Am I Evil?? PROC TEMPLATE Exposed

Kevin P. Delaney MPH, Northrop-Grumman Mission Systems, Atlanta, GA

Abstract:

As more and more people begin to use the Output Delivery System (ODS) more people also want to modify the default output it produces. To do this properly one needs to become familiar with PROC TEMPLATE. Unfortunately PROC TEMPLATE has developed a reputation for being a scary and difficult procedure. Some have even gone so far as to call him "EVIL." In this paper we will get to know PROC TEMPLATE a little better, and you will learn how to create and modify both STYLE and TABLE templates, and hopefully become more comfortable with PROC TEMPLATE. All examples were developed with SAS 8.2 on the WINPRO platform, but should be extendable to other SAS versions and operating systems. This paper assumes some knowledge of ODS and of SAS procedures typically used to produce output.

Introduction

Since it's release as part of SAS 7.0, more and more people are starting to use the Output Delivery System (ODS). Many people would also like to be able to modify the default output that is produced by ODS. As a result more and more SAS users are also coming into contact with PROC TEMPLATE. If you are one of those SAS users, you may have found that PROC TEMPLATE can be tricky, and/or hard to deal with. Some of my colleagues have gone so far as to characterize this procedure as "EVIL." I sat down with PROC TEMPLATE for awhile to get an inside look at how he thinks, and acts. This has lead me to the conclusion that PROC TEMPLATE is not evil, but he ***is*** oft misunderstood. Hopefully this paper will give you a better insight into PROC TEMPLATE, and clear up some of the confusion that has been giving him a bad rap!

Why do we want to get to know PROC TEMPLATE at all? PROC TEMPLATE is VERY shy and modest. Truth is, he'd probably like it better if everyone would just leave him alone. SAS Institute provides 15 default STYLE templates with SAS 8.2, often choosing a different template is all you really need to do to change the way your output looks. But, as far as the appearance of output is concerned, PROC TEMPLATE is the main man. Every time you generate SAS output you are using at least one template, most times you are using two, and if you ever want to modify one of those templates (or create a new one) you will need to do it with PROC TEMPLATE.

What, exactly is a template? Part of the reason PROC TEMPLATE is so misunderstood is due to the fact that he has a split personality. There are actually several different types of templates. These have different roles in the appearance of SAS output, and have somewhat different PROC TEMPLATE code that defines them. STYLE templates control the outer appearance of the SAS output, they are the wardrobe, hair and makeup on SAS's output stage. The core of SAS's output performance is controlled by TABLE templates. Every procedure except the four reporting procedures (PRINT, FREQ, REPORT and TABULATE) has a TABLE template, which defines the structure of the output. Within or attached to a TABLE template can be several COLUMN and/or HEADER templates, which describe particular parts of an output table. These are the main templates that you would want to use or modify to change the way your output looks.

(**NOTE:** There is one other type of template that will gain increasing importance in SAS 9+, the STATGRAPH template. This type of template will be used in conjunction with the statistical graphics capabilities that will be available in upcoming versions of SAS. Once they become production (SAS 9.1) they will be valuable tools for creating high quality graphics to go along with the tabular output of the SAS/STAT procedures. They actually exist in SAS 9, but the syntax will change quite a bit between the experimental implementation in SAS 9 and what will be production for SAS 9.1, and therefore I will not discuss them any further in this paper.)

STYLE Templates

STYLE templates are the most obvious, well known and well understood of all the template types. Anytime you produce output to an ODS destination other than the LISTING, you are using a STYLE template. Table 1 below shows the default style template that is used by the three most common ODS destinations.

Destination	Default Style
ODS HTML	Default
ODS PRINTER (PDF, PS etc.)	PRINTER
ODS RTF	RTF

TABLE 1: Default styles used by the three most commonly used ODS destinations.

In version 8.2 of SAS there are 15 default styles that you may choose from, in Version 9 there will be at least 18 new ones in addition to all those from Version 8.2. If you don't like the way any of these styles look, you can create one of your own. However, because of all the parts of a STYLE template, this can be quite complicated. Probably the easiest way to begin to make a style is to take a look at the ones SAS gives you and modify them to your liking.

In order to see the attributes of a particular style (or to copy them to your Program editor to make changes), highlight the results window in your SAS session, and either RIGHT CLICK and select Templates, or go to the View dropdown menu and select Templates.

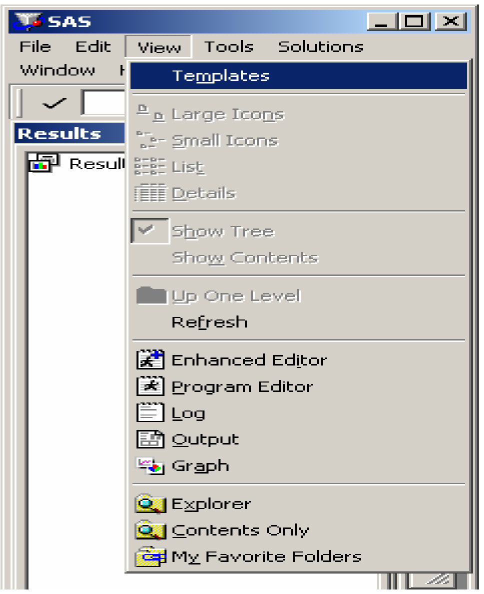


Figure 1: Viewing Templates

Figure 2 shows the contents of the TEMPLATE window. You will notice there will be at least two directories, SASUSER.TEMPLAT and SASHELP.TMPLMST. These are called TEMPLATE STORES and the two listed above are created by default. We will see how to create your own TEMPLATE STORE, and how to tell SAS which TEMPLATE STORES to use, in a little bit. SASHELP.TMPLMST contains all of the templates written and provided by SAS. SASUSER.TEMPLAT is the default location for new templates that you create, and for any templates that you modify (even a TEMPLATE master would not want to overwrite the templates that SAS wrote…). If you click on SASHELP.TMPLMST you will see there are several folders within it, listed alphabetically from BASE to TAGSETS. Nearly all of the templates listed are ***NOT*** STYLE templates, they are mostly TABLE, COLUMN or HEADER Templates. This makes sense when you remember that all but four of the SAS Procedures have their own TABLE templates, there are a lot of SAS Procs, so there are a lot of TABLE templates. The number of table templates is exacerbated by the fact that most procedures produce more than one output object (table) so they need more than one TABLE template. But we are supposed to be talking about STYLE templates now, so, which folder do you think contains the STYLE templates, that's right, the one called STYLES. If you click on the Styles folder you will get a screen that looks a lot like Figure 2, with the contents of the Styles folder listed in the right half of the screen. These are the 15+ STYLE templates that SAS ships as part of SAS 8.2. If you double click on a particular style you will open the TEMPLATE Viewer and be able to look at the entire STYLE Template. Let's click on the PRINTER template to check it out.

