Paper 190

Getting Stylish with Version 7 Base Reporting

David Kelley and Sandy McNeill, SAS Institute Inc., Cary, NC

Abstract

With Version 7 of the SAS System, you can exploit Output Delivery System (ODS) styles to produce HyperText Markup Language (HTML) summary and detail reports ready for Web publishing. This paper shows how you can customize the stylistic appearance of your reports using extensions to PROC REPORT and PROC TABULATE that are specific to ODS.

Introduction

The BASE SAS REPORT and TABULATE procedures are mainstays of the SAS report writer's toolkit. For Version 7, these procedures have been tightly integrated with ODS. You can manipulate ODS styles directly from the PROC REPORT and PROC TABULATE languages, thanks to new extensions. This paper will show you how to use these extensions to create eye-catching and informative reports, with emphasis given to these techniques:

- Hyperlinking – for connecting reports to related resources.
- Traffic lighting – for representing acceptance criteria for data.
- Graphics – for producing snazzier reports.

Styles

Note: For a comprehensive treatment of ODS and styles see *The Complete Guide to the SAS Output Delivery System*.

ODS styles govern the overall look and feel of Version 7 SAS procedure output. Styles determine the colors, fonts, graphic images, and other visual aspects in effect when output is generated. SAS delivers several predefined styles with Version 7, and you can customize them or create new ones.

Exactly one style is in effect at any time during a SAS session. It applies to all procedure output during the period it is in effect. Such output has a uniform look and feel – regardless of the procedure, row headers have the same background color, data cells have the same font, etc.

ODS organizes a style as a named collection of *style elements*. Specifically:

- Style names tell ODS which one of the available styles to apply to the output. If you do not specify a style name, ODS selects the predefined style named *Default*. The selected style remains in effect until you select another one or terminate SAS. You can rename or delete styles. (Be warned that if you rename or delete a predefined style, you could render SAS inoperable.)

- Style elements govern the look and feel of a particular part of the output, such as a PROC REPORT data cell or a PROC TABULATE row header. Style elements have names (*Data*, *RowHeader*) that are fixed by ODS; style elements may not be renamed or deleted.

- ODS organizes style elements as a collection of name-value pairs called *style attributes*. Each attribute determines a particular aspect of the style element, such as the row header background color (background=green) or data cell font size (font_size=4). Style attribute names are fixed by ODS. The domain of style attribute values depends upon the attribute.

Although referenced within ODS, styles are defined outside of ODS using a new Version 7 procedure, PROC TEMPLATE. Refer to Christopher Olinger's paper "Twisty Turny Passages, All Alike – PROC TEMPLATE Exposed" in these conference proceedings for more information about PROC TEMPLATE.

Styles apply to all of the ODS output formats that allow colors, proportional fonts, etc. Although this paper focuses on HTML, the techniques presented apply also to Rich Text Format (RTF), LaTeX, PostScript, Extensible Markup Language (XML) or any other "stylish" format supported now or in the future.

How PROC REPORT and PROC TABULATE Use Styles

PROC REPORT and PROC TABULATE use certain style elements to control the default look and feel of various report regions. The attributes for the style element are supplied along with the report data generated by the procedures to ODS, which then produces HTML output for a particular region.

Here are the default style elements for PROC TABULATE reports:

Table 1

PROC TABULATE Region	Style Element
Table (borders, rules)	Table
Caption	Caption
Column headers	Header
Box above row headers	Header
Row headers	RowHeader
Data cells	Data

Here are the default style elements for PROC REPORT reports:

Table 2

PROC REPORT Region	Style Element
Table (borders, rules)	Table
Column headers and spanning headers	Header
Data cells	Data
Summaries	Data-Emphasis
Lines	TableNote-ContentCell

If you are a motivated report writer, you can learn enough about PROC TEMPLATE to customize the appropriate style elements in a predefined style or one that you create. Then you can tell ODS to use the style for PROC REPORT or PROC TABULATE output. This approach has several benefits:

- You get personalized reports.
- You can apply the new presentation to all of your reports in order to create your own signature look and feel.
- You do not have to modify your PROC REPORT or PROC TABULATE code.

However, an approach based on PROC TEMPLATE has drawbacks as well:

- You have to learn a new procedure.
- It is inconvenient to create "one-off" reports, i.e. reports that have a one-time or highly specific application.
- There is no way to customize the appearance of report regions for which no corresponding style elements are defined. An example of this is the data cells governed by a PROC TABULATE statement crossing – with styles you can personalize the appearance of all PROC TABULATE data cells in a report, but not a subset of them. But what if you want to highlight row or column totals? PROC TEMPLATE cannot do that.
- You cannot read your PROC REPORT or PROC TABULATE code and know how it will render in HTML The content of the report is divorced from the presentation of the content.

We believe that you want to be able to control the presentation of your Version 7 PROC REPORT and PROC TABULATE reports using the procedures themselves. Consequently we have enhanced these procedures to permit you to specify style elements and/or style attributes for report regions. This is accomplished with the new STYLE= option.

The STYLE= Option

The STYLE= option has the following syntax:

STYLE<**(***region (s)***)**>**=**<*element*><**{***attribute(s)***}**>

Note: You can use square brackets ([and]) instead of braces ({ and }).

region
: identifies the region of the report that the STYLE option affects.

element
: is the name of the style element. The default is the default style element for the region. (See Tables 1 and 2.)

attribute
: is the style attribute to be customized.

Note that you can customize only regions within the report. If for instance you desire to change the font of procedure titles, you will have to use PROC TEMPLATE. The STYLE= option is not a replacement for PROC TEMPLATE, but an extension to it.

PROC REPORT and PROC TABULATE HTML Examples

Note: Due to space limitations, the output generated by ODS for the examples discussed here has been omitted from the hardcopy of these proceedings. The online version of this paper includes the output. See the proceedings CD-ROM, or visit the BASE SAS Software Research and Development Web site, http://www.sas.com/rnd/base/.

The rest of this paper is devoted to examples that demonstrate the use of the STYLE= option, as well as procedure-specific enhancements, to create striking presentation effects in HTML reports. A helpful way to start is to illustrate the various regions of a PROC TABULATE report that may be customized with styles. Let's begin by defining a simplistic data set named *sales* that will serve as input to example sales reports we create. The data set defines these variables:

Region
: is the sales region (state).

CitySize
: is the size (Small, Medium or Large) of a city in the sales region.

Population
: is the population of a city in the sales region.

Product
: is the code of a product being sold.

SaleType
: is the type of sale (Wholesale or Retail).

Units
: is the number of units of a product sold.

