Paper SAS675-2017

Diving Deep into SAS® ODS Graphics Styles

Dan Heath, SAS Institute Inc.

ABSTRACT

Creating an effective style for your graphics can make the difference between clearly conveying your message to your audience and hiding your message in a sea of lines, markers, and text. There have been a number of books written on the concepts of effective graphics, but you need an understanding of how styles work in your environment to correctly apply those principles.

The goal of this paper is to give you an in-depth discussion of how styles are applied to SAS® ODS graphics, from the ODS style level all the way down to the graph syntax. This discussion will include information about differences in grouped versus non-grouped plots, precedence order of style application, using style references, and much more. Don't forget your scuba gear!

INTRODUCTION

Creating effective graphics output is an important task. There are a number of considerations involved, including the choice of chart type, chart styling, and possibly annotations that are needed to convey the information to your audience. There have been great books written on this topic, such as *Creating More Effective Graphs* (Robbins 2005) that give you good general principles to consider when creating your graphs. In this paper, however, I will be dealing specifically with the application of chart styling in the ODS Graphics environment.

I will be dividing this chart styling discussion into three major areas, starting from the most global level to the most granular level. These areas are the following:

- Graph style definitions within ODS styles
- Data-driven style overrides
- Syntax-level style overrides

Throughout the paper, you will see examples using both the SAS SG procedures and Graph Template Language (GTL) templates so that you can see how the syntax looks in both cases. In addition, GTL has some options and abilities not surfaced in the SG procedures, so those items will be pointed out as well.

GRAPH STYLE DEFINITIONS WITHIN ODS STYLES

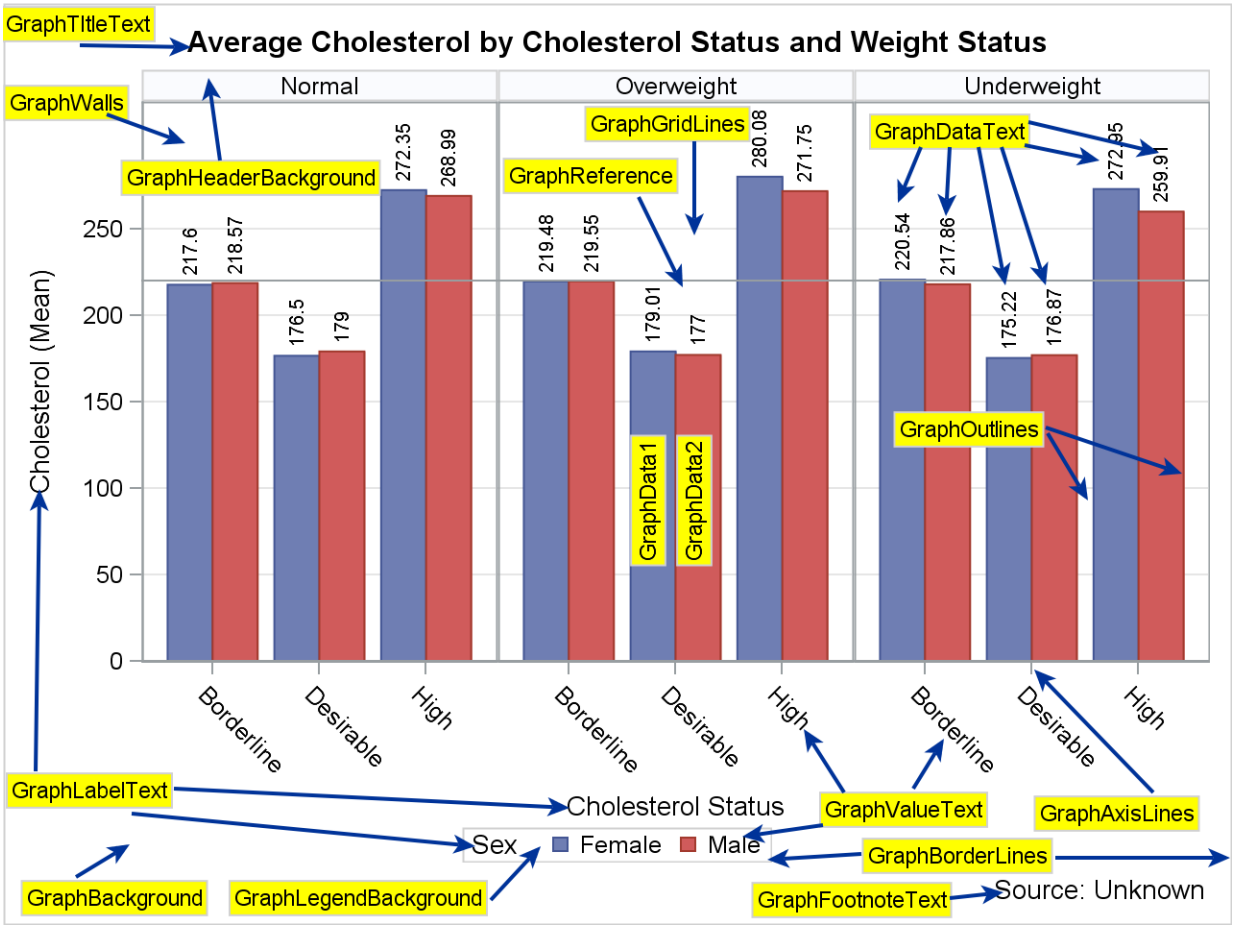
First of all, what is an ODS style? An ODS style is a template definition in the SAS® Output Delivery System (ODS) used to control the appearance of your output. This definition can control output features starting from the page-level appearance, all the way down to the appearance of individual text items. A detailed discussion of ODS styles is beyond the scope of this paper, but I would recommend reading *PROC TEMPLATE Made Easy: A Guide for SAS® Users*, by Kevin Smith (Smith 2013), for more information.

Within the ODS style, there is a collection of style elements that are used by the ODS Graphics system. Some of these graph style elements are shared between ODS Graphics and SAS/GRAPH, and there are some elements that are used by SAS/GRAPH only. The elements that apply only to SAS/GRAPH are in the following list, and will be excluded from this discussion.

- GraphCharts – used to control general transparency for SAS/GRAPH charts.
- GraphTitle1Text – used to control the appearance of the TITLE1 text in SAS/GRAPH charts.
- GraphFloor – used to control the attributes of the floor in PROC G3D graphs using client devices.
- DropShadowStyle – used to control drop shadows on text when using the ACTIVEX or ACTXIMG client devices.

The number of graph style elements is ever growing. The lists below represent the existing elements as of SAS® 9.4, maintenance release 4. I have grouped the elements into different lists to help facilitate subsequent discussion topics. I have also put comments beside the elements to show which graph features use these elements by default. These elements are not limited to these associations, as we will discuss later. The sample charts and plots below will help make visual connections to some of the style elements in the list. I will give more details about the elements that are used most frequently, but I will at least touch on how all of the elements are used.

