','style=[pretext="\ql\li720" font_style=italic]');
endcomp;

run ;

ods rtf close ;
```