NetSales
: is the net sales of a product.

```
data sales;
  input Region $ CitySize $ Population
    Product $ SaleType $ Units NetSales;
  cards;
   NC S   25000 A100 R 150   3750.00
   TX S   37000 A100 R 200   5000.00
   NC M  125000 A100 R 350   8750.00
   TX M  237000 A100 R 600  15000.00
   NC L  837000 A100 R 800  20000.00
   TX L  748000 A100 R 760  19000.00
   NC S   25000 A100 W 150   3000.00
   TX S   37000 A100 W 200   4000.00
   NC M  125000 A100 W 350   7000.00
   TX M  237000 A100 W 600  12000.00
   NC L  625000 A100 W 750  15000.00
   NE L  837000 A100 W 800  16000.00
;
```

Notice that we take advantage of Version 7 naming features. Using mixed-case names yields more readable default variable headers in our reports. While we are it, let's define a format for the *SaleType* variable so that its class variable level value report headers will be more readable as well:

```
proc format;
   value $salefmt 'R'='Retail'
                  'W'='Wholesale';
```

PROC TABULATE Report Regions Affected By Styles

Using the *sales* data set and the *salefmt* format let's create code to identify regions of a TABULATE report affected by styles. The way we will do this is to customize the foreground color of text in the various regions. But before we can begin doing that, we must tell ODS to generate HTML, since it generates only standard listing output by default:

```
ods html file="tabregions.html";
```

Since we did not specify a style for the HTML output, ODS will use the style *Default,* which has gray backgrounds and blue foregrounds.

Now we are ready to review the PROC TABULATE statements:

```
proc tabulate data=sales
            style={foreground=very dark green};
   class product /
            style={foreground=orange};
   class region citysize saletype /
            style={foreground=blue};
   format saletype $salefmt.;
   classlev region citysize saletype /
            style={foreground=yellow};
   var units netsales /
            style={foreground=black};
   label units="Units Sold" netsales="Net Sales";
   keyword all sum /
            style={foreground=white};
   keylabel all="Total";

   table product,
         (region all)*(citysize
            all*{style={foreground=#002288}}),
         (saletype all)*(units*f=COMMA6.
                         netsales*f=dollar10.) /
      style={background=red}
      misstext={label="Missing"
                style={foreground=brown}}
      box={label="Region by Citysize by Saletype"
           style={foreground=purple}};
run;
```

There is a lot of new Version 7 content in that code, so let's break it down into manageable chunks, beginning with the PROC TABULATE statement:

```
proc tabulate data=sales
            style={foreground=very dark green};
```

That STYLE= option sets the default foreground color of all data cells to be "very dark green". So what exactly is "very dark green", other than darker then "dark green" presumably? Well "very dark green" is a SAS/GRAPH® software color name defined to be the RGB (Red-Green-Blue) value of 00-80-00 hexadecimal. Of course "very dark green" is more meaningful than 0080000 to most folks, which is why ODS supports color names. However you can specify an RGB value (using the syntax CV008000 or #008000) if you like.

Why does specifying STYLE= on the PROC TABULATE statement affect data cells? Because TABULATE strives to be consistent in its application of styles and formats, and specifying FORMAT= on the PROC TABULATE statement affects data cells.

Since the default style element for data cells is *Data*, the verbose form of the PROC TABULATE statement is:

```
proc tabulate data=sales
            style=data{foreground=very dark
                       green};
```

Moving on, it is time to declare some class variables:

```
class product /
            style={foreground=orange};
class region citysize saletype /
            style={foreground=blue};
format saletype $salefmt.;
```

With Version 7 PROC TABULATE, you can specify multiple CLASS (and VAR) statements. That is handy, because we want to set blue as the foreground color of headers of all class variables except *Product.*

Besides customizing class variable headings, you can customize class variable level value headers. Version 7 PROC TABULATE introduces a new statement, CLASSLEV, to do that:

```
classlev region citysize saletype /
         style={foreground=yellow};
```

Next we define the analysis variables, giving their headers a black foreground:

```
var units netsales /
           style={foreground=black};
label units="Units Sold"
      netsales="Net Sales";
```

To enable you to customize keyword (statistic and ALL) headers, Version 7 PROC TABULATE has the new KEYWORD statement:

```
keyword all sum /
         style={foreground=white};
```

So far we have been customizing variables and keywords apart from their use by a TABLE statement. These style assignments apply to all TABLE statements defined by the procedure. However PROC TABULATE permits you to customize variables and keywords in the TABLE statement as well. Such customizations take precedence over the CLASS, CLASSLEV and KEYWORD style settings.

Now let's scrutinize each use of STYLE= in the TABLE statement, beginning with the row dimension subexpression:

```
all*{style={foreground=#002288}}
```

Remember that the asterisk operator serves two purposes: nesting (a*b) and attribution (a*format=10.2). Just as you can set a format on the data cells governed by a crossing, you can set a style likewise. The above code highlights row total data cells by setting their foreground color to RGB 002288, which is a dark bluish color. Using styles to highlight total rows (and columns) is a powerful technique for report writers.

You might wonder why the code is not:

```
all*style={foreground=#002288}
```

That is because that syntax invites ambiguities. Consider:

```
all*style=MyCellStyle
```

This could mean two different things:

1. Set the style of total data cells to *MyCellStyle*.
2. Assign the label "MyCellStyle" to the variable or keyword named *style.* This is the legacy behavior. It is also undocumented behavior, so refrain from doing it ☺.

That is why PROC TABULATE insists on enclosing braces

(or brackets) if you want to set the style of data cells.

The next TABLE statement bit:

```
style={background=red}
```

shows the use of the TABLE statement STYLE= option. Using the option in this context affects the report as a whole. Things such as table borders and rules can be modified. In this example, the background color is set to red, which causes the table rules to be red.

Next is:

```
misstext={label="Missing"
          style={foreground=brown}}
```

This code sets the text of data cells with missing values to "Missing", and sets the color of the text to brown. Here again we use enclosing braces, but now they are a virtue rather than a necessity, since they allow the independent setting of the label and the style. The same syntax works for customizing the box above the row headers:

```
box={label="Region by Citysize by Saletype"
     style={foreground=purple}}
```

We are done with PROC TABULATE, so we must tell ODS to stop generating HTML:

```
ods html close;
```

That is it for the example code. Figure 1 shows the HTML generated from it.

Embedding Hyperlinks in a PROC TABULATE Report

Hypertext is what HTML is all about. The ability to link documents across a company intranet or the Internet is the compelling feature that motivated the ODS team to choose HTML as the first new production-quality output format for Version 7.

Unlike PROC REPORT, PROC TABULATE does not have special syntax to support hypertext. However if you know a modicum of HTML you can make PROC TABULATE speak hypertext. That is because HTML is a text markup language, and SAS speaks text fluently. For instance you can easily embed a hyperlink into a PROC TABULATE report, as the next example demonstrates.