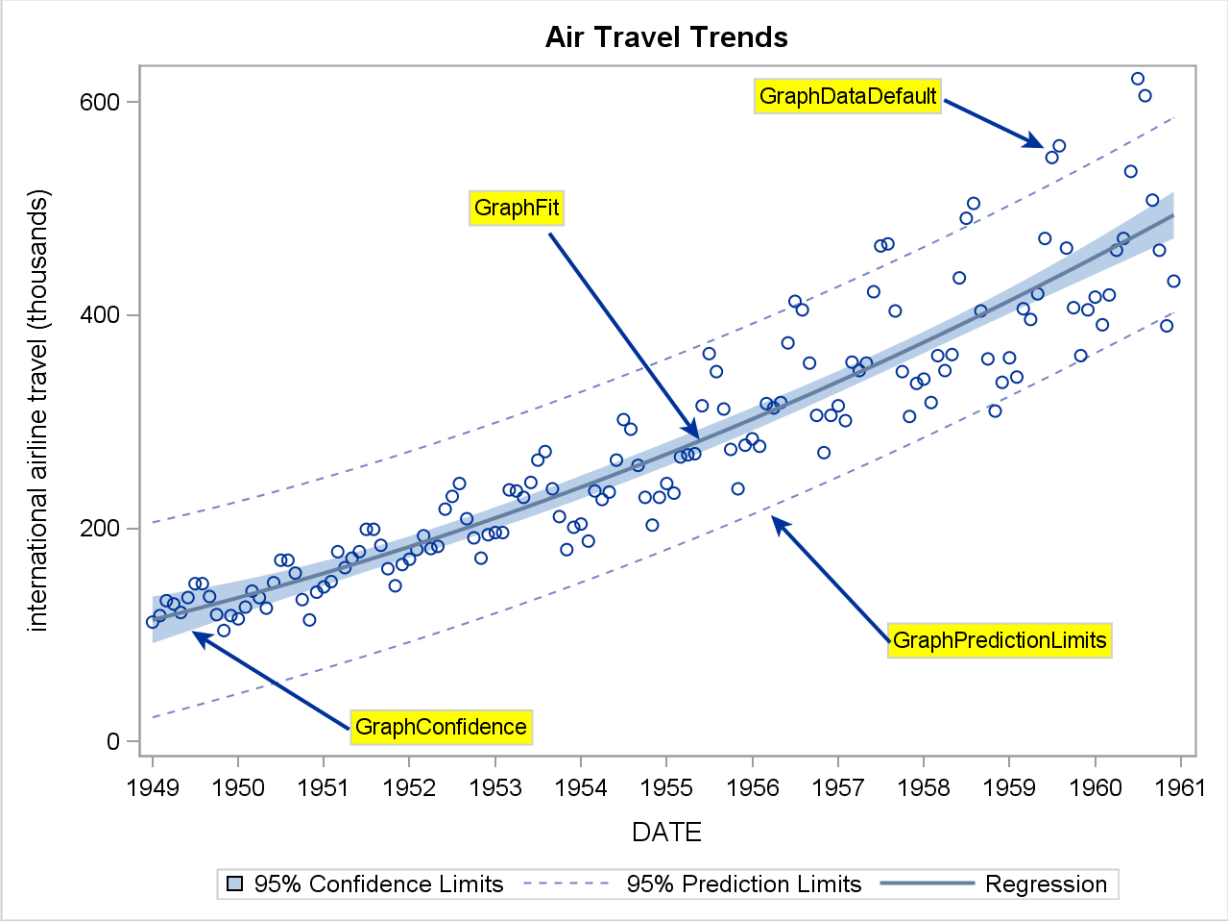
Feature Elements



- GraphWalls – controls the appearance of the wall behind the data area inside the axes.
- GraphAxisLines – controls the attributes of the axis line and tick marks.
- GraphGridLines – controls the attributes of the grid lines.
- GraphMinorGridLines – controls the attributes of the minor grid lines.
- GraphOutlines – controls the attributes of the outlines around data primitives. If the chart is grouped, the outline color will default to the GraphData1-GraphDataN elements.
- GraphBorderLines – controls the border attributes around graphs, layouts, and legends.
- GraphReference – controls the attributes of reference lines.
- GraphTItleText – controls the attributes of title text.
- GraphFootnoteText – controls the attributes of footnote text.

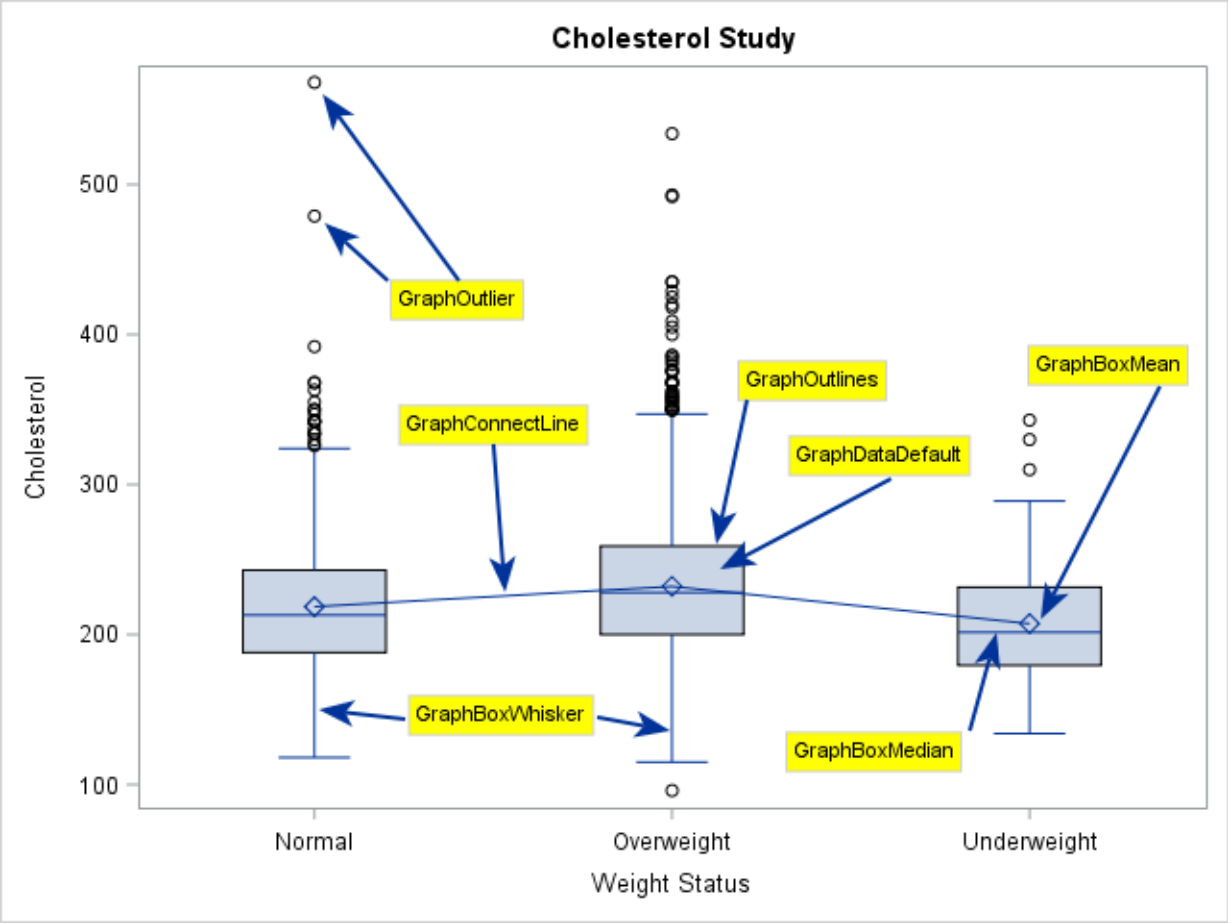
- GraphDataText – controls the attributes of data labels, curve labels, and axis tables. If the chart is grouped, the text color will default to the GraphData1-GraphDataN elements.
- GraphLabelText – controls the attributes of axis labels and legend titles.
- GraphLabel2Text – an element you can reference as an alternative to GraphLabelText. Typically, this element is a non-bold version of GraphLabelText.
- GraphValueText – controls the attributes of axis tick values and legend values.
- GraphUnicodeText – can be referenced when additional Unicode characters are needed.
- GraphBackground – controls the attributes of the entire graph background. You will not see this color behind the axis data area unless the wall fill is turned off.
- GraphLegendBackground – controls the background attributes of legends.
- GraphHeaderBackground – controls the background attributes of cell headers.

