PharmaSUG 2016 - Paper DG04

Fifty Ways to Change your Colors (in ODS Graphics)

Shane Rosanbalm, Rho, Inc., Chapel Hill, NC

ABSTRACT

Back in the good ole days (think GPLOT) there was one primary way to change the colors of your symbols and lines: the COLOR= option of the SYMBOL statement. But now that ODS graphics are taking over (think SGPLOT and GTL), color management is not so straightforward. There are plot statement attribute options, style modifications, the %modstyle macro, discrete attribute maps, and more. Sometimes it feels like there must be 50 ways to change your colors.

In this paper we will explore the various ways to manage colors in ODS graphics. Complex topics will be demystified. Strengths and weaknesses will be examined. Recommendations will be made. And with any luck, you will come away feeling less confused and more confident about how to manage colors in ODS graphics.

INTRODUCTION

In this paper we will explore various ways to manage the colors of markers and lines in ODS graphics. Familiarity with RGB color codes is assumed. If you are not familiar with RGB color codes, see Appendix 1 for a brief introduction.

For the sake of simplicity we will use SGPLOT to drive the conversation. However, the principles apply equally to SGPANEL, GTL, etc.

Throughout this paper we will produce regression plots with the REG statement. For instance, the SASHELP.CARS dataset can be used as follows:

```
proc sgplot data=sashelp.cars;
   reg x=horsepower y=enginesize;
run;
```

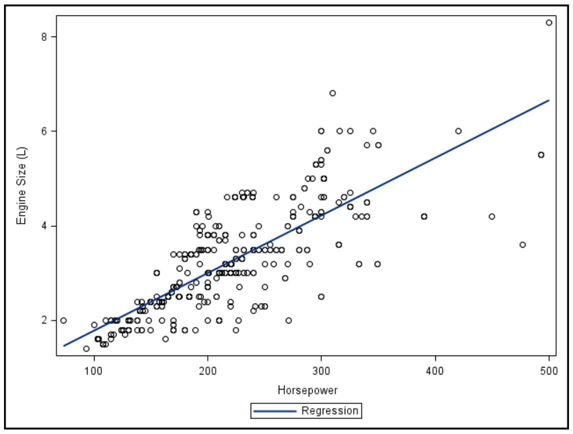


Figure 1. A basic regression plot

Regression plots are convenient examples because they contain both markers and lines, allowing us to cover color management for markers and lines in one go.

OUTLINE

We will begin our exploration of color management with Non-grouped Data. The straightforward MARKERATTRS and LINEATTRS options will be introduced. This will be followed by a very brief introduction to styles, opening the door to managing colors through style modifications. If you are the impatient type, feel free to jump straight to the end of the Non-grouped Data Summary.

We will then switch to Grouped Data. The 9.2 approach of style modifications will be covered first (including the %modstyle macro). Next will be the 9.3 approach of discrete attribute maps, followed by the 9.4 approach of the STYLEATTRS statement. If you are the impatient type, feel free to jump straight to the Grouped Data Summary.

NON-GROUPED DATA

THE GOAL

Suppose it is our goal to change the color of the *markers* in the above SASHELP.CARS regression plot so that they perfectly match the color of the regression line. Something like:

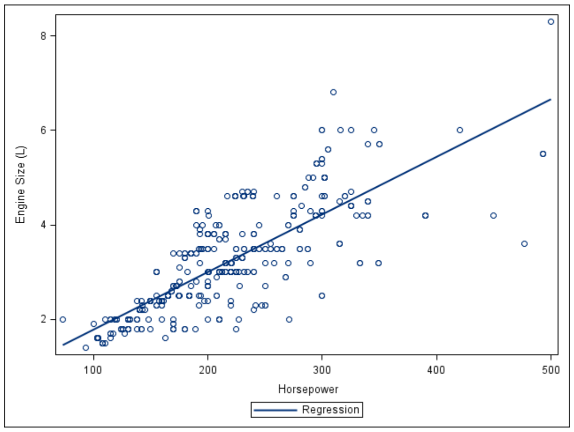
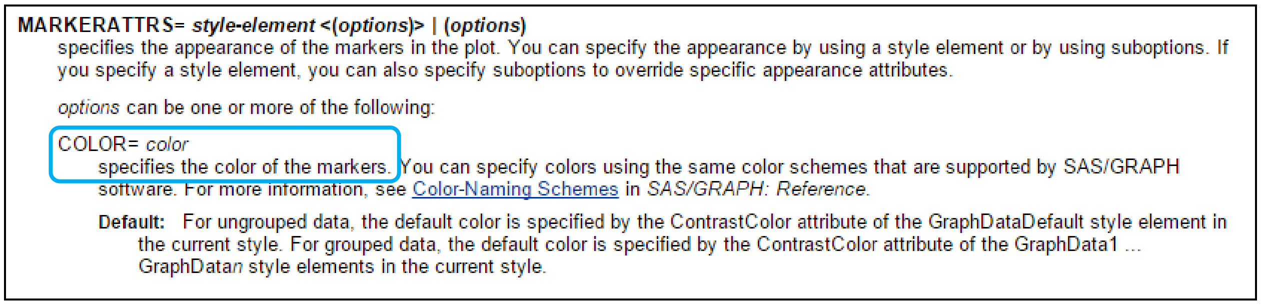