Imagine that we would like to create another sales report using the *Sales* data set, but this time we want to link each sales region to some dedicated Web page about the region. We take advantage of the fact that by default ODS passes through any well formed (i.e. angle bracket enclosed) HTML to the output without interpretation. We define a new user format value called *regfmt* that associates each region with the URL (Uniform Resource Locator) of the associated Web page:

```
proc format;
   value $salefmt 'R'='Retail'
                  'W'='Wholesale';
   value $regfmt  'NC'='<A
     HREF="http://www.somewhere.com/nc.html">North
     <BR>Carolina</A>'
                  'TX'='<A
     HREF="http://www.somewhere.com/tx.html">Texas
     </A>';
```

You might wonder about the "
" between "North" and "Carolina" in the NC region URL. It is a pretty-printing trick. Typically HTML is viewed with a Web browser, which renders to an infinite plane – tables never break across pages, but rather scroll horizontally and vertically. Since browsers do not need to conserve horizontal space, they will not wrap text unless you tell them to do so. "
" is an HTML directive that breaks the text at the designated spot.

As before we instruct ODS to generate HTML prior to running PROC TABULATE. But there is a twist this time – we specify the predefined *D3D* style using the ODS STYLE= option:

```
ods html file="tablinks.html" style=d3d;
```

Instead of the default gray and blue color scheme, we will get reverse video headers and nifty 3D borders.

Do not confuse the ODS STYLE= option with the PROC TABULATE STYLE= option. The former option controls the overall look and feel of the report, whereas the latter one controls the look and feel of a particular region of the report.

Here is the PROC TABULATE code:

```
proc tabulate data=sales format=dollar10.;
   class product region saletype;
   format saletype $salefmt.;
   format region $regfmt.;
   classlev region saletype /
            style={font_style=italic};
   var netsales;
   label netsales="Net Sales";
   keylabel all="Total";

   table product,
    region
    all*{style=<parent>{foreground=yellow
                        }},
    (saletype=" " all)*netsales*
       (sum median) /
    misstext="Missing"
    box="<DIV>Region by<BR>Saletype</DIV>"
    indent=0;
run;

ods html close;
```

Besides defining hyperlinks, this code performs some other customizations:

- The text of class variable level value headings is italicized.
- Row total cells are highlighted. This is done using the PROC TABULATE-specific "<parent>" syntax, which in this case forces the row total cells to have the same style as the ALL row header, except that the foreground color of the cells is set to yellow.
- The "
" trick is repeated to force the box text to wrap. The "
" is specified inside do-nothing "<DIV>" and </DIV>" directives in order to satisfy ODS' requirement for well formed HTML

Figure 2 shows the HTML output that ODS generates for this example.

A potential drawback to adding lots of raw HTML to your PROC TABULATE reports is that they become biased to a particular output destination. This is not a problem if your reports are designed solely for Web publishing. However if you print the above report to the SAS listing you get what is shown in Figure 3.

Using Traffic Lighting and Graphic Images in a PROC TABULATE Report

Traffic lighting describes a reporting technique in which the report data influence the presentation of the data. Traffic lighting serves to draw attention to relationships within the data which, due to their dynamic nature, cannot be expressed with traditional reporting idioms. It is well suited to scenarios in which the data can be categorized into ranges, with each range indicating a level of acceptance (e.g. Poor, Fair, and Excellent). The idea is to highlight the ranges with different colors (e.g. Red, Yellow, and Green).

For Version 7 both PROC REPORT and PROC TABULATE support traffic lighting, but in different ways. Since PROC REPORT provides a robust DATA STEP-like programming language, you specify the data ranges with PROC REPORT syntax. PROC TABULATE does not have such a capability, and instead requires the user to specify the ranges with PROC FORMAT.

The next example shows how to use PROC FORMAT in conjunction with PROC TABULATE's STYLE= option to do traffic lighting of net sales figures. The same scheme is employed to display catchy graphic images in the row headers.

The first order of business is to define the PROC FORMAT ranges for the report data we want to highlight. Here is the PROC FORMAT code, which includes the salefmt and regfmt formats from earlier examples:

```
proc format;
   value $salefmt  'R'='Retail'
                   'W'='Wholesale';

   value $regfmt   'NC'=
               '<DIV><BR>North Carolina</DIV>'
                   'TX'='<DIV><BR>Texas</DIV>';

   value bgfmt     0-17000     ='red'
                   17001-34000 ='yellow'
                   other       ='light green';

   value $giffmt   'NC'='ncflag.gif'
                   'TX'='txflag.gif';
```

The *bgfmt* format will be used to control the background color of certain data cells. The *giffmt* format will be used to control the graphic images for the *Region* class variable level value headers.

Next we enable ODS HTML. We will be using the predefined *Brick* style this time.

```
ods html file="tablighting.html" style=brick;
```

Here is the PROC TABULATE code:

```
proc tabulate data=sales format=dollar10.
              style={just=c
                     vjust=c
                     foreground=black
                     font_weight=bold};

   class product region saletype;
   format saletype $salefmt.;
   format region $regfmt.;
   classlev region /
            style={font_style=italic
                   preimage=$giffmt.};
   classlev saletype /
            style={font_style=italic};
   var netsales;
   label netsales="Net Sales";
   keylabel all="Total";

   table product={style={just=c
                         font_size=6}},
         region,
         (saletype=""*
            {style={background=bgfmt.
                    foreground=black
                    font_weight=bold
                    just=c
                    vjust=c}}
          all)*netsales*sum=" " /
      misstext="Missing"
      box="Region by Saletype"
      indent=0
      style={background=black};
run;
```

Let's go over in detail the "stylish" portions of the code, starting with the PROC TABULATE statement:

```
proc tabulate data=sales format=dollar10.
              style={just=c
                     vjust=c
                     foreground=black
                     font_weight=bold};
```

The STYLE= option above applies these customizations to report data cell text:

- Center justifies it horizontally and vertically.
- Colors it black.
- Bolds it.

The CLASSLEV statement is used to customize class variable level value headers:

```
classlev region /
            style={font_style=italic
                   preimage=$giffmt.};
classlev saletype /
            style={font_style=italic};
```

Of particular interest is the *preimage* style attribute, which tells ODS to generate HTML that will display a graphic image before the text for the header. Notice that *preimage* is bound to the *giffmt* format instead of a string literal. This is how PROC TABULATE leverages the formatting power of PROC FORMAT – the image that gets displayed in the header is determined by applying *giffmt* to each *Region* level value input from the *Sales* data set, and assigning the resulting formatted value to *preimage*.

Now take a look at the TABLE statement page dimension expression:

```
table product={style={just=c
                      font_size=6}}
```

This code center justifies and enlarges the font size of the

page dimension text (“Product A100”).

Here is the TABLE statement column dimension expression:

```
(saletype=""*
            {style={background=bgfmt.
                    foreground=black
                    font_weight=bold
                    just=c
                    vjust=c}}
          all)*netsales*sum=" "
```

The STYLE= option performs these customizations in this code:

- Assigns the *bgfmt* format *to* the *background* style attribute. This is how we create the traffic lighting effect – PROC TABULATE applies *bgfmt* to the values of the data cells governed by the crossing, and uses the formatted values as background colors for the cells.

- Center justifies cell text horizontally and vertically, gives it a black color, and bolds it. Since a STYLE= assignment for a TABLE statement dimension expression takes precedence over one for the PROC TABULATE statement, we have to repeat the cell text customizations in this code if we want cell text to have a uniform appearance.