Data Elements



- GraphDataDefault – the default style element for **most** non-grouped and no-color-response plots and charts. The list below shows the exceptions to this default:
    - Fit plots (regression, loess, and pbspline) and density plots use GraphFit by default.
    - Model bands and band plots use GraphConfidence. For SGPLOT and SGPANEL, the CLI (prediction limits) are internally assigned to GraphPredictionLimits.

    - Block plots use GraphData1-GraphDataN or the GraphBlock/GraphAltBlock elements, depending on the FILLTYPE setting.
    - Any plot where required attributes are not supplied by another style element.
- GraphFit – the default element for fit plots (regression, loess, and pbspline) and density plots.
- GraphFit2 – a secondary element for fit plot styling that can be referenced from any plot. PROC SGPLOT and SGPANEL will use this element automatically if you overlay two fit plots.
- GraphConfidence – the default element for band plots and model bands.
- GraphConfidence2 – a secondary element for confidence styling that can be referenced from any plot. PROC SGPLOT and SGPANEL will use this element automatically if you overlay two fit plots requesting CLM bands.
- GraphPrediction – an element that can be referenced for predicted fits, such as in forecasting.
- GraphPredictionLimits – an element that can be referenced for prediction limits. This assignment is done automatically in SGPLOT and SGPANEL for CLI bands.



- GraphBoxMean – the default element for the mean marker in a non-grouped box plot.
- GraphBoxMedian – the default element for the median line in a non-grouped box plot.
- GraphBoxWhisker – the default element for the box whiskers in a non-grouped box plot.

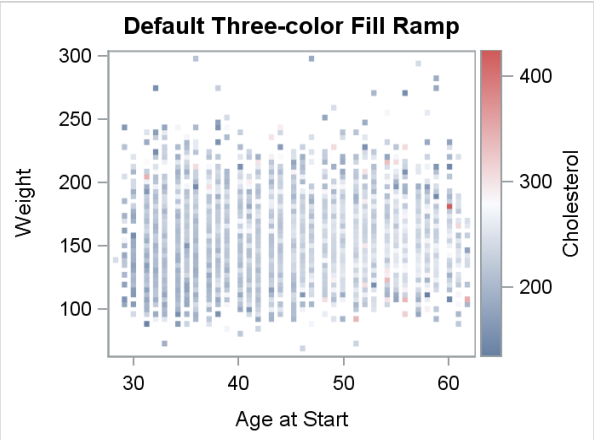
- GraphOutlier – the default element for the box outliers in a non-grouped box plot.
- GraphConnectLine – the default element for the connect line in a non-grouped bar chart or box plot, as well as waterfall charts.
- GraphMissing – contains the attributes used to display missing values in a plot or chart.
- GraphError – the default element for error bars in non-grouped scatter plots and bar charts.
- GraphOther – the default element for the “other” slice in pie charts.
- GraphInitial – the default element for the “initial” bar in waterfall charts.
- GraphFinal – the default element for the “final” bar in waterfall charts.
- GraphUnderflow – the default element for representing underflow values in range attribute maps. (See the section on “data-driven style overrides” for more details.)
- GraphOverflow -- the default element for representing overflow values in range attribute maps. (See the section on “data-driven style overrides” for more details.)

Color Ramps

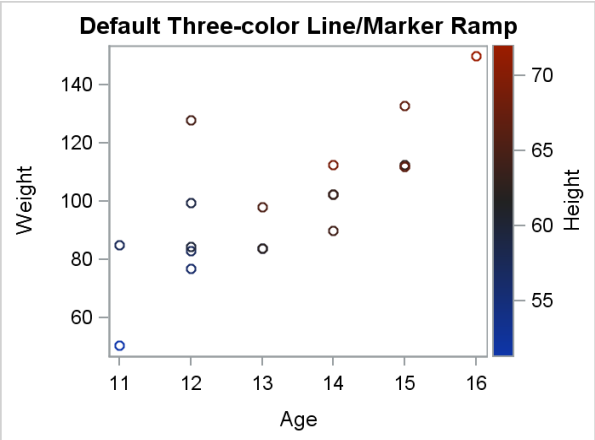
A number of plots in the ODS Graphics system support representing continuous variables as color ramps, including heat maps, scatter plots, and bar charts. The ODS style contains four predefined color ramps, which are the following:

- TwoColorRamp
- TwoColorAltRamp
- ThreeColorRamp
- ThreeColorAltRamp

The colors for the ALT ramps are tuned for use with scatter markers and plot lines, while the other two elements are tuned for filled areas. The examples below show the default three-color ramps in the HTMLBlue style used for fills and markers.

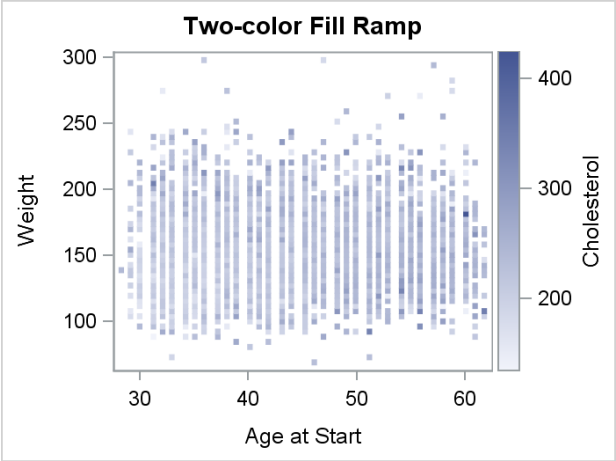


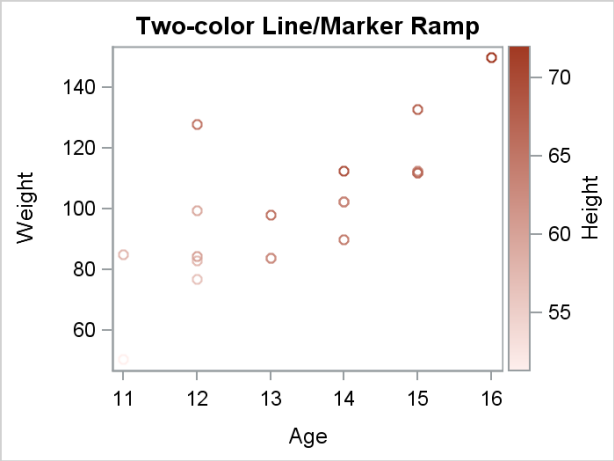

```
proc sgplot data=sashelp.heart;
  heatmap x=AgeAtStart y=weight /
    colorresponse=cholesterol
    colorstat=mean;
run;
```




```
proc sgplot data=sashelp.class;
  scatter x=age y=weight /
    colorresponse=height;
run;
```

For some data, it might be better to use only two colors in the ramp. You can use the two color ramps in the style by *referencing* them in the syntax. We will discuss referencing later in the paper. The pictures below show the same graphs using the two-color ramps:






```
proc sgplot data=sashelp.heart;
  heatmap x=AgeAtStart y=weight /
    colorresponse=cholesterol
    colormodel=TwoColorRamp
    colorstat=mean;
run;
```

```
proc sgplot data=sashelp.class;
  scatter x=age y=weight /
    colormodel=TwoColorAltRamp
    colorresponse=height;
run;
```

Control Elements

Among the graph style elements, there is a class of elements that I like to call “control elements.” These elements do not contain any visual attributes, such as color, shapes, and patterns. But they give you the ability to control the features of certain plots and charts. These elements are a great way to enforce a certain look without having to specify it repeatedly in the graph syntax. The control elements are listed below:

- GraphBar – controls any bar or parameterized bar chart created.
- GraphHistogram – controls any histogram created.
- GraphEllipse – controls any ellipse or parameterized ellipse plot created.
- GraphBand – controls any band or model band plot created.
- GraphContour – controls any contour plot created.
- GraphBox – controls any box plot created.