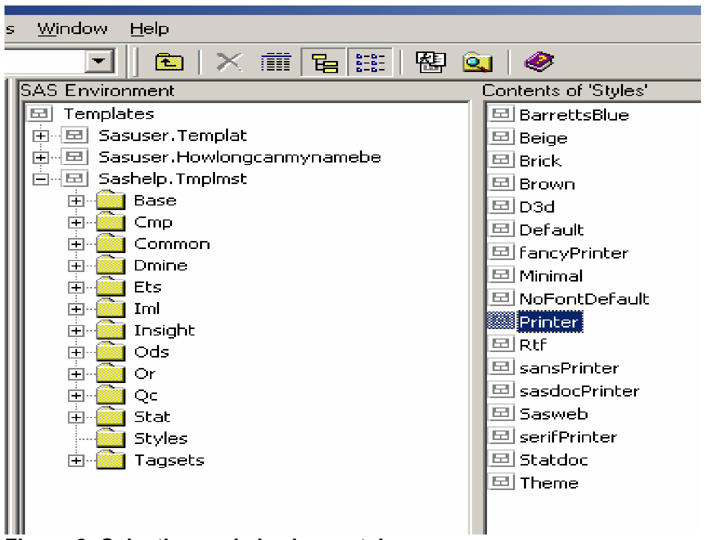


Figure 2: Selecting and viewing a style

Here we finally get our first look at the building blocks of a STYLE template, the PROC TEMPLATE procedure that creates it. The first three lines of this procedure are probably the most important to us at this point:

```
proc template;
  define style Styles.Printer;
    parent = styles.default;
```

The first line invokes the TEMPLATE procedure. The second line gives a name to the style being created; it tells us that this PROC TEMPLATE contains the style definition for STYLES.PRINTER. Notice that we are using the TEMPLATE procedure to DEFINE a STYLE, we will see later that this DEFINE statement is used for all templates, so it is necessary to tell PROC TEMPLATE that we want to work with his STYLE template personality, rather than a TABLE template or something else. This STYLE definition extends all the way to the END statement before a RUN statement which ends the procedure. Within the STYLE Definition there are two levels that divide the template into sections, STYLE elements and their STYLE attributes. Figure 3 shows part of a relatively short STYLE definition called MYNEWSTYLE. Within this Style definition are 3 Style elements, namely FONTS, COLOR_LIST and COLORS. Look at the way the syntax lays out for the COLOR_LIST Style element:

```
replace color_list /
'link' = blue
'bgH' = white
'fg' = Dark blue
'bg' = white
'fg2' = Red
        ;
```

Color_list Style element and it's attributes

Notice that all the STYLE attributes are listed as part of the same statement, that is the STYLE element starts with Replace style-element name, then lists all the changes to Style attributes, before reaching a semi-colon that ends the definition of the STYLE element. The syntax to set a given Style attribute is simply:

Style attribute = Value

Even in this short STYLE template there are an awful lot of Style attributes being set, imagine how many more would be necessary to create your own TEMPLATE from scratch! Ok, don't imagine, the entire DEFAULT template takes up 7 single space pages, even with Arial Narrow 8pt font! This is what I was referring to when I said it would be difficult to create a complete style template without copying some information from an existing template!!!

```
proc template;
  define style mynewstyle ;
    parent = styles.default;
    replace fonts /
 'TitleFont2' = ("Times New Roman",8pt,Bold)
 'TitleFont' = ("Times New Roman",8pt,Bold)
 'StrongFont' = ("Times New Roman",8pt,Bold)
 'EmphasisFont' = ("Times New Roman",8pt,Italic)
 'FixedEmphasisFont' = ("Courier New, Courier",7.5pt,Italic)
 'FixedStrongFont' = ("Courier New, Courier",7.5pt,Bold)
 'FixedHeadingFont' = ("Courier New, Courier",7.5pt,Bold)
 'BatchFixedFont' = ("SAS Monospace, Courier New, Courier",4.5pt)
'FixedFont' = ("Courier New, Courier",7.5pt)
'headingEmphasisFont' = ("Times New Roman",8pt,Bold Italic)
'headingFont' = ("Times New Roman",28pt,Bold Italic)
 'docFont' = ("Times New Roman",26pt, Italic);

replace color_list /
'link' = blue
'bgH' = white
'fg' = Dark blue
'bg' = white
'fg2' = Red
        ;
replace colors
     "Abstract colors used in the default style" /
      'headerfgemph' = color_list('fg')
      'headerbgemph' = color_list('bgH')
      'headerfgstrong' = color_list('fg')
      'headerbgstrong' = color_list('bgH')
      'headerfg' = color_list('fg2')
      'headerbg' = color_list('bgH')
      'datafgemph' = color_list('fg')
      'databgemph' = color_list('bg')
      'datafgstrong' = color_list('fg')
      'databgstrong' = color_list('bg')
      'notebg' = color_list('bg')
      'bylinefg' = color_list('fg')
      'bylinebg' = color_list('bg')
      'captionfg' = color_list('fg')
      'captionbg' = color_list('bg')
      'proctitlefg' = color_list('fg')
      'proctitlebg' = color_list('bg')
      'titlefg' = color_list('fg')
      'titlebg' = color_list('bg')
      'systitlefg' = color_list('fg')
      'systitlebg' = color_list('bg')
```

Style Elements

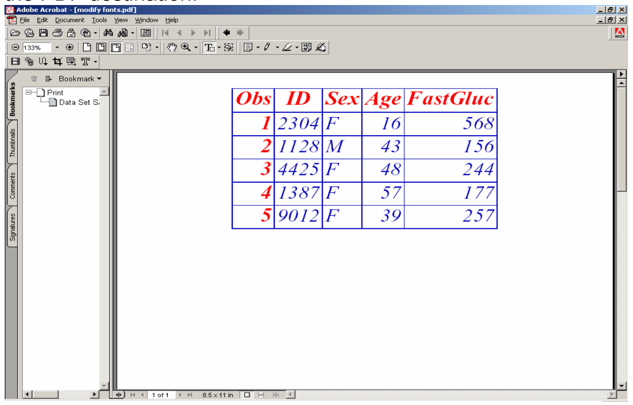
Style Definition (extends to the END at the top of the next page!)

Style Attributes

```
      'Conentryfg' = color_list('fg')
      'Confolderfg' = color_list('fg')
      'Contitlefg' = color_list('fg')
      'link2' = color_list('link')
      'link1' = color_list('link')
      'contentfg' = color_list('fg')
      'contentbg' = color_list('bg')
      'docfg' = color_list('fg')
      'docbg' = color_list('bg');
end;
run;
```

FIGURE 3: Sample PROC TEMPLATE STYLE definition, showing parts of a STYLE TEMPLATE

A PROC PRINT using the Style above produces the following in the PDF destination.