You should be mindful that PROC TABULATE does not verify the sanity of style attribute values. If you specify a foreground color of “HelloDolly”, PROC TABULATE will blithely pass it through to the HTML output, resulting in unpredictable browser behavior. Another potential landmine to avoid is a style attribute format that does not define all data values that PROC TABULATE might supply it. Often this happens because the format definition omits the catchall “other” category. If your report has data that fall outside of the format, PROC TABULATE will not know what to do with it, and will generate ill formed HTML that likely will confuse your browser.

The only style customization in the remaining code is to color the table rules black:

```
/
      misstext="Missing"
      box="Region by Saletype"
      indent=0
      style={background=black};
run;

ods html close;
```

Figure 4 shows the HTML generated for the code in this example.

PROC REPORT

Now that we have an understanding about how PROC TABULATE handles styles, let’s take a look at the same types of tasks using PROC REPORT.

PROC REPORT Regions Affected by Styles

As we start going through some examples, you’ll see that the syntax for styles in PROC REPORT is just slightly different from PROC TABULATE in that it uses location identifiers to represent the different regions in the report that the style should modify.

Using the same SALES data set as described earlier, let’s create a report which shows the different regions of the report modifiable by styles by changing the foreground color of the text in the different regions. First, however, we need to instruct ODS to generate HTML.

```
ods html file=”Repregions.html”;
```

Here’s the PROC REPORT code:

```
proc report nowd
     style(REPORT)={background=red}
     style(HEADER)={foreground=blue}
     style(COLUMN)={foreground=green}
     style(LINES) ={foreground=black}
     style(SUMMARY)={foreground=brown};

column region citysize saletype units
netsales;
define region   / group ‘Region’
     style(COLUMN)={foreground=purple);
define citysize / group ‘Citysize’;
define saletype / group ‘Saletype’
            format=$salefmt. ;
define units    / sum format=comma6. ;
define netsales / sum ‘Net Sales’
     format=dollar10.
     style(HEADER)={foreground=very dark
                    green};

rbreak after / summarize;

compute before _page_ /
     style={foreground=deep pink
            background=oldlace} LEFT;
     line ‘Retail and Wholesale Amounts
            per Region’;
endcomp;

break after region / summarize
     style(SUMMARY)={foreground=orange};

compute after;
     line ‘Total Sales for All Regions ‘
            netsales.sum dollar11.2 ;
endcomp;

run;

ods html close;
```

As you can see in the report statements, the style syntax is slightly different from the PROC TABULATE syntax in that PROC REPORT uses location identifiers in the parentheses to instruct which region of the report the style should modify.

```
      style(REPORT)={background=red}
      style(HEADER)={foreground=blue}
```

These location identifiers must be used when you put a style option on the PROC REPORT statement, but they are optional (and actually defaulted) on the other PROC REPORT statements that allow the style option.

```
compute before _page_ /
     style={foreground=deep pink
            background=oldlace} LEFT;
```

This COMPUTE block statement with a style option is an

example of a style option with no location identifier. Because only one style identifier, LINES, is valid on the COMPUTE statement (see Tables 3 and 4), there is no need to specifically type in the location identifier. It is defaulted and assumed to be LINES. We are going to discuss these location identifiers a little bit more, and refer to Tables 3 and 4 for a reference of which location identifiers are allowed on which PROC REPORT statements.

The valid PROC REPORT style location identifiers are:

- CALLDEF
- COLUMN
- HEADER
- LINES
- REPORT
- SUMMARY

These location identifiers are mandatory on the PROC statement because otherwise PROC REPORT would not know which style should modify which region. On the other report statements, the use of these locations is restricted (and usually defaulted). However, for use in the statements that are associated with the particular location identifiers, most of the uses of the location identifiers are optional (since it defaults depending on which statement the style is located in) as in the example of the COMPUTE block earlier.

Here's a table that shows the association among the location identifiers, the PROC REPORT statements in which these identifiers are valid, and the PROC REPORT statement in which the particular location identifier is defaulted:

Table 3

Location Identifier	Can be used in these statements	Is the default in this statement
Calldef	Report Calldef	Calldef
Report	Report	
Column	Report Define	Define
Header	Report Define	Define
Lines	Report Compute	Compute
Summary	Report Rbreak Break	Rbreak Break

So now that we know which location identifiers can be used in which PROC REPORT statements, here's another table that shows which region of the report is affected when the style is used with a particular location identifier in a particular PROC REPORT statement:

Table 4

Location Identifier	Used in this state-ment	Affects this region of the report
Calldef	Report	All cells that are identified by any CALL DEFINE statement in the report
Calldef	Calldef	Just the cells particular to that CALL DEFINE statement
Report	Report	The structural part of the report; that is, the underlying table. Use the REPORT location identifier to set things such as the width of the border and the space between cells.
Column	Report	The data cells of the entire table
Column	Define	Just the cells of the particular column governed by the define
Header	Report	All column headers for the entire table
Header	Define	Just the column header of the column governed by the define
Lines	Report	All lines in the report generated by line statements
Lines	Compute	Just the line statements in the COMPUTE block to which the style is attached
Summary	Report	All default summary lines for the entire table
Summary	Rbreak Break	Just the default summary lines associated with the particular Break or Rbreak to which the style is attached

Now that we have our definitions of what the different regions of the report are that the location identifiers modify, let's go back to the example and discuss some of this.

```
proc report nowd
     style(REPORT)={background=red}
     style(HEADER)={foreground=blue}
     style(COLUMN)={foreground=green}
     style(LINES) ={foreground=black};
```

When a style is placed in the report statement, the regions that will be affected are in a broader sense. So for example,

```
style(COLUMN) = {foreground=green}
```

in the PROC REPORT statement above will change the foreground of **all** the data cells in the report to green. The style option

```
style(HEADER) ={foreground=blue}
```

will change the foreground of **all** the column headers to blue. Now contrast this with the style option and the COLUMN or HEADER location identifier being used on the DEFINE statement.

```
define region   / group 'Region'
     style(COLUMN)={foreground=purple);
```

So what's the difference between placing the *STYLE(COLUMN*) on the report statement compared to placing it as shown here in the DEFINE statement? When the *STYLE(COLUMN)* is used on a define statement, only the cells for that particular column will have their foreground set to purple. And what happens to the *style(COLUMN)* on the report statement? This will change the foreground to

green of all the data cells in the report that are **not** affected by a *STYLE(COLUMN)* on a DEFINE statement. In this case, all the data cells in the table will have a foreground of green except for the data cells in the *region* column which will have a foreground of purple.