Figure 2. Change marker color to match line

MARKERATTRS/LINEATTRS

The simplest way to change marker and line colors in SGPLOT is through the MARKERATTRS and LINEATTRS options. Both options contain a COLOR= sub-option (focus on the text in the blue rectangle).



MARKERATTRS= ***style-element*** **<(*options*)> | (*options*)**

specifies the appearance of the markers in the plot. You can specify the appearance by using a style element or by using suboptions. If you specify a style element, you can also specify suboptions to override specific appearance attributes.

options can be one or more of the following:

COLOR= *color*

specifies the color of the markers. You can specify colors using the same color schemes that are supported by SAS/GRAPH software. For more information, see Color-Naming Schemes in *SAS/GRAPH: Reference*.

Default: For ungrouped data, the default color is specified by the ContrastColor attribute of the GraphDataDefault style element in the current style. For grouped data, the default color is specified by the ContrastColor attribute of the GraphData1 ... GraphData*n* style elements in the current style.

If you wanted to change your markers from black to blue in the SASHELP.CARS regression plot, you would use the MARKERATTRS option as follows:

```
proc sgplot data=sashelp.cars;
   reg x=horsepower y=enginesize /
      markerattrs=(color=blue);
run;
```

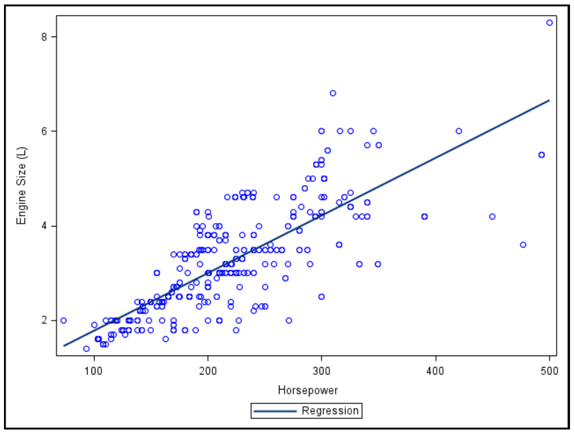
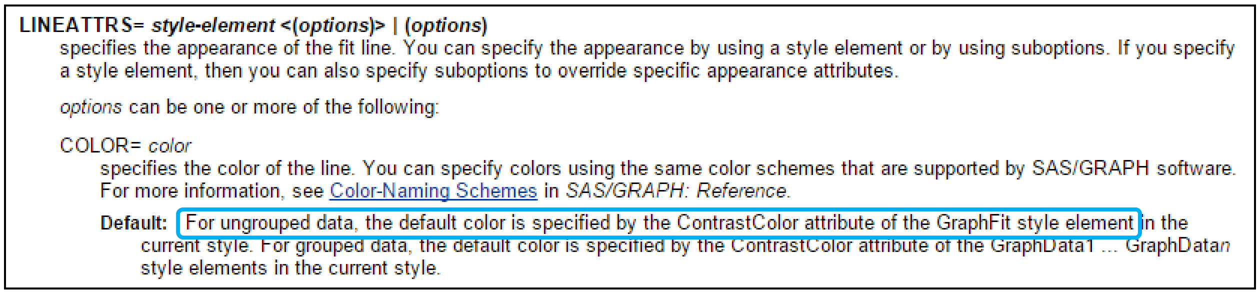


Figure 3. Hack solution

Unfortunately, this is not the exact same shade of blue as the regression line. How do we find the exact right shade of blue? This is where styles come in.