Obs	ID	Sex	Age	FastGluc
1	2304	F	16	568
2	1128	M	43	156
3	4425	F	48	244
4	1387	F	57	177
5	9012	F	39	257

Figure 4: PDF with Fonts modified using PROC TEMPLATE

Inheritance

No, this has nothing to do with Grandpa's Money; we are talking STYLE templates here! One of the things that makes PROC TEMPLATE such a confusing little procedure is the complexity of the links between templates which occur as a result of INHERITANCE. Perhaps the most important and one of the most confusing lines in this STYLE template is the one which says PARENT=STYLES.DEFAULT. This tells SAS what template STYLES.PRINTER ***inherits*** its attributes from. STYLES.DEFAULT has many attributes that you may never need or want to change. STYLES.PRINTER INHERITS a lot of these attributes directly from STYLES.DEFAULT. For me the easiest way to understand inheritance in STYLE templates is this, anything that is not explicitly changed in the new template will be inherited from the parent. That is, any attribute which not listed in STYLES.PRINTER will be set according to its definition in STYLES.DEFAULT.

There is also Inheritance of style attributes within a given template. Look again at the COLOR_LIST Style element from Figure 3:

```
replace color_list /
'link' = blue
'bgH' = white
'fg' = Dark blue
'bg' = white
'fg2' = Red
      ;
```

Notice how the colors are assigned to aliases such as 'link,' 'bgH' and 'fg'? Usually there are going to be a lot more STYLE elements that need a color defined for them, then there are colors in document. For instance in our PROC PRINT example (FIGURE 4) the data in the cells, and the gridlines are the same color blue. Rather than having to go in and change the colors of each STYLE element (remember how many there are in STYLES.DEFAULT) we assign colors references like 'link.' But we don't stop there, look at the COLORS element:

```
replace colors
      "Abstract colors used in the default style" /
      'headerfgemph' = color_list('fg')
      'headerbgemph' = color_list('bgH')
      'headerfgstrong' = color_list('fg')
      'headerbgstrong' = color_list('bgH')
```

In this syntax SAS is saying, in order to assign a color to the alias 'headerfgemph' use the value of 'fg' from the COLOR_LIST style element. Then further down in the Template, anywhere that you see 'headerfgemph' you would get COLOR_LIST('fg') which in our case is DARK BLUE.

For example, the style element Headeremphasis uses 'headerfgemph' as its foreground color:

```
Style HeaderEmphasis from Header
      "Controls emphasized table header cells." /
      foreground = colors('headerfgemph')
      background = colors('headerbgemph')
      font = fonts('EmphasisFont');
```

This STYLE element, three layers of inheritance away from where we actually set (changed) a color, is where the Value DARK BLUE actually gets assigned to the foreground for the HEADEREMPHASIS style. There is something else going on here as well. The syntax Style HeaderEmphasis from Header tells SAS to inherit more attribute values from another section of the template. In this case HeaderEmphasis gets most of its attributes from the HEADER Style element, the only difference between HeaderEmphasis and Header will be in the values we changed for Foreground (font) color, Background color and font. The purpose of this is to keep the size of the template small; you don't have to retype attributes of the Style element HEADER, if you aren't going to change them. However, this is the main thing about PROC TEMPLATE that makes it hard to understand, because attributes in the HEADER are inherited from another style element, and some of those attributes in that style element are inherited from somewhere else, you may have to really dig around in the TEMPLATE to figure out which style attributes are coming from where, and what you really are looking to change.

In my opinion Inheritance is the main reason people don't like PROC TEMPLATE, they don't take enough time to understand where he is coming from. As I have tried to show you above, there are many different types of inheritance all working simultaneously in a given Style template, so it might be a good time to stop and summarize inheritance, before we move on to some examples.

The real issue in figuring out all this inheritance stuff is understanding that there are really three different types of inheritance, and then knowing the order in which they are resolved:

1) Inheritance from a Parent Style, as assigned with a PARENT= statement
2) Inheritance from another Style element within a given TEMPLATE, via the StyleA from StyleB construct
3) Use of an alias to refer to a Style attribute value from another Style element (e.g. Color_LIST('fg') as part of a given Style attribute assignment.

There is one other thing about inheritance that I haven't touched on yet. You may have noticed that all of the Style elements in the example in FIGURE 3 begin with a REPLACE statement, while the ROWHEADER example above begins with a STYLE statement. These two ways of defining a Style element are fundamentally different, and directly related to the order of resolution of inheritance issues.

Order of Inheritance resolution:

1) PROC TEMPLATE takes any Style elements in the Child STYLE definition that were defined with a REPLACE statement, and puts them into the PARENT STYLE as is
2) PROC TEMPLATE resolves all the inheritance issues defined in the PARENT template
3) PROC TEMPLATE applies any new STYLE element changes as defined in the CHILD by a STYLE statement

To see how this works, lets break down the color changes we made in the PROC TEMPLATE code in Figure 3. First, what would happen if 'REPLACE COLOR_LIST' were instead 'STYLE COLOR_LIST'?

1) SAS looks for a REPLACE statement, but doesn't find one
2) SAS resolves all of the inheritances in the PARENT template, in this case STYLES.DEFAULT. What does this mean? Anywhere that SAS sees, for example BACKGROUND = colors("Headerfgemph") it first sticks in Color_LIST("fg") and then finally the color that corresponds to "FG" in the color list IN THE DEFAULT template.
3) SAS goes ahead and makes the changes it finds in STYLE statements in the CHILD TEMPLATE.

So if we had only said STYLE COLOR_LIST from COLOR_LIST SAS would have made the changes to COLOR_LIST ***AFTER*** the color attributes of individual style elements (such as HEADEREMPHASIS) had been defined (in step 2).

You may have noticed that there is a lot of code included in FIGURE 3 to change the color and typeface of two fonts. The only actual changes that we made to the Template are shown in **RED**. The problem is, because of the order of resolution described above, we have to REPLACE the color and font Style elements rather than modify them. This means that in Step 1 above, SAS does find a REPLACE statement in the child, so that whatever is written there goes into the PARENT templates corresponding STYLE ELEMENT Definition. All of the color and font aliases listed in FIGURE 3 are used at some point by the template STYLES.DEFAULT. So, if we had only included the aliases we wanted to change, (e.g. 'headingFont' = ("Times New Roman",**28pt,Bold Italic**)) SAS would have generated errors as it moved through STEP 2 and resolved all the inheritances in the default template. For example if we had only included 'Headingfont' in our REPLACE font STYLE element, when another STYLE element called for fonts('Strongfont') SAS would have looked for it in the FONTS STYLE element, and, not finding it, generated a error.

So in our case, since we were modifying the Fonts and colors in FIGURE 3, we had to list all of the fonts and all of the colors, in order to actually change two of them. This isn't really a big deal because you can copy and paste the attributes you want to change from STYLES.DEFAULT.

After this revelation, I was starting to understand that PROC TEMPLATE was a very complex procedure indeed. I decided to push on and try to get him to explain some of his other interesting features. I asked PROC TEMPLATE what other issues he thought people had with him, what other misunderstandings did he want to resolve. Feeling comfortable opening up to me, he gave me a few more examples.