What about styles for a particular region being set from both the PROC REPORT statement and from a different statement? Styles set from statements other than the PROC REPORT statement will augment the style for a particular region that may have been set on the PROC REPORT statement. Here's what I mean:

```
define region   / group 'Region'
     style(COLUMN)={font_weight=bold);
```

This DEFINE statement shows the style option will bold the data cells of the column *region*. In addition to the bolding, we also look up to the PROC REPORT statement and see if any style was globally set for the data cells for the entire report.

```
proc report nowd
     style(COLUMN)={foreground=green}
```

On the PROC REPORT statement, there's a style that will set the text color (foreground) of all the data cells in the report to green. Since the style on the DEFINE statement for *region* sets the font_weight and not the foreground, we have two different style attributes that are being set. Since two separate style options are modifying the same report region, the style on the DEFINE statement for *region* will add to the other style thus giving us the data cells in the column *region* bolded with a text color of green.

It is also legal to put more than one location identifier within the parentheses (separated by a space). The advantage to this is that it reduces the number of style commands if you had a particular style you wanted to use for two or more regions of the report. As always, we programmers are always looking for shortcuts and ways to get out of typing. ☺ So say, for example, that you wanted all lines produced from LINE statements to have blue text and to be bolded, and you also wanted to have all default summaries to have blue text and to be bolded. One way to accomplish this is to have two separate style commands:

```
proc report nowd
     style(LINES)=  {foreground=blue
                     font_weight=bold}
     style(SUMMARY)={foreground=blue
                     font_weight=bold}
```

or you can combine these two style commands into one command:

```
proc report nowd
     style(LINES SUMMARY)={foreground=blue
                           font_weight=bold}
```

Another thing you may be asking is if the COLUMN location identifier changes the data cells of the table, what region does the REPORT location identifier change? Using the style option with the REPORT location identifier affects the report as a whole. Things such as table border colors and border widths can be modified. In the style option:

```
style(REPORT)={background=red}
```

the background color is set to red which causes the table rules to be red.

Now let's move on down the example report and discuss the LINES location identifier. Here are the snippets of the report that we will be discussing.

```
proc report nowd
     style(LINES) ={foreground=black};

compute before _page_ /
     style={foreground=deep pink
            background=oldlace} LEFT;
     line 'Retail and Wholesale Amounts
            per Region';
endcomp;

compute after;
     line 'Total Sales for All Regions '
            netsales.sum dollar11.2 ;
endcomp;
```

The LINES location identifier is used when you want to modify the style of lines produced by LINE statements in a compute block. If the LINES identifier is used on the REPORT statement, every line generated by a LINE statement in the report will be affected. If you want to modify lines from a particular compute block, the style option should be placed on the compute block that contains the lines that are to be affected as in the COMPUTE BEFORE _PAGE_ snippet of code above. The line "Retail and Wholesale Amounts per Region" will have a text color of deep pink with a background color of oldlace (which is a fancy name for off-white). The COMPUTE AFTER block has no style command local to it, but it will get its style from the STYLE(LINES) command on the PROC REPORT statement, so its text will have a color of black.

The last report region to discuss in this section is the region affected by the SUMMARY location identifier. This location identifier is used to modify all or some default summary lines.

```
break after region / summarize
     style(SUMMARY)={foreground=orange};
```

The style option shown here on the BREAK AFTER REGION will change the text of the default summary line to an orange color.

```
rbreak after / summarize;
```

The default summary generated from the RBREAK AFTER statement will inherit its style from the global style for SUMMARY lines that was set on the PROC REPORT statement.

```
proc report nowd
     style(SUMMARY)={foreground=brown}
```

One thing to keep in mind regarding the style option and the placement of the style option is that the style option used on the PROC REPORT statement will affect **all** the regions in the report according to which location identifier is used. However, a style option used on any other statement only affects that particular column, header, summary, etc. for which the style option is attached. Furthermore, if the same attribute is set via a style option in both the PROC REPORT statement and any one of the other valid statements, the style attribute from the localized PROC REPORT statement

(such as the DEFINE, COMPUTE, BREAK or RBREAK) will take precedence.

Embedding Hyperlinks in PROC REPORT

As mentioned in the PROC TABULATE section of this paper, hypertext is what HTML is all about. The ability and power to link from one report to another can be a very powerful presentation tool. Most reports are usually too busy – too much detail crammed in a report. The result of busy reports is that the reader won't grasp what was intended by the report. Busy reports are usually hard to read and hard to understand.

Placing hypertext in a cell in PROC REPORT is accomplished by using the CALL DEFINE statement within a compute block. Let's go through an example and I'll show you how.

Once again, we instruct ODS to generate HTML prior to running PROC REPORT.

```
ODS HTML file="replinks.html";
```

Here's the PROC REPORT code:

```
proc report nowd
     style(REPORT)={foreground=red}
     style(HEADER)={foreground=blue}
     style(COLUMN)={foreground=green}
     style(LINES) ={foreground=black};

column Region saletype netsales;
define region   / group 'Region'
       style(COLUMN)={foreground=purple};
define citysize / group 'Citysize';
define saletype / group 'Saletype'
            format=$salefmt.;
define netsales / sum 'Net Sales'
            format=dollar10.
            style(HEADER)=
                   {foreground=very dark green
                    font_weight=bold};

compute before _page_ /
            style={foreground=deep pink
                   background=oldlace} LEFT;
     line 'Retail and Wholesale Amounts per
            Region';
endcomp;

compute before;
     numregion = 1;
endcomp;

compute after region;
     numregion = numregion + 1;
endcomp;

break after region / summarize
            style(SUMMARY)={foreground=white
                            font_weight=bold};

compute region;
     urlstring = "details.html#region" ||
            left(put(numregion,3.0));
     call define (_COL_, 'URL', urlstring );
     call define (_COL_, 'style',
            'style={flyover="Click to go to
                   details for the region"}');
endcomp;

compute after;
     line 'Total Sales for All Regions:  '
            netsales.sum dollar11.2;
endcomp;

run;

ods html close;
```

This is a short summary report with links to a detailed report, so if you want to see more information about the 'NC' region, click on the link and you'll be taken to the section of the detailed report containing the 'NC' region information. The detailed report that this summary report is linked to is a single report containing three separate tables of information – one table for each region. Each of the tables in the detailed report has an HTML anchor associated with it. This anchor tag is the string after the '#' in the *URLSTRING* in the example above. Each *region* is counted and assigned a unique number in the report, and then this number is concatenated to the anchor tag. The urlstrings for the three different regions are:

```
details.html#region0
details.html#region1
details.html#region2
```

Another way the detail information could have been organized would have been to place each region into a separate report. This is easily accomplished by the use of the "newfile=table" syntax on the ODS statement:

```
ODS HTML file="region1.html"
newfile=table;
```

What this does is to place the first region (which in our example is 'NC') into "REGION1.HTML". The second region, 'NE', will be placed into "REGION2.HTML", and the third, and last, region 'TX' will be placed into "REGION3.HTML". The CALL DEFINE in the summary report can then be modified to be:

```
compute region;
     urlstring = "region" ||
            left(put(numregion,3.0)) ||
            ".html";
     call define (_COL_,'URL',urlstring);
     call define (_COL_, 'style',
            'style={flyover="Click to go
                   to details for the
                   region"}');
endcomp;
```

The special attribute name "URL" (the second parameter of the CALL DEFINE statement) is what is used to signal to PROC REPORT to use the URL string (the third parameter of the CALL DEFINE statement) as the address to link to. The data in the cell is changed from being just text to now being a hyperlink.