STYLES 101

In order to be able to match the color of the markers to the color of the regression line, we need to know exactly what color the line is. In ODS Graphics, colors are controlled by styles, as evidenced by the following excerpt from the online documentation for the REG statement (focus on the text in the blue rectangle).



LINEATTRS= ***style-element*** **<(*options*)> | (*options*)**
specifies the appearance of the fit line. You can specify the appearance by using a style element or by using suboptions. If you specify a style element, then you can also specify suboptions to override specific appearance attributes.

options can be one or more of the following:

COLOR= *color*
specifies the color of the line. You can specify colors using the same color schemes that are supported by SAS/GRAPH software. For more information, see Color-Naming Schemes in *SAS/GRAPH: Reference*.

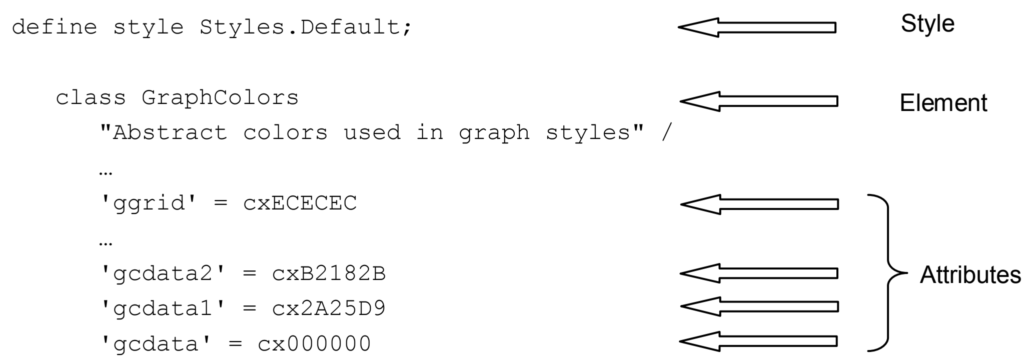
Default: For ungrouped data, the default color is specified by the ContrastColor attribute of the GraphFit style element in the current style. For grouped data, the default color is specified by the ContrastColor attribute of the GraphData1 ... GraphData*n* style elements in the current style.

Elements!? And attributes!? Whaaaat?

The appearance of ODS graphics output is controlled by styles. The most basic style in SAS® is STYLES.DEFAULT. The source code for a style can be revealed using the SOURCE statement in PROC TEMPLATE.

```
proc template;
   source styles.default;
run;
```

This SOURCE statement writes the source code for STYLES.DEFAULT to the SAS log. A partial excerpt of the source code for STYLES.DEFAULT contains:



```
define style Styles.Default;

   class GraphColors
      "Abstract colors used in graph styles" /
      …
      'ggrid' = cxECECEC
      …
      'gcdata2' = cxB2182B
      'gcdata1' = cx2A25D9
      'gcdata' = cx000000
```

What this partial excerpt helps to illustrate is that styles consist of elements and attributes.

- Attributes are the fine details about how various components of a graph are to be displayed.
- Elements are collections of related attributes.

The element GraphColors shown above is a collection of attributes related to the colors of various components of graphical output. For instance:

- 'ggrid' defines the color used for grid lines
- 'gcdata' defines the color used for non-grouped markers and lines
- Etc.

Yeah, but what color was that regression line!?

The online documentation tells us to look for element = GraphFit and attribute = ContrastColor. Using Ctrl+F in the log we easily find GraphFit.

```
   class GraphFit /
         linethickness = 2px
         linestyle = 1
         markersize = 7px
         markersymbol = "circle"
         contrastcolor = GraphColors('gcfit')
         color = GraphColors('gfit');
```

Note that the ContrastColor value is not actually a color at all, but is instead a reference to another element + attribute pair within the style. Undeterred, we use Ctrl+F in the log to search for GraphColors.

```
   class GraphColors
         "Abstract colors used in graph styles" /
         …
         'gcfit2' = cx780000
         'gcfit' = cx003178
         …
         'gdata1' = cx7C95CA;
```

And at last we have found the color used for the regression line: cx003178. We now put that color into our MARKERATTRS option to achieve the desired plot:

```
   proc sgplot data=sashelp.cars;
      reg x=horsepower y=enginesize /
         markerattrs=(color=cx003178);
   run;
```