RELATIVE FONTS

PROC TEMPLATE seemed distressed that people always got mad at him for not giving them the fonts they asked for when they were generating HTML files, they would give him one font size, and get two different ones depending on whether they were making a PDF or an HTML file. He lamented, "You don't know how many times I have heard, 'Why is a 28pt font not a 28pt font.'" He explained to me that this is not really his fault.

Through SAS 8.2 PROC TEMPLATE produces HTML documents written to be compliant with HTML version 3.2. In this version of the HTML standard, ***there are no absolute FONT sizes associated with a given object, only a relative size can be assigned***. This means that even if you specify a font size of 28pt in your STYLE template, SAS will convert it to a relative size when rendering the HTML. Usually this is the relative size 7, which will render a font somewhere between 30pt and 34pt depending on the browser and will then still be affected by the Text Size option set on an individual's computer. This can prove very annoying if you have a file that barely fits on a page in the PDF or RTF destination which you will lose control of in the HTML destination! In the latest HTML standard, HTML 4.0, absolute Font sizes can be used as part of a Cascading Style Sheet (CSS), allowing you to gain complete control of font sizes in HTML output. HTML 4.0 will become the default in SAS 9, but you can implement CSS in SAS 8.2. On your ODS HTML statement add the Stylesheet option:

```
ODS HTML body="PATH to a file.htm" stylesheet;
*PROC(S) that produce output;
ODS HTML close;
```

This, combined with listing absolute font sizes in your style template, will give you control of the font size in the HTML you create, regardless of how it is ultimately viewed by a user.

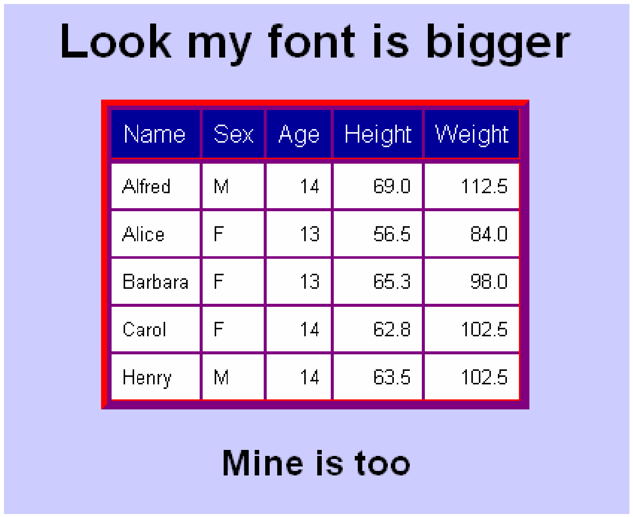
One other note on FONTS: You may have noticed that there were several different font faces listed for a couple of the aliases assigned in the FONTS style attribute (e.g. 'BatchFixedFont' = ("**SAS Monospace, Courier New, Courier**",**4.5**pt)). This is good practice when designing STYLE templates for HTML files. HTML does not embed fonts in a document, it keeps track of the name of the font, and renders the output in that font, if it is available on the computer of the person viewing the file. Although there are some fonts that will be available on almost every computer (Arial, Helvetica, Times New Roman) knowing whether a given font is available for a given viewer is nearly impossible. Thus it is good practice to give the browser some choices to pick from when it comes time to render a font.

BORDERS and RULES

In addition to colors and fonts, we can also change the RULES in the table, and/or the borders and shading. PROC TEMPLATE told me that getting this to work correctly is another point of contention that people have with him. This is due to two competing Style attributes that can affect the way RULES show up: the CELLSPACING = attribute and the RULES = attribute. CELLSPACING is the sneaky one here, if it is anything other than zero the background color of the table will show through in the spaces between the cells. RULES = generally works the way it is supposed to, with a choice of COLS, ROWS, ALL, NONE, and GROUPS as settings. With CELLSPACING set to 0, RULES = COLS will put lines between the columns, RULES= ROWS will place them between rows, and RULES=GROUPS will place a divider between the Header and the table area of the table. (Hopefully ALL and NONE are self-explanatory).

RULES= only effects the lines WITHIN the table, however, to affect the table border, you would use the FRAME = Attribute. FRAME = can take values of HSIDES, VSIDES, ABOVE , BELOW, LHS, RHS, and BOX. Box would put a border all the way around a table, HSIDES above and below the table, VSIDES on the left and right side, and the others on one specific side only. Setting FRAME= VOID will make sure no frame appears around the table. You may specify a single color for the border with the BORDERCOLOR= attribute, or a two color scheme with the BORDERCOLORLIGHT = and BORDERCOLORDARK = attributes. The code below illustrates some of these options, Comments 1-7 describe what is going on in particularly important places, FIGURE 5 shows the results of running this code:

```
Proc template;
define style Styles.bordersnrules;
  parent = Styles.Default;
/* Font changes */
replace fonts /
  'TitleFont2' = ("Helvetica, Helv",16pt,Bold Italic)
  'TitleFont' = ("Arial, Helvetica, Helv",18pt,Bold Italic)
  'StrongFont' = ("Helvetica, Helv",16pt,Bold)
  'EmphasisFont' = ("Helvetica, Helv",14pt,Italic)
  'FixedEmphasisFont' = ("Courier",12pt,Italic)
  'FixedStrongFont' = ("Courier",12pt,Bold)
  'FixedHeadingFont' = ("Courier",12pt)
  'BatchFixedFont' = ("SAS Monospace, Helv",12pt)
  'FixedFont' = ("Courier",12pt)
  'headingEmphasisFont' = ("Helvetica, Helv",16pt,Bold Italic)
  'headingFont' = ("Helvetica, Helv",10pt, Medium)
  'docFont' = ("Arial, Helvetica, Helv",10pt, Medium)
  /*1)  Add Three new font aliases*/
  "systemTitleFont"=("Arial, Helvetica, Helv", 20pt,Bold)
  "systemFootnoteFont"=("Arial, Helvetica, Helv", 16pt,Bold)
  "tableBodyFont"=("Arial, Helvetica, Helv", 8pt, Medium);
/* Color changes */
replace color_list
'Change color palette to something prettier' /
  'fgB2' = cx0066AA
  'fgB1' = cx004488
  'fgA4' = cxAAFFAA
  'bgA4' = cx880000
  'bgA3' = cxFFFFFF
  'fgA2' = cxFFFFFF
  'bgA2' = cx000099
  'fgA1' = cx000000
  'bgA1' = cx000000
  'fgA'  = cx000000
  'bgA'  = cxCCCCFF
  'blight' = red
  'bdark' = purple;

style RowHeader from Header
  'Change the default row fg/bg colors' /
   foreground  = Color_list("fgA2")
   /*2)  Notice how I avoid replacing the Colors Style
         element by referring directly to Color_list  when
         I am making changes*/
   background  = Color_list("bgA2")
   font=fonts("tableBodyFont");

style SystemTitle from TitlesAndFooters
  "Controls system title text." /
  font=fonts("systemTitleFont");
style SystemFooter from TitlesAndFooters
  "Controls system footer text." /
  font=fonts("systemFootnoteFont");
replace Data from Cell
  "Default style for data cells in columns." /
  foreground = color_list("fgA1")
  background = color_list("bgA3")
 /*3)  Changing the Background color here would actually change the
        background color of the data cells */
  font=fonts("tableBodyFont");

/* Table & Cell changes */
replace Output from Container
  "Abstract. Controls basic output forms." /
 /*4) Changing the background here changes what
       color  will show through the space between cells*/

   background = color_list('bdark')
   foreground = color_list('fgA2')
   rules = groups
  /*5) RULES= GROUPS puts a line between the header and table body
       FRAME=BOX puts a border around the whole table */
   frame = box
   cellpadding = 7
   cellspacing = 2
 /* 6)  CELLSPACING bigger than zero allows the Table background to show
through between cells.  Borderwidth sets the thickness of the outside border,
but not the rules.  You can use CELLSPACING to make the RULES look wider
*/
   borderwidth = 4

/* 7)  Finally, you can set a two color border by using BORDERCOLORLIGHT=
and BORDERCOLORDARK= together*/

   bordercolordark  = color_list('bdark')
   bordercolorlight = color_list('blight');
end;
run;

ods html  file="C:\temp\bordersnRules.html" style=bordersnrules;
proc print data=SAShelp.class (obs=5) noobs;
Title "Look my font is bigger";
footnote "Mine is too";
run;
ods html close;
```