The amount of control in each element varies. All of them give you the ability to control the display features of the plot or chart, but some of them have additional features. For example, the GraphBox element also gives you the ability to know which feature of the box plots are connected when connect lines are displayed, as well as which endcap style to use for the box whiskers.

In addition to the control elements, there are four “attribute” elements mentioned earlier that also have some control abilities:

- GraphWalls – the FRAMEBORDER attribute can control the display of the wall outline.
- GraphGridLines – the DISPLAYOPTS attribute can control the display of the grid lines.
- GraphMinorGridLines – the DISPLAYOPTS attribute can control the display of the minor grid lines.

- GraphAxisLines – the TICKSTYLE attribute can set the style of how the tick marks are drawn.

To see a good use for these elements, consider the Journal2 style. This style was designed to be a black-and-white style. Therefore, the designers wanted to disable as many fill areas in the plots as possible. Here are the control elements in that style:

```
style GraphHistogram from GraphComponent /
    displayopts = "outline";
style GraphEllipse from GraphComponent /
    displayopts = "outline";
style GraphBand from GraphComponent /
    displayopts = "outline";


style GraphBox from GraphComponent /
    displayopts = "caps median mean outliers"
    connect = "mean"
    capstyle = "serif";
```

In all four style elements, the "fill" is removed from the displayopts attribute, making the plots better for line-based black-and-white graphics.

Data Array Elements

We have discussed the GraphDataDefault element, but there is a collection of graph data elements we have not discussed. The elements start with “GraphData” and end with an index. These array elements must start with 1 and run continuously to the last index.

To fully understand how these elements work, let’s look at the possible attributes in the array element:

```
class GraphData1 /
    color = cxFF0000 /* Used for fill colors */
    contrastcolor = cxcc0000 /* Used for line/marker colors */
    linestyle = 1
    markersymbol = "circle"
end;
```

It is not required that GraphDataN classes contain all of these attributes. However, it is required that an attribute be specified contiguously, starting from GraphData1. This means, for example, that I can specify colors for GraphData1 through GraphData12, but I can specify marker symbols for only GraphData1 through GraphData5. If I were to accidentally specify marker symbols for GraphData1 through GraphData4 and also for GraphData6, the marker symbol in GraphData6 would be ignored because of the “hole” in GraphData5. Only the marker symbols in GraphData1 through GraphData4 would be used. The table in Table 1 makes it easier to visualize how the attributes are mapped to each unique group value.

Style Class	Color	ContrastColor	LineStyle	MarkerSymbol
GraphData1	Blue	Blue	1	Circle
GraphData2	Red	Red	4	Plus
GraphData3	Green	Green	8	Triangle
GraphData4	Yellow	Yellow	5	
GraphData5	Cyan	Cyan		

Table 1. Style Example

If we were creating a grouped scatter plot, the first group value would be drawn as a blue circle, with all of the attributes coming from GraphData1. The next two unique group values would get all of their attributes from GraphData2 and GraphData3, respectively. However, for the next group value, GraphData4 has a color defined, but not a marker symbol. Therefore, the style processor will loop back to GraphData1 and

use its symbol, giving us a yellow circle. The next value would be a cyan plus, and so on, until all unique group values are mapped. If the colors were exhausted, they, too, would loop back to the beginning. Therefore, it is good to use a prime number of attribute values to help maximize the number of attribute combinations for a large number of group values. For this reason, most of the ODS styles shipped with SAS specify 12 colors, 11 line patterns, and 7 markers.

Now that we have discussed the GraphDataN array elements, there is a very useful attribute in the Graph style element called ATTRPRIORITY that you can use to control how the GraphDataN attributes are processed. In addition to the Graph element, this option can be set in the ODS GRAPHICS statement and the BEGINGRAPH statement in GTL.

Setting ATTRPRIORITY="none" (or not setting the attribute at all) gives you the default behavior described earlier. If you set ATTRPRIORITY="color", the line pattern or marker symbol is held constant until all colors are exhausted. Then, the line pattern or marker symbol is incremented, and the color list is traversed again until all group values have attributes assigned. This option can be a useful way to favor solid lines in a grouped SERIESPLOT output or to favor a particular symbol shape in a grouped SCATTERPLOT output.

Other Miscellaneous Elements

There are a number of other graph style elements, but they are not used in most graphing situations. However, I want to briefly describe them in case you should run into situations that use them.

The KPIRange array elements define colors that are typically used in the ranges represented in key performance indicators (KPIs). The ODS Graphics system does not currently support KPIs, but the colors in these elements might be useful in certain graphing situations. Therefore, you might find it useful to reference these elements from a plot or an attributes map (more on this later).

The graph style elements that contain the word "Node" are used only by PATHDIAGRAM and DECISIONTREE plots. The GraphDataNodeN elements typically contain colors that are more "washed-out" versions of the colors in the GraphDataN elements, so you might find them useful in certain situations.

The graph style elements that contain the word "Overview" specify appearance options for the OVERVIEW statement used with the DECISIONTREE statement.

The graph style elements that contain the word "Anno" set default attributes for annotations. Typically, you will fully specify the attributes of your annotations, so these elements will probably not affect your output. However, these elements also affect the annotation tools in the SAS® ODS Graphics Editor.

The GraphCutLine element is used to control the attributes of the cut line that is used in DENDROGRAMs.

The following elements are used specifically by the SAS/QC product:

- GraphControlLimits
- GraphRunTest
- GraphStars
- GraphClipping
- GraphZoneA
- GraphZoneB
- GraphZoneC
- GraphPhaseBox
- GraphBlockHeader

The GraphSelection element is not currently used.

DATA-DRIVEN STYLE OVERRIDES

When the GraphData style elements are assigned to group values, they are assigned in the order in which the data is processed. However, you might need to assign specific attributes to group values, regardless of data order. Sorting your data does not always work, because group values can drop out if data sets are updated before each run. Attribute maps are a great solution for this situation. They give you the ability to associate either style references or literal attributes to either discrete group values or continuous ranges.

Attribute maps can be defined in two ways. For the SG procedures (and for GTL in SAS® 9.4), you can define the map in a SAS data set. For GTL only, you can also define the attribute maps directly in the template syntax. There are two types of attribute maps:

- Discrete attribute maps – assign attributes to inherently discrete data or data made discrete by a SAS format.
- Range attribute maps – assign colors or color ramps to continuous range values.