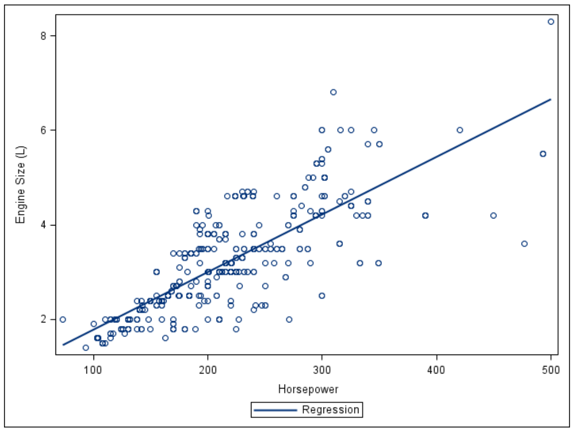


Figure 4. Marker color now matches line color

STYLE MODIFICATION

The advantage of using MARKERATTRS and LINEATTRS to change colors is that this approach is straightforward. Unfortunately, these options do not scale well. If you are producing several plots, or managing the colors for grouped data, MARKERATTRS and LINEATTRS just don't work as well.

A solution that scales better is style modification, something which can be achieved with just a couple of lines of code. For instance, the following code creates a new style named MYSTYLE. This new style uses STYLES.LISTING as a starting point (i.e., the parent), and changes the default non-grouped marker color from black to cx003178.

```
   proc template;
      define style MyStyle;                          ⇦ Name the new style being created.
         parent = Styles.Listing;                       ⇦ Use STYLES.LISTING as a starting point.
         class GraphDataDefault / contrastcolor = cx003178;  ⇦ Change this one tiny detail.
      end;
   run;
```

You may have noticed that we used STYLES.DEFAULT in our SOURCE statements in the previous section, but we switched to using STYLES.LISTING as the PARENT in the style modification above. The reason we switched to STYLES.LISTING is because this is the default style in the LISTING destination.

The default style for every ODS destination can be found at Summary of Default Destinations, Styles, and Devices.

Now that this new style has been defined, we simply point to it in an ODS LISTING statement. We then simplify our REG statement by *removing* the unsightly MARKERATTRS option.

```
ods listing style=MyStyle;

proc sgplot data=cylinders;
   reg x=horsepower y=enginesize;
run;
```

Putting the TEMPLATE, ODS, and SGPLOT code together in one continuous code block we can see:

```
proc template;
   define style MyStyle;
      parent = Styles.Listing;
      class GraphDataDefault /
         contrastcolor = cx003178
         ;
   end;
run;

ods listing style=MyStyle;

proc sgplot data=cylinders;
   reg x=horsepower y=enginesize;
run;
```

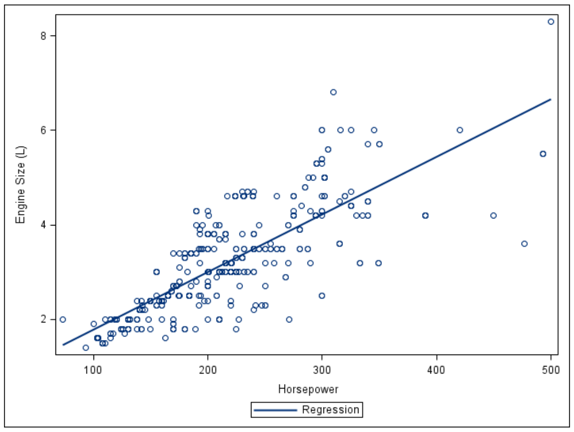


Figure 5. Marker color changed through styles

Hardcoded Colors vs. Element + Attribute References

Recall how we found the value cx003178 in the first place. First we looked at the documentation to discover that the color of the regression line came from the ContrastColor in GraphFit. When we looked in our log we saw that ContrastColor referenced GraphColors('gcfit'). It was under 'gcfit' that we finally found cx003178.

When defining a new style in PROC TEMPLATE we don't actually have to hardcode a color value. We are also allowed to refer to element + attribute pairs. For instance, GraphColors('gcfit'). Making this substitution in our above code would result in:

```
proc template;
   define style MyStyle;
      parent = Styles.Listing;
      class GraphDataDefault /
         contrastcolor = GraphColors('gcfit')
         ;
   end;
run;
```

This approach has the advantage of being robust to changes in styles. For instance, suppose we wanted to switch to the ANALYSIS style. This style uses a lot of greens, and the color cx003178 would not match the regression line. When given the choice, use element + attribute references instead of hardcoded colors to make style modifications.