Look my font is bigger

Name	Sex	Age	Height	Weight
Alfred	M	14	69.0	112.5
Alice	F	13	56.5	84.0
Barbara	F	13	65.3	98.0
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5

Mine is too

Figure 5: Changing Table Borders and RULES with STYLE templates

Other STYLE ELEMENTS

As I said before, the STYLES.DEFAULT template is over seven pages long, so thus far, we have only scratched the surface of what you can do with a STYLE TEMPLATE. I will present one more example using STYLE templates to demonstrate how complicated things can get. So far we have been looking only at the main body file in our HTML output, but SAS can create a table of contents and a frame file for your HTML files as well. Let's add a few more outputs and a Table of Contents to our HTML document from before.

The relevant STYLE TEMPLATE code (add this to the code from the previous example) is included below:

```
replace Contents from document
"Controls the Contents file." /
      bullet = "decimal"
      tagattr = " onload=""if (msie4 == 1)expandAll()"""
      pagebreakhtml = html('break')
      rightmargin = 8
      leftmargin = 8
 /* Color changes: Change the background color here to match the body file
    It would be nice if changing the foreground color here had an effect, but it
    is set further down the trail of inheritance*/
```

```
      background=cxCCCCFF
      Foreground=Purple;
replace IndexItem from Container
      "Abstract. Controls list items and folders for Contents and Pages."  /
      leftmargin = 6pt
      posthtml = html('posthtml flyover')
      prehtml = html('prehtml flyover bullet')
      listentryanchor = on
      bullet = NONE
      background = cxCCCCFF
      foreground = Purple;
  /* Color changes here don't do it either*/
replace ContentFolder from IndexItem
      "Controls the generic folder definition in the Contents file." /
      listentryanchor = off
      foreground = purple;
 /* Color changes still don't take effect*/
replace  IndexProcName from Index
      "Abstract. Controls the proc name in the list files." /
      listentryanchor = off
      bullet = "decimal"
      posthtml = html('posthtml flyover')
      prehtml = html('prehtml flyover')
      posttext = text('suffix1')
      pretext = text('prefix1')
      foreground = purple;
 /* Finally get the Purple color I was looking for in the links*/

  replace ProcTitle from TitlesAndFooters
      "Controls procedure title text." /
      background = colors('proctitlebg')
      foreground = cxCCCCFF;
  /* PROC MEANS and PROC FREQ have extra system titles that I don't want
     see, so set them to be the same color as the background (There is probably
     a better way to do this*/

ods html  contents="C:\temp\Contents1.html" frame="C:\temp\frame1.html"
          file="C:\temp\bordersnRules.html" style=bordersnrules stylesheet ;
proc print data=SASHelp.class (obs=5) noobs;
run;
Proc means data=sashelp.class;
var weight height;
run;
Proc freq data=sashelp.class;
tables sex;
run;
ods html close;
```

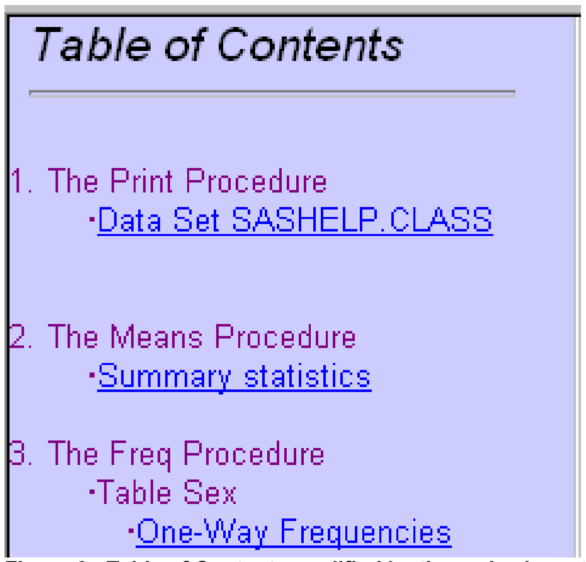


Table of Contents

1. The Print Procedure
  ·Data Set SASHELP.CLASS

2. The Means Procedure
  ·Summary statistics

3. The Freq Procedure
  ·Table Sex
    ·One-Way Frequencies

Figure 6: Table of Contents modified by the code above to look like the body file

This Example shows how the nesting caused by inheritance can make modifying STYLE templates tricky, if we had left out any of those CONTENTS file modifying STYLE elements, our output just wouldn't have looked quite right. But by following the layers of inheritance in the STYLE template, you should, with the help of a little trial and error, be able to modify the STYLE Templates to get you close to what you want.

That's about where we are now with our running example, we have gotten the TABLE OF CONTENTS colors and fonts looking pretty good, but personally I would prefer the actual text in the TOC to be a little more descriptive. Do I have control over that?? You bet, but not from a STYLE template!!

TABLE Templates

To this point, we have only been talking about STYLE Templates, but these only control the outward appearance of our output. What if we want to change the layout of the output objects, or maybe change a column header, the TABLE of CONTENTS label, or to format a specific cell a certain way? For this we have to modify TABLE TEMPLATES.