Range maps can control only colors, while discrete maps can be used to control the following attributes:

- color
- contrast color
- line patterns
- line thickness
- marker symbol
- marker size
- fill and marker transparency
- text attribute support for AXISTABLE statements

Data-based discrete attribute maps use reserved column names to associate group values with attributes. Except for the ID and VALUE columns, all other columns contain attribute values that affect the appearance of the graph. These attribute columns are optional, meaning that you need to define only the columns you need for your graph. However, if you are using attribute maps for many different plot types, it is helpful to define the map enough to create a consistent appearance across all of the graphs in your report.

The ID and VALUE columns are required. The ID column is used to identify a particular map in the data set, because you can define multiple attribute maps within the same data set. This ID value is referenced from the procedures that use the map. The VALUE column contains the group values to map to the attributes. It is important to note that the values in the attribute map data set are treated as *case-sensitive* by default, meaning that the raw data values or the resulting formatted values must have the same case as the map values. Otherwise, the values will not match, and the map attributes for that value will not be used. To make the comparisons case-insensitive, you can create a NOCASE column (SAS 9.4, maintenance release 3) in the attribute map data set and set the value to “True”.

Figure 1 contains a simple example using a data set attribute map. The light blue and the pink need to be associated with the correct gender to enhance the gender-based plot. The data set was defined with only what was needed for this plot.

```
proc format;
   value agefmt 20-29="20-29"
                30-39="30-39"
                40-49="40-49"
                50-59="50-59"
                60-69="60-69";
run;
```

```
data attrmap;
retain id "gender";
length value $ 6 fillcolor $ 9 linecolor $ 8 makercolor $ 8;
input value $ fillcolor $ linecolor $ markercolor $;
cards;
Male   lightblue blue   blue
Female pink      maroon maroon
run;

Title "Cholesterol Levels by Age Group and Gender";
proc sgplot data=sashelp.heart dattrmap=attrmap;
   format AgeAtStart agefmt.;
   vbox cholesterol / category=AgeAtStart group=sex attrid=gender;
   keylegend / location=inside;
run;
```

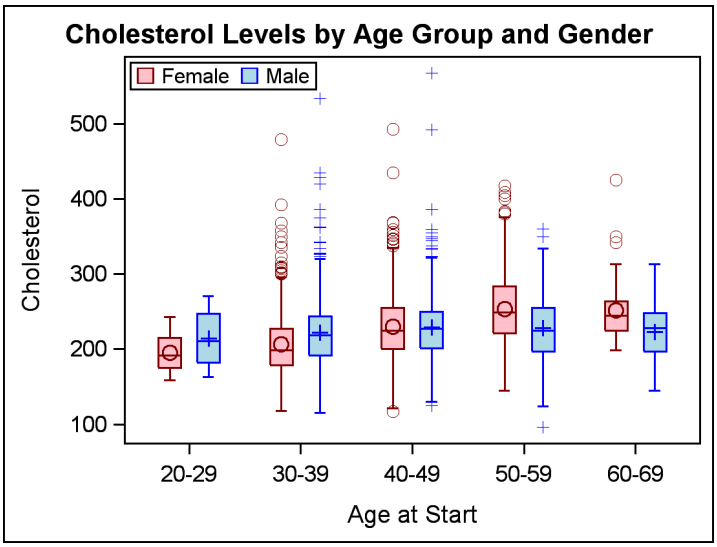


Figure 1. Box Plot with Discrete Attributes Map

The attribute map is bound to the SGPLOT output by two options: the DATTRMAP option to reference the data set and the ATTRID option on the plot to reference the desired map within the data set.

In the first maintenance release for SAS 9.4, the ability to specify attribute maps in data sets has been extended to GTL output via PROC SGRENDER. Keeping the same format code and DATA step in Figure 1, the GTL coding equivalent for Figure 1 would be the following:

```
proc template;
define statgraph boxplot;
begingraph;
EntryTitle "Cholesterol Levels by Age Group and Gender" /;

layout overlay;
   BoxPlot x=AgeAtStart y=Cholesterol / group=sex legendLabel="Cholesterol"
                                        name="VBOX" groupdisplay=cluster;
   DiscreteLegend "VBOX" / Location=Inside
                           autoAlign=(TopLeft TopRight BottomRight
                                      BottomLeft Bottom Top Right Left);
endlayout;
endgraph;
end;
```

```
proc sgrender data=sashelp.heart template=boxplot dattrmap=attrmap;
   format AgeAtStart agefmt.;
   dattrvar sex="gender";
run;
```

In this case, the binding for the attribute map occurs in the DATTRVAR statement of PROC SGRENDER. The value on the left of the equal sign (=) is the group variable to bind to the map. The quoted value on the right is the attribute map ID from the data set. Multiple assignments can be made in the DATTRVAR statement, or multiple statements can be specified.

Using attribute map data sets instead of embedded GTL attribute map syntax is strongly encouraged for several reasons:

1. You can create or modify attribute maps without having to recompile the GTL template.
2. As mentioned earlier, you want to avoid embedding colors in your GTL templates used in a library.
3. You can create and reference multiple maps that have colors customized for different ODS styles while using the same GTL template.
4. The data set approach removes any attribute map dependency from the GTL template.

However, if you are using GTL in SAS® 9.3, it is important for you to understand how the syntax is constructed. Using Figure 1 again, the GTL template with an embedded discrete attributes map would look like the following:

```
proc template;
define statgraph boxplot;
begingraph;
DiscreteAttrMap name="__ATTRMAP__GENDER";
   Value "Male" / markerattrs=(color=CX0000FF)
                  lineattrs=(color=CX0000FF)
                  fillattrs=(color=CXADD8E6);
   Value "Female" / markerattrs=(color=CX800000)
                    lineattrs=(color=CX800000)
                    fillattrs=(color=CXFFC0CB);
EndDiscreteAttrMap;
DiscreteAttrVar attrvar=GENDER_SEX var=SEX attrmap="__ATTRMAP__GENDER";
EntryTitle "Cholesterol Levels by Age Group and Gender" /;
layout overlay;
   BoxPlot X=AgeAtStart Y=Cholesterol / Group=GENDER_SEX
                                        LegendLabel="Cholesterol"
                                        name="VBOX" groupdisplay=cluster;
   DiscreteLegend "VBOX" / Location=Inside
                           autoAlign=(TopLeft TopRight BottomRight
                                      BottomLeft Bottom Top Right Left);
endlayout;
endgraph;
end;

proc sgrender data=sashelp.heart template=boxplot;
   format AgeAtStart agefmt.;
run;
```

The DISCRETEATTRMAP block is used to associate formatted discrete values with visual attributes. The attributes are specified using standard GTL syntax for defining fill, line, and marker attributes. The DISCRETEATTRVAR statement is used to bind the attribute map to the group variable. The group variable is specified in the VAR option, and the map name is specified in the ATTRMAP option. The name

specified in the ATTRVAR option is then used as the group variable in the plot. If the original variable is used in the plot instead of the ATTRVAR name, the attribute map is completely ignored.