NON-GROUPED DATA SUMMARY

For non-grouped data we looked at two ways to change colors.

MARKERATTRS/LINEATTRS

```
proc sgplot data=sashelp.cars;
   reg x=horsepower y=enginesize / markerattrs=(color=cx003178);
run;
```

Style Modifications

```
proc template;
   define style MyStyle;
      parent = Styles.Listing;
      class GraphDataDefault / contrastcolor = GraphColors('gcfit');
   end;
run;

ods listing style=MyStyle;

proc sgplot data=cylinders;
   reg x=horsepower y=enginesize;
run;
```

While the MARKERATTRS approach certainly has the advantage of brevity, you should not dismiss the style modifications approach. As we shall see in the grouped data section, a basic understanding of styles and style modifications is essential.

GROUPED DATA

We will now move on to color management for grouped data. Again, we use a regression plot based on SASHELP.CARS to drive the conversation.

```
proc sort data=sashelp.cars out=cars;
   by cylinders;
   where n(cylinders);
run;

proc sgplot data=work.cars;
   reg x=horsepower y=enginesize /
      group=cylinders;
run;
```

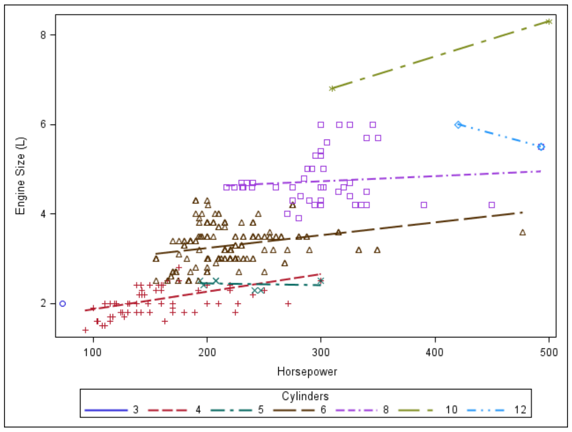


Figure 6. A basic grouped regression plot

THE GOAL

Let's switch to a slightly less busy regression plot. Specifically, let's subset down to the cylinder values of 4, 6, and 8. This gives us colors that are roughly (but not exactly) blue, red, and green.

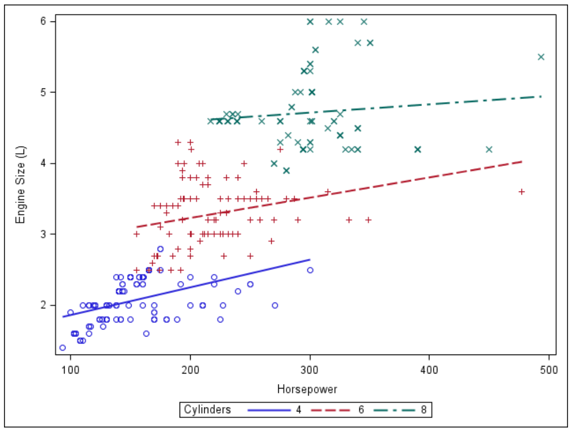


Figure 7. Subset down to 4, 6, and 8 cylinders

Recall that the first grouped regression plot had 7 different colors. Suppose we like the colors that are roughly (but not exactly) brown, purple, and pea green. That is, we like the colors that are associated with the 4th, 5th, and 6th groups in that original 7-group plot. How would we go about updating our plot to use these colors?