Almost every SAS procedure has a TABLE template attached to it. In fact there are only four that don't (PRINT, REPORT, TABULATE and FREQ). How do we know which TABLE TEMPLATES are used with which procedure output? We can find this out by using ODS TRACE to tell us which tables are used to create our output. Adding the statement ODS TRACE ON; before our previous example results in the following information being added to our log.

```
Output Added:
-------------
Name:       Print
Label:      Data Set SASHELP.CLASS
Data Name:
Path:       Print.Print
-------------
Output Added:
-------------
Name:       Summary
Label:      Summary statistics
Template:   base.summary
Path:       Means.Summary
-------------
Output Added:
-------------
Name:       OneWayFreqs
Label:      One-Way Frequencies
Template:  Base.Freq.OneWayFreqs
Path:       Freq.Sex.OneWayFreqs
-------------
```

This provides two important pieces of information:

1. The TEMPLATE line for both PROC MEANS (middle output) and PROC FREQ (bottom output) tell us where to find the tables we want to modify.
2. The output for PROC PRINT is shown with no TABLE TEMPLATE listed in the ODS TRACE output for this procedure. Since the "shape" of the output from the PRINT procedure (as well as REPORT, TABULATE and a MULTI-WAY FREQ) can't be "known" ahead of time, it is impossible to have only one TABLE TEMPLATE for these procedures.

OK, so from this information we know that the PROC MEANS output is using the TABLE TEMPLATE Base.summary, and PROC FREQ is using BASE.FREQ.ONEWAYFREQS. These Templates are bound to their corresponding procedures by name, that is, MEANS will always use BASE.SUMMARY. But where are they located, and how do we modify them?

We can modify the location of the Templates that ODS uses, and tell SAS where we want to store new templates using the ODS PATH statement. For example:

```
ods path work.mytemp(update)
         sasuser.templat(update)
         sashelp.tmplmst(read);
```

Much like the FMTSEARCH option, this sets up a list of TEMPLATE stores in which SAS will look for both TABLE and STYLE templates (and any other types of TEMPLATES as well). In this example we are telling SAS to first look in the temporary TEMPLATE store WORK.MYTEMP and then in the two permanent TEMPLATE stores that we saw earlier, SASUSER.TEMPLAT and SASHELP.TMPLMST. SAS will look for templates in these TEMPLATE stores in the order they are listed, and use the first one that it finds. Also, since we have listed WORK.MYTEMP as the first TEMPLATE store with update status, any new or modified templates we create will be stored there by default. Finally, by setting SASHELP.TMPLMST to read only, you can't help but copy existing templates, and don't have to fear overwriting them.

So now that we know where to find existing TABLE templates, and how to access new ones, how do we edit them? Another reason PROC TEMPLATE is so misunderstood, is the fact that he has a bit of a split personality. We have seen some of the quirks of his STYLE template side, but things are quite different, yet just as complicated, when dealing with his "TABLE Template personality." His TABLE template side has very different syntax from the STYLE templates, as shown in TABLE 2 below:

STYLE Templates	TABLE Templates
Starts with a DEFINE STYLE statement	Starts with either a DEFINE TABLE or a EDIT TABLE statement
Has STYLE elements that start with a STYLE or REPLACE statement	HAS other DEFINE statements that describe TABLE items (e.g. DEFINE COLUMN, DEFINE HEADER)
A STYLE element can have many attributes defined in one statement (ending with one semicolon) STYLE element Attribute1=value Attribute2=value;	Each statement within a Table Item definition has its own Semicolon Define Table Item; Attribute1=value; Attribute2=value; End;
A REPLACE element can have many attributes defined in one statement (ending with one semicolon) REPLACE element Attribute1=value Attribute2=value;	Each statement while editing a TABLE ITEM has its own Semicolon EDIT Table Item; Attribute1=value; Attribute2=value; End;

Table 2: Syntax differences between STYLE and TABLE templates

BASIC syntax

If I were to propose a basic model for editing TABLE templates it would be to:

1) Identify which template you need to edit using ODS TRACE
2) Use the Template viewer to take a look at the current structure of the template
3) Use syntax similar to that provided below to edit the template

```
Proc template;
edit Name-of-template-to-edit;*1)  Tell sas which template to edit;
column col1 col2 ... coln;*2)  List the columns in the template;
             edit col1;*3) Edit each column as necessary;
             Header="header-text";
             format= formatname.;
             other TABLE template statements;
             end;
             edit OThercols;

             End;
Other Table template statements go here;*4) Add other Template modifiers
                                             here;
end;
run;
```

TABLE TEMPLATES: Changing a format in a calculated column

One of the attributes set as part of a TABLE template is the format for a calculated column, such as the Frequency count in PROC FREQ. We can change the format of a dataset variable in PROC FREQ using the format statement, but in order to change the format of a calculated variable, we will have to change the TABLE template. Recall from our adventures with ODS TRACE that the TABLE template for a One-way Frequency table is BASE.FREQ.ONEWAYLIST. If you open it up in View Templates you will see (minus the comments I have added to describe the template:

```
proc template;
  define table Base.Freq.OneWayFreqs;
      *1)  Sets up inheritance from the table defined below;
    parent = Base.Freq.OneWayList;
    notes "One-Way Frequency table";
  end;
  define table Base.Freq.OneWayList;
    notes "Parent for One-Way Frequency table and LIST table";
    dynamic page needlines plabel varlabel lw varjust gluef gluep;
    column Line FVariable Variable FListVariable ListVariable Frequency
      TestFrequency Percent TestPercent CumFrequency CumPercent;
     *2)  Column statement lists all the columns that are defined below;

    header h1;
    translate _val_=._ into "";
    define h1;
      text varlabel;
      space = 1;
      split = "";
      spill_margin;
      highlight;
    end;
    define Line;
      header = "Line";
      format_ndec = 0;
      format_width = lw;
      just = c;
      style = RowHeader;
      id;
    end;
    define FVariable;
      just = varjust;
      style = RowHeader;
      id;
      generic;
    end;
    define Variable;
```

```
     print = OFF;
     generic;
   end;
   define FListVariable;
     just = varjust;
     style = RowHeader;
     generic;
   end;
   define ListVariable;
     print = OFF;
     generic;
   end;
   define Frequency;
  *3)  The first column we are interested in, this is the count  variable in the
       frequency table.  It's default format is best8. but we will change that to
       5.2 in a moment;
     header = "Frequency";
     glue = gluef;
     format = BEST8.;
     just = c;
   end;
   define TestFrequency;
     header = ";   Test  ;Frequency;";
     glue = 4;
     format = BEST8.;
     just = c;
   end;
   define Percent;
 *4)  Take note of the Percent column as well, we might as well give them the
same format;
     header = "; Percent;";
     glue = gluep;
     format = 6.2;
     just = c;
   end;
   define TestPercent;
     header = ";   Test; Percent;";
     glue = 3;
     format = 6.2;
     just = c;
   end;
   define CumFrequency;
     header = ";Cumulative; Frequency;";
     glue = 4;
     format = BEST8.;
     just = c;
   end;
   define CumPercent;
     header = ";Cumulative;  Percent;";
     format = 6.2;
     just = c;
   end;
   *5) the attributes below are for the TABLE definition, we will add one in the
       next example;
   required_space = needlines;
   print_headers = plabel;
   newpage = page;
   underline;
   use_name;
  end;
run;
```

As you can see the TABLE templates can be a little long (although they are nothing compared to STYLES.DEFAULT). However, editing a TABLE template is very simple because we only have to type in settings for the attributes we are going to change. So in our case we want to change the formats for the Frequency column and the Percent column to both be 5.2. This means we have merely to edit BASE.FREQ.ONEWAYFREQS and change the formats in those two columns. That can be done like so:

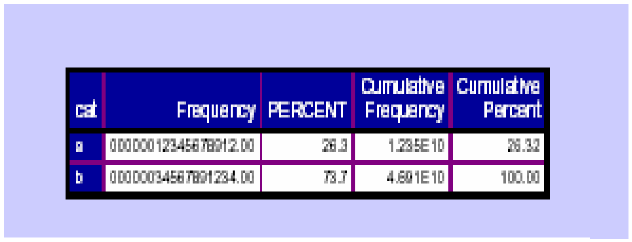
```
proc template;
edit base.freq.onewayfreqs;
define Frequency;
       header = "Frequency";
       just = c;
       format = 5.2;
       glue = gluef;
    end;
define Percent;
       header = ";Percent";
       just = c;
       format = 5.2;
       glue = gluef;
    end;
end;
run;
```

To make this more dynamic, we can set the formats to MACRO variable values:

```
proc template;
edit base.freq.onewayfreqs;
Mvar freqfmt perfmt; *The MVAR statement defines variables to be used as
Macro variables;
define Frequency;
       header = "Frequency";
       just = c;
       format = freqfmt;  *Notice, no Ampersand!;
       glue = gluef;
    end;
define Percent;
       header = ";Percent";
       just = c;
       format = perfmt;
       glue = gluef;
    end;
end;
run;
```

And then set these with %LET Statements before our PROC FREQ:

```
data test;
input cat $ oth $ count;
datalines;
a  c 12345678912
b  d 34567891234
;
run;

%LET freqfmt=Z20.2;  %LET PERfmt=best4.;
ods pdf file="C:\temp\freqy formats.pdf"  style=bordersNrules;
proc freq data=test;
tables cat;
weight count;
run;
ods pdf close;
```



cat	Frequency	PERCENT	Cumulative Frequency	Cumulative Percent
a	00000012345678912.00	26.3	1.235E10	26.32
b	00000034567891234.00	73.7	4.691E10	100.00

Figure 7: Changing the formats for columns calculated by PROC FREQ

As you can see, this allows you a great deal of flexibility in formatting the values of columns calculated by the procedure.

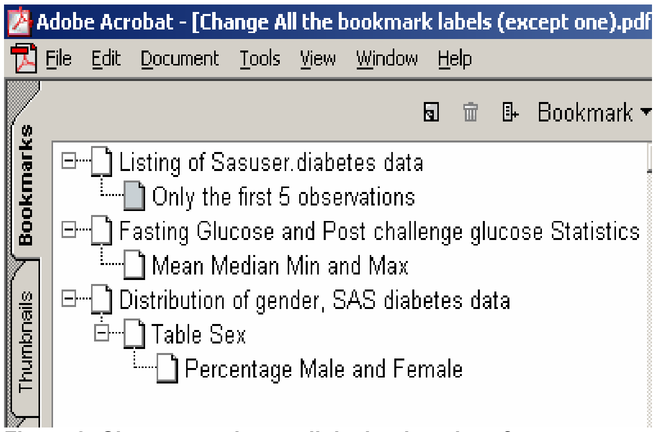
TABLE TEMPLATES: Modifying Bookmarks, PROC MEANS and PROC FREQ

This next quick example modifies an attribute of the TABLE itself (rather than an attribute of a specific column). We learned how to modify the colors of the TABLE of CONTENTS text using STYLE Templates, but we will now modify a TABLE template attribute to change the actual text that appears in the TOC. Again from our ODS TRACE information we know the templates associated with PROC MEANS and PROC FREQ are BASE.SUMMARY and BASE.FREQ.ONEWAYFREQS respectively. Once we have identified the TABLE TEMPLATES we wish to modify we can do so using PROC TEMPLATE. In this example I use the MACRO variable MYLABEL to set the CONTENTS_LABEL attribute for both TABLES, we will see why in a moment.

```
proc template;
edit Base.Freq.Onewayfreqs ;
     mvar mylabel;
     contents_label= mylabel;
end;

edit base.summary ;
    mvar  mylabel;
    contents_label = mylabel;
end;

run;
```

We can now use the MACRO variable MYLABEL to set the labels in our bookmarks.

```
ods pdf file="C:\Your Path\Change All the bookmark labels (except one).pdf"
style=mynewstyle;

ods proclabel "Listing of Sasuser.diabetes data";
proc print data=sasuser.diabetes (obs=5) contents="Only the first 5
observations";
var id age sex fastgluc;
run;
ods proclabel "Fasting Glucose and Post challenge glucose Statistics";
%let mylabel=Mean Median Min and Max;
Proc means data=sasuser.diabetes n mean median min max;
var fastgluc postgluc;
run;

ods proclabel "Distribution of gender, SAS diabetes data";
%let mylabel=Percentage Male and Female;
Proc freq data=sasuser.diabetes ;
tables sex;
run;
ods pdf close;
```


Adobe Acrobat - [Change All the bookmark labels (except one).pdf
File
Edit
Document
Tools
View
Window
Help
Bookmark
Bookmarks
Thumbnails

Listing of Sasuser.diabetes data
Only the first 5 observations
Fasting Glucose and Post challenge glucose Statistics
Mean Median Min and Max
Distribution of gender, SAS diabetes data
Table Sex
Percentage Male and Female

Figure 8: Changes to almost all the bookmarks, after modifying the corresponding TABLE TEMPLATES

Table Templates: Changing the STRUCTURE of an output table

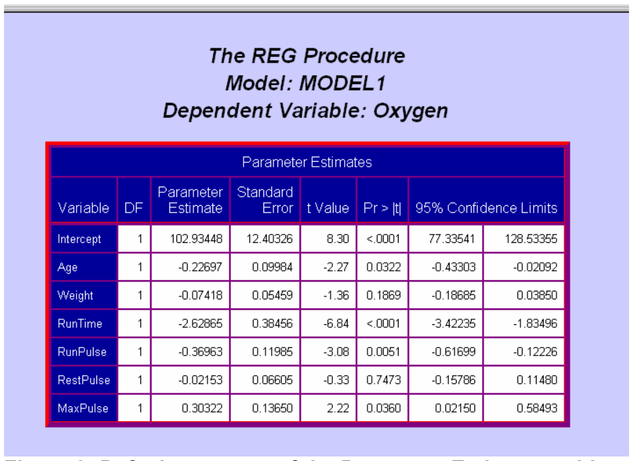
Until now we have still only really changed the formatting of our output, using both TABLE templates and STYLE templates to do so. In my last example, I will modify the structure of an output table by modifying its TABLE template. For this example I will use the PROC REG procedure, and the 'FITNESS' data set that SAS uses with this procedure. The data step to generate this dataset can be found with the PROC REG documentation.