Range attribute maps have a similar construction. But instead of assigning the ATTRVAR name to a discrete group option, you must assign it to a continuous variable option, such as COLORRESPONSE or MARKERCOLORGRADIENT. In Figure 2, the range map is used to discretely color the bubbles based on the average cholesterol level for the age group. The size of the bubble (and the label) is based on the number of people in each age group.

```
proc template;
define statgraph bubble;
begingraph;
RangeAttrMap name="__ATTRMAP__CHOL";
   Range 100-<200 / rangecolor=green;
   Range 200-<240 / rangecolor=yellow;
   Range 240-400 / rangecolor=red;
EndRangeAttrMap;
RangeAttrVar attrvar=chol_range var=cholesterol attrmap="__ATTRMAP__CHOL";
EntryTitle "Average Cholesterol Levels by Age Group";
layout overlay / xaxisopts=(type=discrete);
   BubblePlot X=AgeAtStart Y=Cholesterol size=_freq_ / datalabel=_freq_
              datalabelattrs=graphdatatext colorresponse=chol_range
              name="scat";
   ContinuousLegend "scat";
endlayout;
endgraph;
end;

proc summary data=sashelp.heart nway;
   format AgeAtStart agefmt.;
   class AgeAtStart;
   var cholesterol;
   output out=heart mean=;
run;

proc sgrender data=heart template=bubble;
run;
```

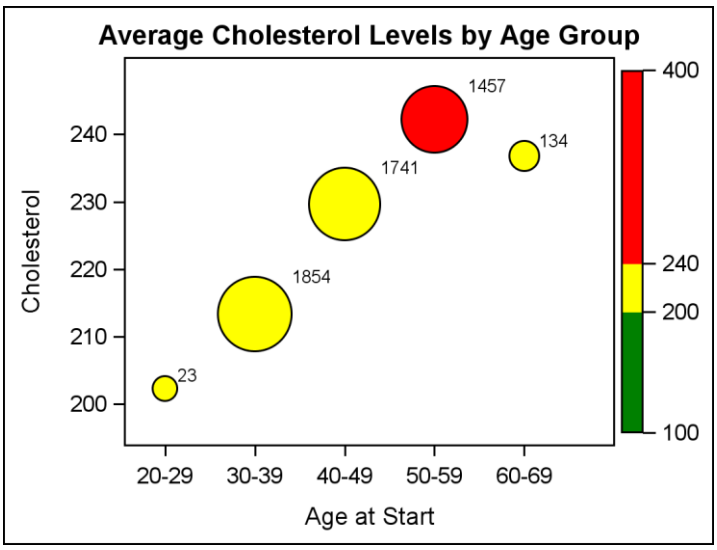


Figure 2. Bubble Plot with Range Attributes Map

Typically, you would use the MIN/MAX keywords (or possibly OVERFLOW/UNDERFLOW) to ensure that all of the plotted data was affected by the map. In this case, however, none of the mean values fell into the desirable range. Because of that, using the MIN keyword would not have allowed all of the cholesterol ranges to appear in the legend. Therefore, I used ranges that I knew included all data so that all ranges would appear in the legend.

Starting with SAS 9.4, maintenance release 3, range attribute maps can now be defined in data sets. Using the example from Figure 2, the code for creating and using the attribute map would look something like the following:

```
data attrmap;
retain id "myid";
length excludemax $ 5 color $ 6;
input min max excludemax $ color $;
cards;
100 200 true  green
200 240 true  yellow
240 400 false red
;
run;

proc format;
   value agefmt 20-29="20-29"
                30-39="30-39"
                40-49="40-49"
                50-59="50-59"
                60-69="60-69";
run;

proc summary data=sashelp.heart nway;
   format AgeAtStart agefmt.;
   class AgeAtStart;
   var cholesterol;
   output out=heart mean=;
run;

proc sgplot data=heart rattrmap=attrmap;
  bubble X=AgeAtStart Y=Cholesterol size=_freq_ / datalabel=_freq_
         datalabelattrs=graphdatatext colorresponse=cholesterol
         rattrid=myid;
run;
```

The ID column is required, as it was with the discrete attribute map. The MIN and MAX columns define the endpoints of each range. The EXCLUDEMAX column determines whether the MAX value is included in the range specification. The COLOR column determines the color assigned to the range. There are a number of other color capabilities using this data set functionality. Please consult the documentation for more details.

When you need to create gradient color transitions across ranges, and you do not have SAS 9.4, maintenance release 3, it is possible to map continuous ranges to discrete colors using a combination of SAS formats and discrete attribute maps to display the plot in a similar way. That way, you can create the plot with either the SG procedures or GTL, and you can use data-driven maps if desired (Figure 3).

```
proc format;
value cholfmt low-<200 = "Desirable"
              200-<240 = "Borderline"
```

```
              240-high = "High";
run;

data attrmap;
retain id "cholrange";
length value $ 10 fillcolor $ 6 linecolor $ 9;
input value $ fillcolor $ linecolor $;
cards;
Desirable  green  darkgreen
Borderline yellow gold
High       red    maroon
;
run;

proc summary data=sashelp.heart nway;
format AgeAtStart agefmt.;
class AgeAtStart;
var cholesterol;
output out=heart mean=;
run;

data heart2 (drop=_temp_);
set heart;
label cholgroup="Cholesterol Level";
if (_n_ = 1) then do;
  _temp_=cholesterol;
  cholgroup = 100;
  cholesterol=.;
  output;
  cholgroup=_temp_;
  cholesterol=_temp_;
  output;
end;
else do;
  cholgroup=cholesterol;
  output;
end;
run;

Title "Average Cholesterol Levels by Age Group";
proc sgplot data=heart2 dattrmap=attrmap;
   format AgeAtStart agefmt. cholgroup cholfmt.;
   xaxis type=discrete;
   bubble x=ageatstart y=cholesterol size=_freq_ / group=cholgroup
          datalabel=_freq_ datalabelattrs=GraphDataText attrid=cholrange;
run;
```

The CHOLFMT format was created to map the cholesterol values to discrete ranges that can be mapped in the ATTRMAP data set. The DATA step for the HEART2 data does two important things for us:

1. It copies the cholesterol values to another column called CHOLGROUP so that we can assign the CHOLFMT format without affecting the axis variable.
2. It prepends a dummy observation with a CHOLGROUP value in the desirable range so that the entry will appear in the legend.

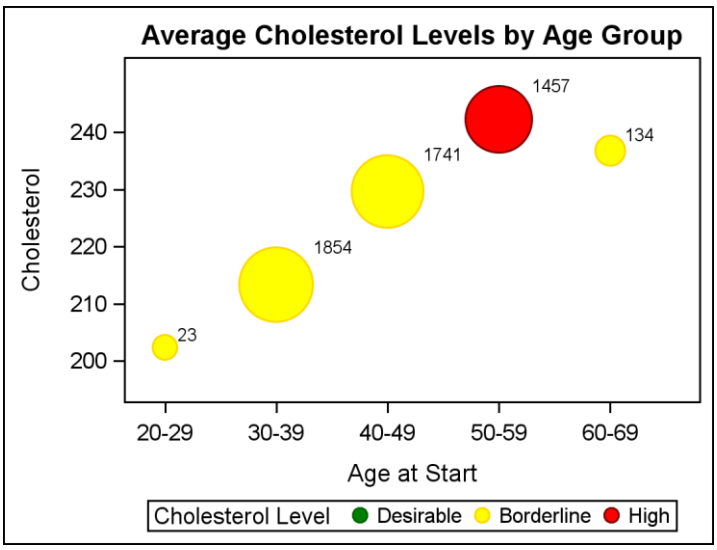


Figure 3. Bubble Plot with Discrete Attributes Map

There is an additional option for style indexing when you use GTL. Several plot types have an option called INDEX. The INDEX option takes a column of index values (1 through N) that correspond to the index values on the GraphDataN elements. The column cannot contain missing values. The SERIESPLOT statement contains multiple index options to give you control over each of the possible four attributes in a series plot.