Besides performing hyperlinks, this example code performs a few other style customizations:

- The text for the detault summary line for each region has been bolded and the color of the text is white.
- The data cells for the column *region* have a text color of purple.
- The column header for the column *netsales* has been

bolded and the color of the text is “very dark green”.

- The data cell for REGION has a “flyover” associated with it. This is a nifty trick of attaching a textual string to a cell. To see this effect, just hover over the cell for a few seconds with the mouse.

Using Traffic Lighting in PROC REPORT

As mentioned in the PROC TABULATE section, traffic lighting is used to draw attention to values in a report that deserve specific attention – values above or below a certain threshold, values in a range, or even the change in values from one row to the next that would otherwise be overlooked.

Since PROC REPORT has the power of the SAS DATA STEP within the compute blocks, it uses this scheme of employing traffic lighting instead of the format scheme that PROC TABULATE uses. The power of DATA STEP traffic lighting is that the decision and the traffic lighting can be performed dynamically. An example of this dynamic nature is if we wanted to highlight sales numbers that were above the average of the sales in the report. First, though, we need to make sure to direct our output to HTML.

```
ODS HTML file=”rephilight.html”;
```

Here’s the PROC REPORT traffic lighting example:

```
proc report data=sales ls=100 nowd;

column product region saletype netsales
     netsales=netmean comments;

define product  / group;
define region   / group;
define saletype / group;
define netsales / analysis sum format=dollar11.2;
define netmean  / mean noprint;
define comments / width=30 computed "Comments";

compute before;
     totalsales = netsales.sum;
endcomp;


compute after region /
        style(LINES)={font_style=italic
                      background=oldlace
                      foreground=blue};
     length txt $35.;
     if netsales.sum > netmean then
        txt = "Well Done! Sales above average!";
     else if netsales.sum < netmean then
        txt = "Keep working -- Sales below
     average";
     else if netsales.sum = . then
        txt = "Sales figures are missing";
     else
        txt = "Keep up the good work ";
     line region $3. ’Region -- ’ txt $35.;
endcomp;

compute comments / character length=30;
     if netsales.sum = . then
     do;
        comments = ’Sales figures missing’;
        call define (_COL_, ’style’,
             ‘style={foreground=cranberry}');
        call define ('netsales.sum', 'style',
             'style={foreground=cranberry}');
     end;

     else if netsales.sum < netmean then
     do;
        comments = "Sales below average";
        call define (_COL_, 'style',
             'style={foreground=red

             background=oldlace}');
         call define ('netsales.sum',
'style',
             ‘style={background=red
                    foreground=oldlace}');
     end;

     else if netsales.sum >= netmean then
     do;
        comments = "Sales above average";
        call define (_COL_, 'style',
             'style={foreground=green

             background=oldlace}');
        call define ('netsales.sum',
     'style',
             'style={background=green
                    foreground=oldlace}');
     end;

endcomp;

run;

ods html close;
```

In this example, we are testing the *NETSALES* to see if they are below or above the MEAN of *NETSALES*. We then place a comment in the *COMMENT* column and depending on whether the value was above or below average, we traffic light the *COMMENT* data cell and the corresponding *NETSALES* data cell. Using the power of the DATA STEP within the COMPUTE block, we are able to use the average of the actual data to help determine what color we should use for traffic lighting. Contrast this to the formatting scheme employed by PROC TABULATE where the range of values must be decided upon in advance. Using the dynamic powers of PROC REPORT, you can use the derived statistics in the role of determining appropriate ranges for traffic lighting.

A different way to employ traffic lighting might be if you want to draw attention to an entire row instead of just particular cells. A new symbol _ROW_ was created to attain this functionality. Here is the COMPUTE COMMENTS compute block from the previous example with a change – instead of traffic lighting the *COMMENTS* data cell and the *NETSALES* data cell, we are going to traffic light the entire row.

```
compute comments / character length=30;
     if netsales.sum = . then
     do;
        comments = ’Sales figures
                   missing’;
        call define (_ROW_, ’style’,
             ‘style={foreground=cranberry}'
             );
     end;

     else if netsales.sum < netmean then
     do;
        comments = "Sales below average";
        call define (_ROW_, 'style',
```

```
            'style={foreground=red
                   background=oldlace}');
     end;

     else if netsales.sum >= netmean then
     do;
        comments = "Sales above average";
        call define (_ROW_, 'style',
             'style={foreground=green
                   background=oldlace}');
     end;
endcomp;
```

Embedding a SAS Graph within PROC REPORT

Now that we've introduced the special *_ROW_* variable which we discussed previously as a way of changing the style for an entire row, this is a good time to discuss embedding SAS GRAPHS within a PROC REPORT. Embedding a GRAPH uses the CALL DEFINE statement, a new attribute name GRSEG, and the use of either the _ROW_ or _COL_ variable. Currently, embedding a GRAPH in a PROC REPORT is only available to the HTML destination.

Now that we've introduced it and briefly discussed this new feature, let's take a look at an example. This example will not only show the PROC REPORT code, but will also show the PROC GCHART code which creates the graphs that we will embed in the report. This example uses the *SALES* data set that we have been using throughout this paper.

First, here's the PROC GCHART code which creates the graphs that we will embed.

```
goptions device=gif nodisplay ftext=swiss
            hsize=5in vsize=5in  ;


proc gchart data=work.sales gout=work.sales;
   format region $3.;
   format saletype $salefmt.;
   format netsales dollar11.2;

   pie saletype
      / discrete
        sumvar=netsales
        percent=outside
        value=outside
        group=region;

      run;
```

I'm no PROC GCHART expert, but let me try to explain briefly what we're doing here in this code. First we set up some options used by PROC GCHART to output the graph as a GIF image, do not send the output to the display screen, set the font of the text contained in the graph, and also set the horizontal and vertical sizing of the graph. The graph that is output will be a pie chart ordered by *REGION* and then by *SALETYPE* with the variable *NETSALES* as the analysis variable. This code will generate multiple pie charts, one for each region. In our small example, we will have three pie charts: NC, NE, and TX regions.