First let's run a regression model on these data and take a look at the default parameter estimates table, to see where we are starting from:

```
ods select ParameterEstimates;
ods html file='C:\temp\Regbefore.html' style=bordersnrules;
proc reg data=fitness;
    model Oxygen=Age Weight RunTime RunPulse RestPulse MaxPulse/clb;
  run;
  quit;
ods html close;
```

Just in case you are not familiar with this code, the ODS SELECT statement tells SAS to send only the Paramenter Estimates table to the HTML file, and in PROC REG the CLB option on the MODEL statement gives the 95% confidence limits for the parameter estimates.

This code produces the output below after applying our Bordersnrules STYLE template:



The REG Procedure
Model: MODEL1
Dependent Variable: Oxygen

Parameter Estimates							
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|	95% Confidence Limits	
Intercept	1	102.93448	12.40326	8.30	<.0001	77.33541	128.53355
Age	1	-0.22697	0.09984	-2.27	0.0322	-0.43303	-0.02092
Weight	1	-0.07418	0.05459	-1.36	0.1869	-0.18685	0.03850
RunTime	1	-2.62865	0.38456	-6.84	<.0001	-3.42235	-1.83496
RunPulse	1	-0.36963	0.11985	-3.08	0.0051	-0.61699	-0.12226
RestPulse	1	-0.02153	0.06605	-0.33	0.7473	-0.15786	0.11480
MaxPulse	1	0.30322	0.13650	2.22	0.0360	0.02150	0.58493

Figure 9: Default structure of the Parameter Estimates table for PROC REG

This, I think, looks pretty good, but your boss isn't quite as convinced. He thinks the table is too wide, and would like you to:

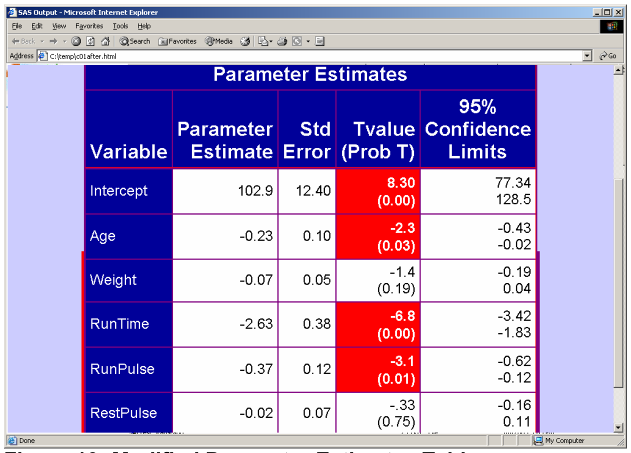
1) Stack the 95% Confidence Limits columns one on top of the other
2) Place the T Value over the top of the Probability of T (PROB >|t|) and change the header to Tvalue (Prob T)
3) Put Paraentheses around the probability value
4) Change the header of the Standard Error column to STD Error
5) Find a way to highlight the significant probability values
6) Get rid of some of those decimal places, no estimate has that much precision

Of course, after reading this paper, your reaction will be, what, that's it, come on, you can do better than that!!! ;-)

Let's look at the PROC TEMPLATE code that does all these things, and explain what's going on:

```
proc template;

edit stat.reg.parameterestimates; *1) Do you know how we got this?;
```

```
column variable Estimate StdErr tValue Probt (LowerCl UpperCL) ;
*2)  Columns we want in the table, the parens are one way to get  stacking;

          edit clhead;
          *3) Modify the Confidence limits Header to make it tighter
               The first character in a header is the split character, so
               We are using '#' to divide the header over 3 lines.
               Start and end set up a spanning header, not really necessary
               Since we will be stacking the columns, but lets leave it the way
               the default is set up;

          Text "#95%#Confidence#Limits";
          start = lowercl;
          end = uppercl;
          End;

          edit tvalue;
          header = "#T Value";
          merge; ; *4) Merges Tvalue column with the column to its right,
                    another  way to stack columns;

          format=4.2; *5) Gets rid of some decimal places;

          end;

          edit probt;
          header = "#Tvalue #(Prob T)";
          translate _val_ into "#("||put(_val_,4.2)||")";
          *6) Translate takes the VALue of PROBT and puts parens around it
          and a split character in front to push it to a new line when it merges
          with Tvalue;

          text_split="#"; *7) Sets the same Split for the cell value as for the
                         header;

          cellstyle _VAL_ le .05 as {background=red foreground=white f
                                   font_weight=bold};
          *8) The CELLSTYLE attribute allows VALue specific formatting of
          cells.  In this case we want to change the background of significant
          values to red and set their foreground to white;

          format= best8.; *9) This format needs to be longer to accommodate
                           the parentheses being added to the value;
          end;

          edit estimate;
          format = 5.2;
          end;

          edit stderr;
          header = "#Std#Error";
          format = 5.2;
          end;
          *10) Don't forget to fix the STD ERROR header and the other
          formats for the decimal places;

          edit uppercl;
           format=5.2;
           end;

          edit lowercl;
           format=5.2;
           end;
end;
run;
```


SAS Output - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\temp\c01after.html
Go
Done
My Computer

Parameter Estimates

Variable	Parameter Estimate	Std Error	Tvalue (Prob T)	95% Confidence Limits
Intercept	102.9	12.40	**8.30 (0.00)**	77.34 128.5
Age	-0.23	0.10	**-2.3 (0.03)**	-0.43 -0.02
Weight	-0.07	0.05	-1.4 (0.19)	-0.19 0.04
RunTime	-2.63	0.38	**-6.8 (0.00)**	-3.42 -1.83
RunPulse	-0.37	0.12	**-3.1 (0.01)**	-0.62 -0.12
RestPulse	-0.02	0.07	-.33 (0.75)	-0.16 0.11

Figure 10: Modified Parameter Estimates Table

Conclusions

I hope this interview with PROC TEMPLATE has taught you a little more about him, and let you see that he is really not that bad of a guy.   Sure he ma seem scary and intimidating at first, and he may not make a whole lot of sense, but when you get to know him, I am pretty sure you will really like him!! Be patient with him and, like most other friendly SAS procedures, he will open up to you. I trust that you will become very good friends. Feel free to contact me if you need a relationship counselor at any point along the way!!

References

SAS Institute, Inc. (1999) The complete guide to the SAS Output Delivery System, Version 8. Cary, NC. SAS Institute, Inc. Particularly the Chapter on PROC TEMPLATE

SAS Institute, Inc. (2002) Advanced Output Delivery System Topics Course Notes. SAS Institute, Inc. Cary, NC

Schellenberger, Brian (2002) V8+ ODS PRINTER FAQ http://www.sas.com/rnd/base/topics/odsprinter/faq.html (December, 2002)

Haworth, Lauren SAS ® with Style: Creating your own Style Template. *Proceedings of the Twenty-Seventh Annual SAS Users Group Conference.* (March 2002). http://www2.sas.com/proceedings/sugi27/p186-27.pdf (December 2002)

Contact Information:

Kevin P. Delaney MPH
Northrop-Grumman Mission Systems
Atlanta, GA
KDelaney@cdc.gov