To use the INDEX option, you must assign a GROUP variable. However, you can use this GROUP and INDEX combination creatively to create colored Y axis values. In the example below, I indexed the MPG_CITY variable for every 5 MPG except for the values over 35 MPG. To use the index variable, I had to assign the MPG_CITY to both the Y option (for the Y axis) and to the GROUP option. (So it is treated as a grouping variable.) Since I also assigned the STYLE_IDX column to the INDEX option, the default grouping assignment is defeated, and the indices from the STYLE_IDX column are used (Figure 4).

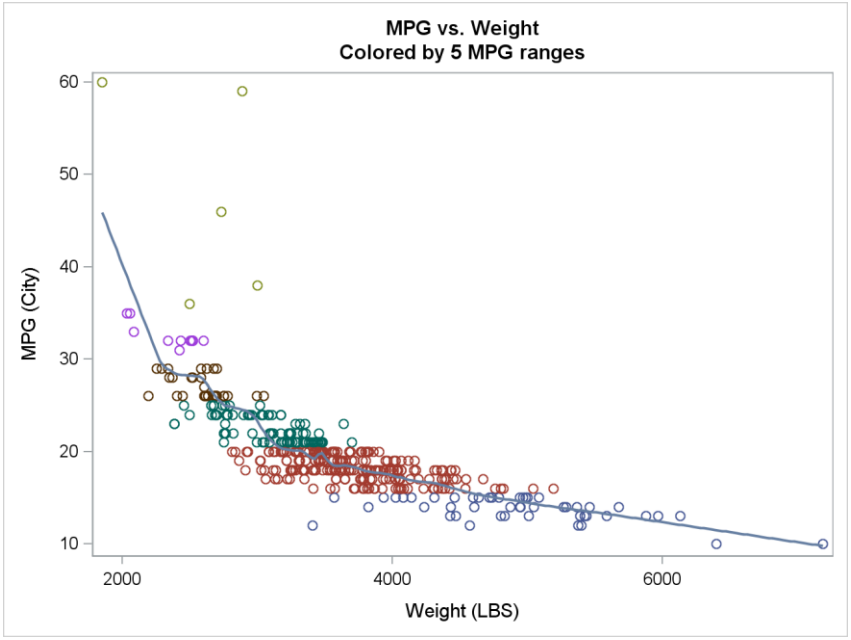


Figure 4. Scatter Plot Using GROUP and INDEX Options

```
data cars;
set sashelp.cars;
if (mpg_city <= 15) then style_idx=1;
else if (mpg_city <= 20) then style_idx=2;
else if (mpg_city <= 25) then style_idx=3;
else if (mpg_city <= 30) then style_idx=4;
else if (mpg_city <= 35) then style_idx=5;
else style_idx=6;
run;

proc template;
define statgraph styleIdx;
begingraph;
  entrytitle "MPG vs. Weight";
  entrytitle "Colored by 5 MPG ranges";
  layout overlay;
     scatterplot x=weight y=mpg_city / group=mpg_city index=style_idx;
     loessplot x=weight y=mpg_city;
  endlayout;
endgraph;
end;
run;

proc sgrender data=cars template=styleIdx; run;
```

This technique could also be used to identify individual points in a plot, such as outliers.

SYNTAX-LEVEL STYLE OVERRIDES

In the graph style section, we discussed how certain elements are automatically assigned to parts of the graph. We also mentioned that style elements can be used by reference from the graph syntax. We will discuss how referencing works in the examples below, as well as ways you can directly override the default style elements.

First, you can override the list of attributes in the GraphDataN elements directly from syntax. The SG procedures have a statement called STYLEATTRS that contains the options to override the GraphDataN style classes. For GTL users, those options are in the BEGINGRAPH statement. The following options are used to replace the style attributes:

- DATACOLORS—used to replace the fill colors from the GraphDataN classes.
- DATACONTRASTCOLORS—used to replace the line and marker colors from the GraphDataN classes.
- DATALINEPATTERNS—used to replace the line patterns from the GraphDataN classes.
- DATASYMBOLS—used to replace the marker symbols from the GraphDataN classes.

In Figure 5, the COLOR and CONTRASTCOLOR values from the GraphDataN element are totally replaced by the values in the DATACOLORS and DATACONTRASTCOLORS options, respectively. If the CONTRASTCOLORS were not replaced by the single black color, the outlines of the bars would have used each CONTRASTCOLOR from GraphData1-5.

```
proc sgplot data=sashelp.prdsale noautolegend;
   where year=1993 and quarter<=2;
   styleattrs datacolors=(cxedf8fb cxb2e2e2 cx66c2a4 cx2ca25f cx006d2c)
              datacontrastcolors=(black);
   hbar product / response=actual group=product;
run;
```

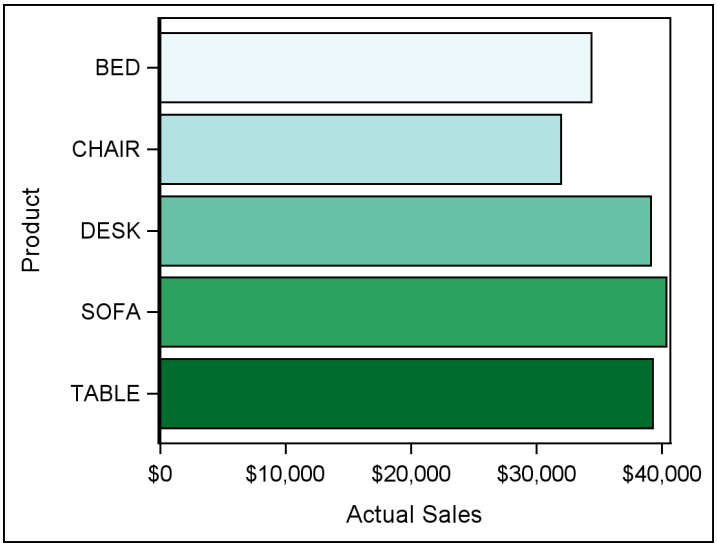
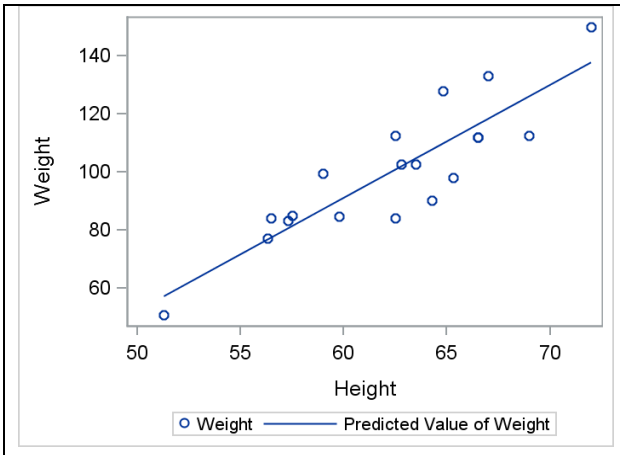