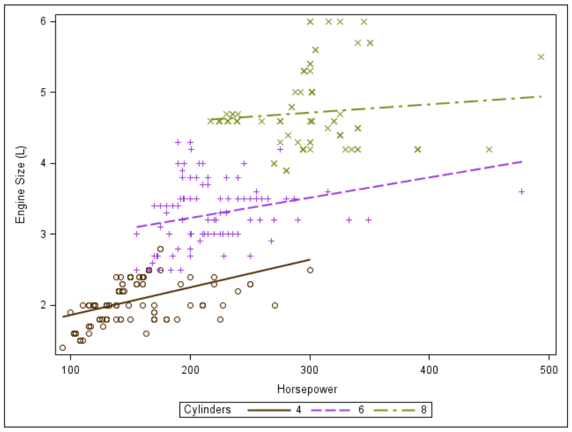
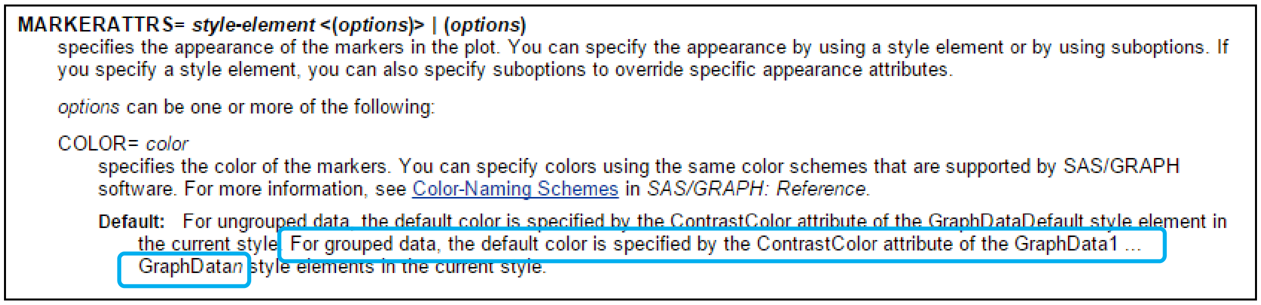


Figure 8. Change colors away from defaults

STYLE MODIFICATION FOR GROUPED DATA

Looking back at the documentation for the REG statement, we can see how colors are assigned for grouped data (focus on the text in the blue rectangle).



MARKERATTRS= ***style-element*** **<(*options*)> | (*options*)**

specifies the appearance of the markers in the plot. You can specify the appearance by using a style element or by using suboptions. If you specify a style element, you can also specify suboptions to override specific appearance attributes.

options can be one or more of the following:

COLOR= *color*

specifies the color of the markers. You can specify colors using the same color schemes that are supported by SAS/GRAPH software. For more information, see Color-Naming Schemes in *SAS/GRAPH: Reference*.

Default: For ungrouped data, the default color is specified by the ContrastColor attribute of the GraphDataDefault style element in the current style. For grouped data, the default color is specified by the ContrastColor attribute of the GraphData1 ... GraphData*n* style elements in the current style.

Looking in our log at the source code for STYLES.DEFAULT we can see how the colors are defined for each group. The ContrastColor attribute within each of the GraphData*n* elements refers to an attribute of the form 'gcdata*n*'. For example:

```
      class GraphData1 /
         markersymbol = "circle"
         linestyle = 1
         contrastcolor = GraphColors('gcdata1')
         color = GraphColors('gdata1');
      class GraphData2 /
         markersymbol = "plus"
         linestyle = 2
         contrastcolor = GraphColors('gcdata2')
         color = GraphColors('gdata2');
      class GraphData3 /
         markersymbol = "X"
         linestyle = 8
         contrastcolor = GraphColors('gcdata3')
         color = GraphColors('gdata3');
```

The colors brown, purple, and pea green come from gcdata4, gcdata5, and gcdata6. In order to get these colors into our plot we simply need to update the ContrastColor definitions for GraphData1, GraphData2, and GraphData3.

```
proc template;
   define style MyStyle;
      parent = Styles.Listing;
      class GraphData1 / contrastcolor = GraphColors('gcdata4');
      class GraphData2 / contrastcolor = GraphColors('gcdata5');
      class GraphData3 / contrastcolor = GraphColors('gcdata6');
   end;
run;

ods listing style=MyStyle;

proc sgplot data=work.cars;
   reg x=horsepower y=enginesize /
     group=cylinders;
   where cylinders in (4 6 8);
run;
```

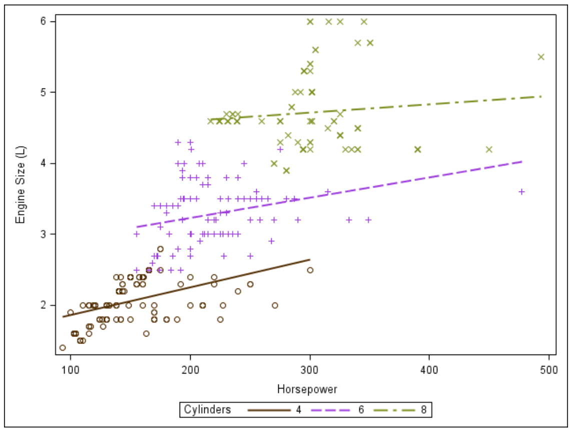


Figure 9. Target colors achieved

Notice that the colors of both the lines and markers changed as a result of this style modification. This happens because the lines and markers both use GraphData*n* for their colors.