Now let's turn on the HTML output and then generate the report.

```
ods html body="repgraph.html" nogtitle
            device=gif;
ods listing close;
```

```
proc report data=sales ls=100 nowd;

column region saletype netsales ;
define region   / group;
define saletype / group;
define netsales / analysis sum
format=dollar11.2;

break before region / page;

compute before _page_ /
         style={background=CX0088EE
                foreground=black
                font_weight=bold};
     if( region = 'NC' ) then
        call define( _ROW_, "GRSEG",
                     "work.sales.gchart" );
     else if (region = 'NE' ) then
        call define( _ROW_, "GRSEG",
                     "work.sales.gchart1"
             );
     else if (region = 'TX' ) then
        call define( _ROW_, "GRSEG",
                     "work.sales.gchart2"
               );


     line 'Net Sales for ' region $3. '
Region by
                     Saletype';
endcomp;


run;

ods html close;
```

Let's walk through it a little bit and try and explain what's going on.

```
break before region / page;
```

Since we have generated a graph for each of the regions, we are going to separate each region onto a separate page. When outputting to HTML, the PAGE option tells HTML to create a new table and each table (which is a separate REGION in our case) is also placed on a separate page.

```
compute before _page_ /
         style={background=CX0088EE
                foreground=black
                font_weight=bold};
```

This COMPUTE block is new to Version 7. I don't know how many users have gotten frustrated at not being able to easily place information before the headers of the report, but the COMPUTE BEFORE _PAGE_ compute block is the solution. This compute block tells PROC REPORT to output any of the line statements within the COMPUTE block **before** the column headers. The style option is setting the style for any LINE statements that may be used within this COMPUTE block. In this case, the line produced by

```
line 'Net Sales for ' region $3. '
       Region by Saletype';
```

will have a background color of a very pretty blue which is

denoted by using the graph color notations CX0088EE, a text color of black, and the text will be bolded.

Now, before this line statement are several IF.THEN.ELSE statements in which we are testing the *region* variable to see which graph we should include (depending upon which REGION we are currently processing). Remember earlier when we discussed the generation of the graphs that I mentioned that three graphs (one for each REGION) were created.

```
if( region = 'NC' ) then
     call define( _ROW_, "GRSEG",
             "work.sales.gchart" );
else if (region = 'NE' ) then
     call define( _ROW_, "GRSEG",
             "work.sales.gchart1" );
else if (region = 'TX' ) then
     call define( _ROW_, "GRSEG",
             "work.sales.gchart2" );
```

Did you notice the _ROW_ variable? We are telling HTML that we want to place the graph in the HTML table as one of the rows of the table. Furthermore, since this CALL DEFINE is in the COMPUTE BEFORE _PAGE_ block, this ROW (which becomes the graph) will be embedded in the table at the top of every page before the column headers. Depending upon which REGION we are processing, the correct graph will be embedded.

Embedding the graphs as part of a report is a very useful tool that hopefully benefits the reader of the report. I know I would much rather read a report that has the graphics that pertain to a particular section of the report **with** that section of the report instead of placing all the graphs before or after the report. This removes the burden from the reader of having to keep flipping back and forth between the graph and the report. In this case, the graph and the report are together, and it is easy to see a graphic representation of the report.

Conclusion

You have seen how we used ODS styles to dress up some sample PROC REPORT and PROC TABULATE HTML reports. Now it is your turn to apply these techniques to your real-world reports. You will find that with a relatively small amount of effort you can create visually appealing reports that will grab the attention of your reporting audience.

We look forward to learning about your success stories at future SAS user conferences.

References

SAS Institute Inc. (1999), *The Complete Guide to the SAS Output Delivery System*, Cary, NC: SAS Institute Inc.

Olinger, Christopher (1999), "Twisty Turny Passages, All Alike – PROC TEMPLATE Exposed," *Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference*.

These references are available in online form at the BASE SAS Software Research and Development Web site, http://www.sas.com/rnd/base/.

Contact Information

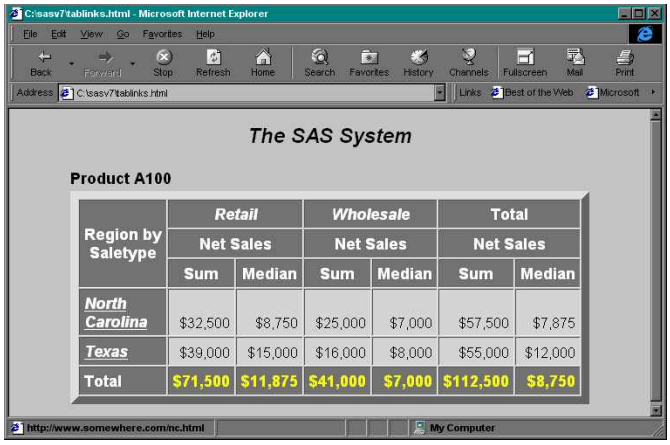
Your comments and questions are valued and encouraged. Contact the authors at:

SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513-2414
Work Phone: (919) 677-8000
Fax: (919) 677-4444
Email: ods@unx.sas.com

Trademarks

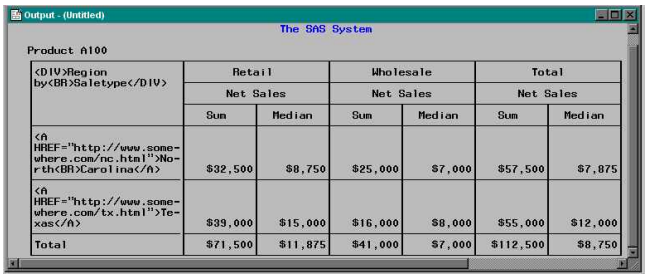
Region by Citysize by Saletype		SaleType				Total	
		Retail		Wholesale			
		Units Sold	Net Sales	Units Sold	Net Sales	Units Sold	Net Sales
		Sum	Sum	Sum	Sum	Sum	Sum
Region	CitySize						
NC	L	800	$20,000	750	$15,000	1,550	$35,000
	M	350	$8,750	350	$7,000	700	$15,750
	S	150	$3,750	150	$3,000	300	$6,750
	Total	1,300	$32,500	1,250	$25,000	2,550	$57,500
NE	CitySize						
	L	Missing	Missing	800	$16,000	800	$16,000
	Total	Missing	Missing	800	$16,000	800	$16,000
TX	CitySize						
	L	760	$19,000	Missing	Missing	760	$19,000
	M	600	$15,000	600	$12,000	1,200	$27,000
	S	200	$5,000	200	$4,000	400	$9,000
	Total	1,560	$39,000	800	$16,000	2,360	$55,000
Total	CitySize						
	L	1,560	$39,000	1,550	$31,000	3,110	$70,000
	M	950	$23,750	950	$19,000	1,900	$42,750
	S	350	$8,750	350	$7,000	700	$15,750
	Total	2,860	$71,500	2,850	$57,000	5,710	$128,500

Figure 1 – Style Regions with PROC TABULATE



Region by Saletype	Retail		Wholesale		Total	
	Net Sales		Net Sales		Net Sales	
	Sum	Median	Sum	Median	Sum	Median
North Carolina	$32,500	$8,750	$25,000	$7,000	$57,500	$7,875
Texas	$39,000	$15,000	$16,000	$8,000	$55,000	$12,000
Total	$71,500	$11,875	$41,000	$7,000	$112,500	$8,750