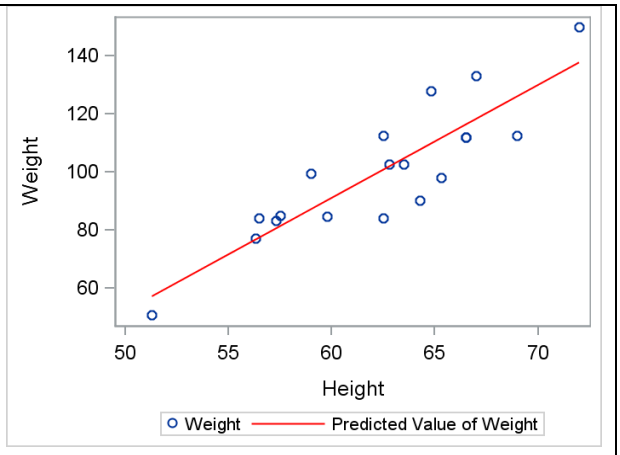
Figure 5. Bar Chart with New Style Colors

Throughout the ODS Graphics system, the plot statements contain what are called “attribute bundles.” They generally fall into four categories:

- Fill attributes – contains color and transparency values.
- Line attributes – contains color, pattern, and thickness values.
- Marker attributes – contains color, symbol, and size values.
- Text attributes – contains family, size, weight, and style values.




```
proc sgplot data=sashelp.classfit;
scatter x=height y=weight;
series x=height y=predict;
run;
```

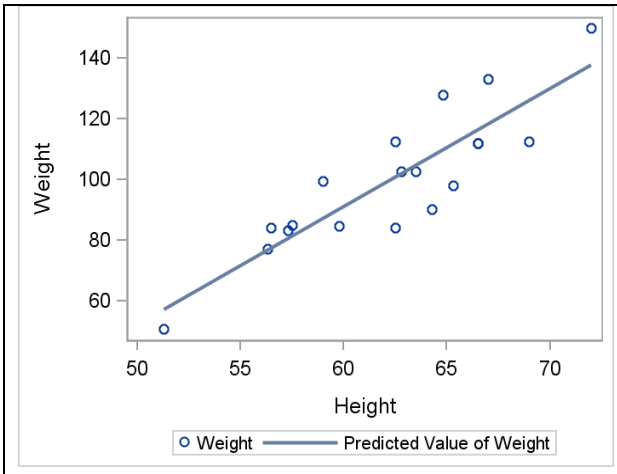



```
proc sgplot data=sashelp.classfit;
scatter x=height y=weight;
series x=height y=predict /
       lineattrs=(color=red);
run;
```

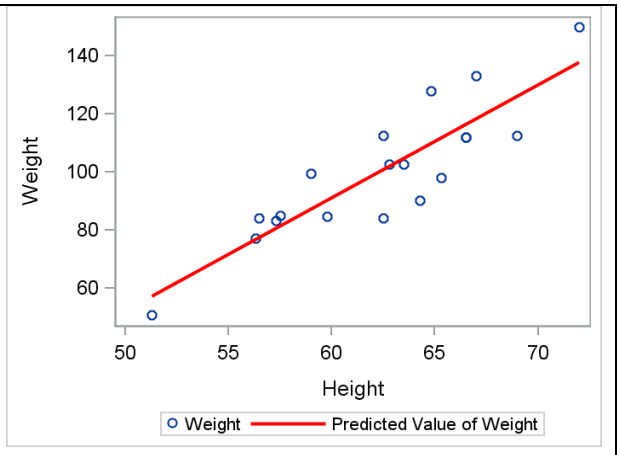
Figure 6. Default Color versus Style Override

The examples in Figure 6 show how the CONTRASTCOLOR in the GraphDataDefault element can be overridden by the red color in the LINEATTRS bundle. With the data, however, we know we are dealing with fit data. Therefore, it would probably be best to make it look like a fit line. The easiest way to do this is to make a style reference to the GraphFit element. After making this reference, it is still possible to

override attributes from the GraphFit element by using the bundle options. In this case, I set the color back to red (Figure 7).




```
proc sgplot data=sashelp.classfit;
scatter x=height y=weight;
series x=height y=predict /
       lineattrs=GraphFit;
run;
```

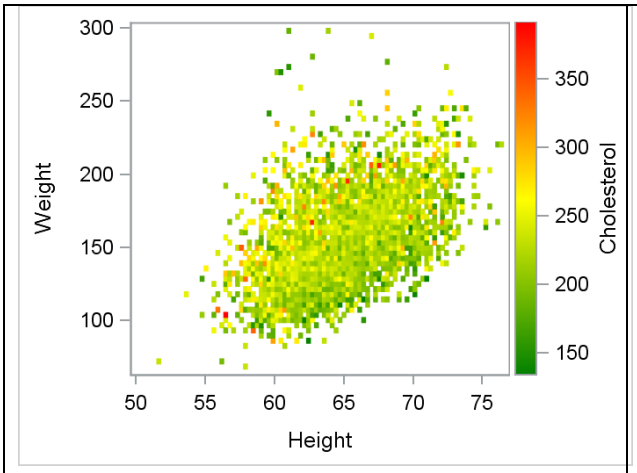



```
proc sgplot data=sashelp.classfit;
scatter x=height y=weight;
series x=height y=predict /
       lineattrs=GraphFit(color=red);
run;
```

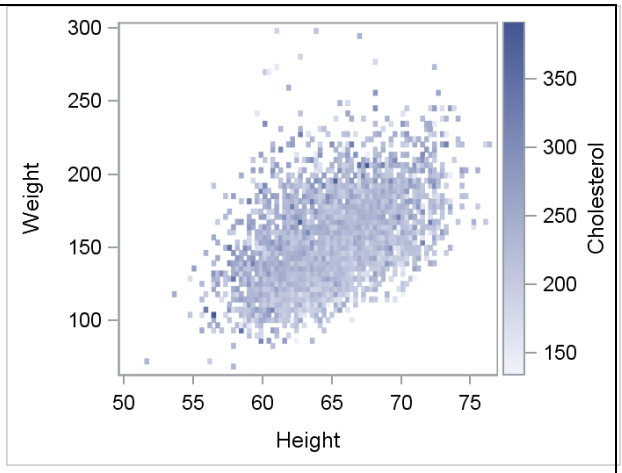
Figure 7. Style Reference with and without an Override

You can refer to any style element you want, but the graph feature will use only the applicable attributes. Any missing required attributes will be picked up from the feature's default style element.

The style reference concept also applies to the COLORMODEL option, which is used to control the color ramps used for the COLORRESPONSE and other continuous variable options. Unlike the previous bundles, the COLORMODEL option is either a list of color or a style reference, but not both (Figure 8).




```
proc sgplot data=sashelp.heart;
heatmap x=height y=weight /
       colorresponse=cholesterol
       colorstat=mean
       colormodel=(green yellow red);
run;
```




```
proc sgplot data=sashelp.heart;
heatmap x=height y=weight /
       colorresponse=cholesterol
       colorstat=mean
       colormodel=twocolorramp;
run;
```

Figure 8. Colormodel Option

CONCLUSION

In this paper, we have covered a lot of material regarding graph styles, including how to reference them and how to override them. I encourage you to use the first section on the graph elements as a reference as you work on your graphs. I also encourage you to explore the documentation on attribute maps to learn all of the available options to get the most of that functionality.

REFERENCES

Robbins, Naomi B. 2005. *Creating More Effective Graphs.* New Jersey: John Wiley & Sons, Inc.

Smith, Kevin D. 2013. *PROC TEMPLATE Made Easy: A Guide for SAS Users.* Cary, North Carolina: SAS Institute, Inc.

CONTACT INFORMATION

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

Dan Heath
SAS Institute, Inc.
(919) 677-8000
Dan.Heath@sas.com