THE %MODSTYLE MACRO

SAS provides a macro for making updates to the GraphData*n* style elements. The macro is called %modstyle. Consider the following sample call:

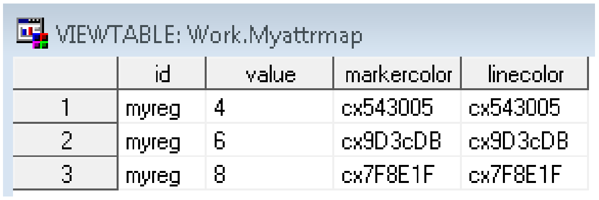
```
%modstyle
   (name=MyStyle                              ⇦ Name the new style being created.
   ,parent=Listing                            ⇦ Use STYLES.LISTING as a starting point.
   ,type=CLM                                  ⇦ See Appendix 2 for more details on this parameter.
   ,colors=cx543005 cx9D3CDB cx7F8E1F         ⇦ These colors go with gcdata4, gcdata5, and gcdata6.
   );
```

This macro call produces the same resulting style as the style modification in the previous section. While there is less typing involved, you do lose a little bit of flexibility when using the macro. For instance, you cannot reference element + attribute pairs (e.g., GraphColors('gcdata4')).

DISCRETE ATTRIBUTE MAPS

Beginning in 9.3, SAS introduced discrete attribute map datasets. Similar in many respects to an annotate dataset, a discrete attribute map dataset provides instructions to SGPLOT for how to associate colors (and other visual attributes) with specific groups. As is the case with annotate datasets, the discrete attribute map dataset uses reserved variable names.

Let's recreate the above grouped plot (with brown, purple, and pea green) using a discrete attribute map. We begin by creating a dataset. The reserved variables we need are ID, VALUE, MARKERCOLOR, and LINECOLOR.



VIEWTABLE: Work.Myattrmap

	id	value	markercolor	linecolor
1	myreg	4	cx543005	cx543005
2	myreg	6	cx9D3cDB	cx9D3cDB
3	myreg	8	cx7F8E1F	cx7F8E1F

Figure 10. Attribute map dataset

Having created this dataset, we now need to make reference to it within SGPLOT. We need to do this in two places: DATTRMAP= and ATTRID=.

```
proc sgplot data=cylinders_sub dattrmap=MyAttrMap;
   reg x=horsepower y=enginesize / group=cylinders attrid=myreg;
run;
```

The DATTRMAP= option tells SGPLOT the name of the discrete attribute map dataset. Because multiple maps can be contained within a single dataset, we also need the ATTRID= option, which tells SGPLOT which records within the dataset are of interest for the REG plot.

A complete set of code to produce the brown, purple, and pea green regression plot with discrete attribute maps might appear as follows:

```
data MyAttrMap;
   retain id "myreg";
   input value $ markercolor $;
   linecolor = markercolor;
   datalines;
4 cx543005
6 cx9D3cDB
8 cx7F8E1F
;
run;

proc sgplot data=work.cars
            dattrmap=MyAttrMap;
   reg x=horsepower y=enginesize /
      group=cylinders attrid=myreg;
   where cylinders in (4 6 8);
run;
```

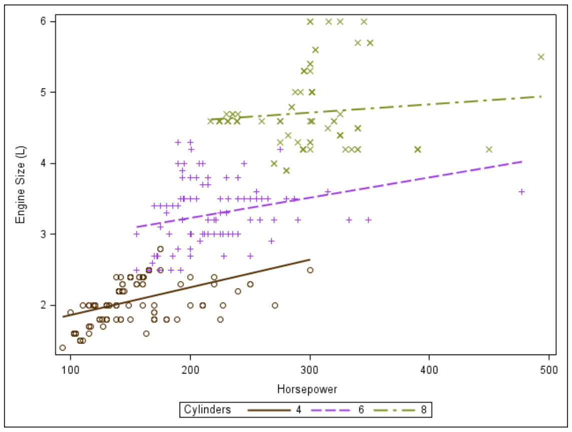
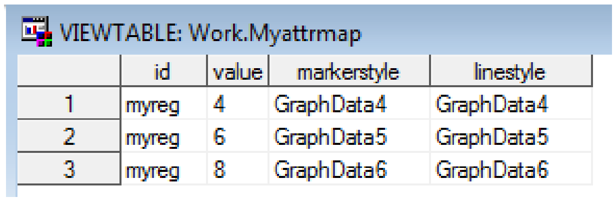