Figure 2 – Linking with PROC TABULATE



<DIV>Region by Saletype</DIV>	Retail		Wholesale		Total	
	Net Sales		Net Sales		Net Sales	
	Sum	Median	Sum	Median	Sum	Median
<A HREF="http://www.somewhere.com/nc.html">North Carolina</A>	$32,500	$8,750	$25,000	$7,000	$57,500	$7,875
<A HREF="http://www.somewhere.com/tx.html">Texas</A>	$39,000	$15,000	$16,000	$8,000	$55,000	$12,000
Total	$71,500	$11,875	$41,000	$7,000	$112,500	$8,750

Figure 3 – Listing for a PROC TABULATE HTML Report

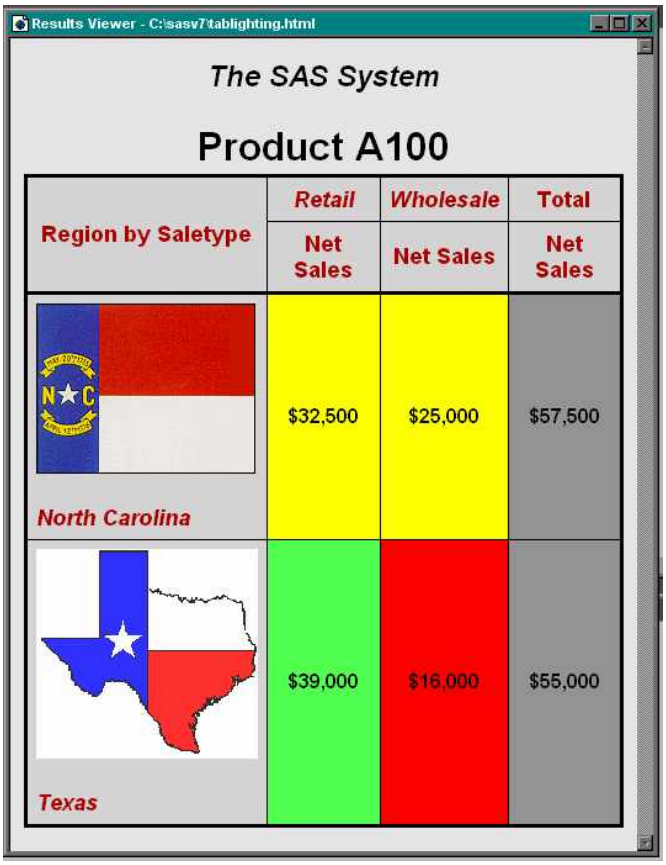


Figure 4 -- Traffic Lighting and Graphic Images in a PROC TABULATE Report

The SAS System

Retail and Wholesale Amounts per Region				
Region	**Citysize**	**Saletype**	**Units**	**Net Sales**
NC	L	Retail	800	$20,000
		Wholesale	750	$15,000
	M	Retail	350	$8,750
		Wholesale	350	$7,000
	S	Retail	150	$3,750
		Wholesale	150	$3,000
NC			*2,550*	*$57,500*
NE	L	Wholesale	800	$16,000
NE			*800*	*$16,000*
TX	L	Retail	760	$19,000
	M	Retail	600	$15,000
		Wholesale	600	$12,000
	S	Retail	200	$5,000
		Wholesale	200	$4,000
TX			*2,360*	*$55,000*
Total Sales for All Regions: $128,500.00				

Figure 5 – Style Regions with PROC REPORT

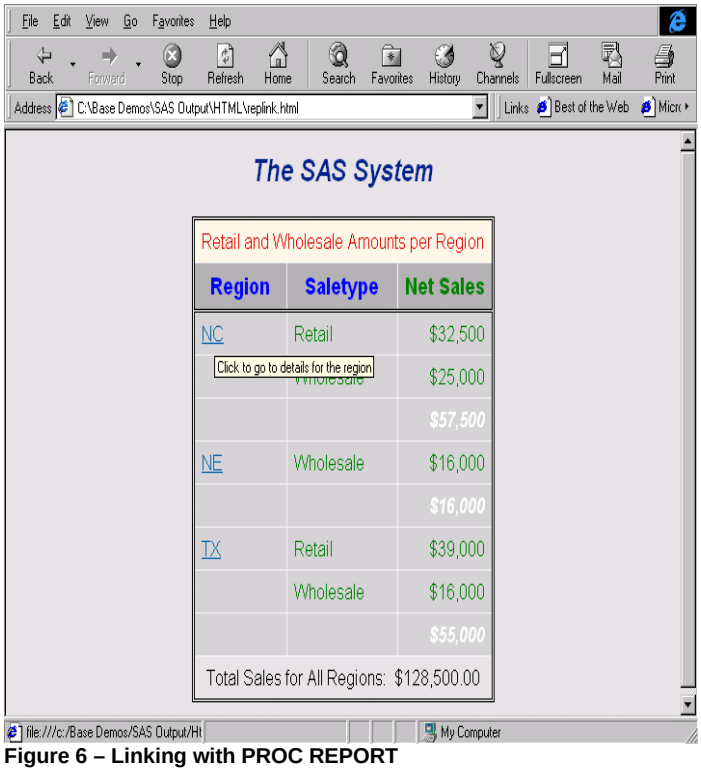
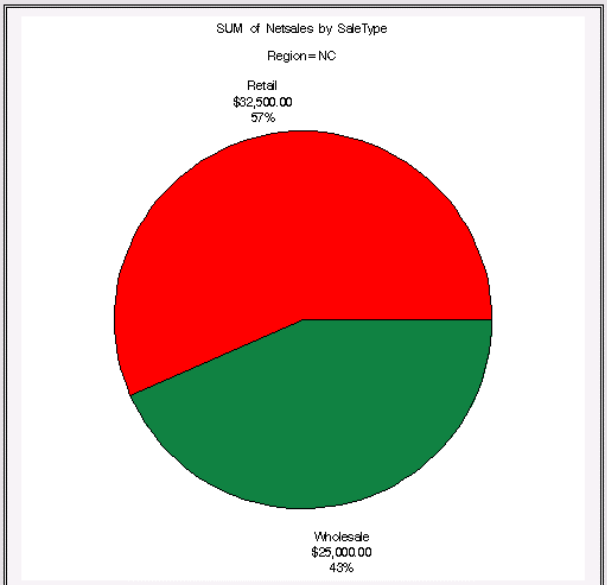


Figure 6 – Linking with PROC REPORT

The SAS System

Product	Region	SaleType	Netsales	Comments
A100	NC	R	$32,500.00	Sales above average
		W	$25,000.00	Sales above average
NC Region -- Well Done! Sales above average!				
	NE	W	$16,000.00	Sales above average
NE Region -- Keep up the good work				
	TX	R	$39,000.00	Sales above average
		W	$16,000.00	Sales above average
TX Region -- Well Done! Sales above average!				

Figure 7 – Traffic Lighting with PROC REPORT



Net Sales for NC Region by Saletype			
Region	**SaleType**	**Netsales**	**Comments**
NC	Retail	$32,500.00	Sales above average
	Wholesale	$25,000.00	Sales above average
NC Region -- Well Done! Sales above average!			

Figure 8 – Embedding a SAS/GRAPH with PROC REPORT