Figure 11. Target colors using attribute maps

Discrete attribute maps have an interesting feature; they have the reserved variable names MARKERSTYLE and LINESTYLE. These allow us to make references to style element in our discrete attribute map datasets.

```
data MyAttrMap;
   retain id "myreg";
   length markerstyle $10;
   input value $ markerstyle $;
   linestyle = markerstyle;
   datalines;
4 GraphData4
6 GraphData5
8 GraphData6
;
run;

proc sgplot data=work.cars
            dattrmap=MyAttrMap;
   reg x=horsepower y=enginesize /
      group=cylinders attrid=myreg;
   where cylinders in (4 6 8);
run;
```


VIEWTABLE: Work.Myattrmap

	id	value	markerstyle	linestyle
1	myreg	4	GraphData4	GraphData4
2	myreg	6	GraphData5	GraphData5
3	myreg	8	GraphData6	GraphData6

Figure 12. Attribute map dataset with styles

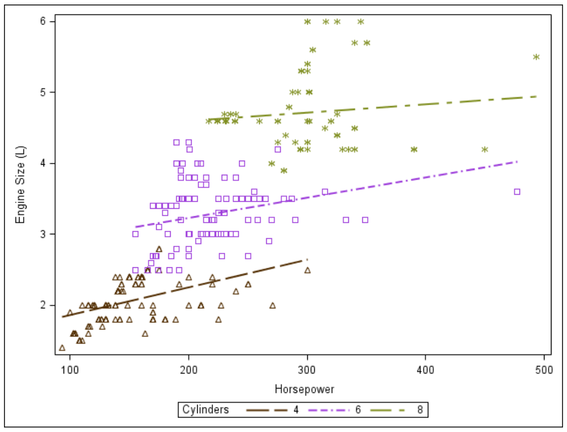
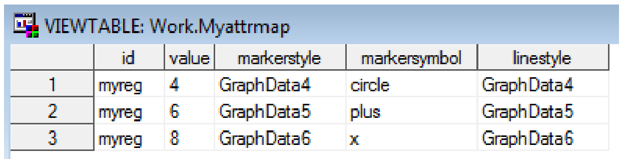


Figure 13. Shapes and patterns changed – oops!

You will notice one curious side-effect of this approach: the symbols and line patterns changed! This is because when we reference a style element we get ALL of the attributes that correspond to that element, not just the color. If you don't like this side-effect, there is a way around it: the reserved variable MARKERSYMBOL.

```
data MyAttrMap;
   retain id "myreg";
   length value $1 markerstyle $10
      markersymbol $10;
   input  value $  markerstyle $
      markersymbol $;
   linestyle = markerstyle;
   datalines;
4 GraphData4 circle
6 GraphData5 plus
8 GraphData6 x
;
run;
```


VIEWTABLE: Work.Myattrmap

	id	value	markerstyle	markersymbol	linestyle
1	myreg	4	GraphData4	circle	GraphData4
2	myreg	6	GraphData5	plus	GraphData5
3	myreg	8	GraphData6	x	GraphData6

Figure 14. Attribute map dataset with overrides

But that only takes care of the symbols. We might also want to change the line types back. So, what's the point? The point is that using MARKERSTYLE and LINESTYLE is convenient, but this convenience does come at a small price. Sometimes you're better off using hardcoded color values.

STYLE MODIFICATIONS VS. DISCRETE ATTRIBUTE MAPS

Which is better: style modifications or discrete attribute maps? That's mostly a matter of personal preference. Though there is one consideration that might tip your opinion on one direction or the other: missing data.

Consider our grouped regression plot with 4, 6, and 8 cylinder vehicles. We are going to reproduce that plot using both style modifications and discrete attribute maps. However, we are going to remove the 6-cylinder vehicles and notice the slight difference in results.

Style Modifications

```
proc template;
   define style MyStyle;
      parent = Styles.Listing;
      class GraphData1 / contrastcolor = GraphColors('gcdata4');
      class GraphData2 / contrastcolor = GraphColors('gcdata5');
      class GraphData3 / contrastcolor = GraphColors('gcdata6');
   end;
run;

ods listing style=MyStyle;

proc sgplot data=work.cars;
   reg x=horsepower y=enginesize /
      group=cylinders;
   where cylinders in (4 8);
run;
```

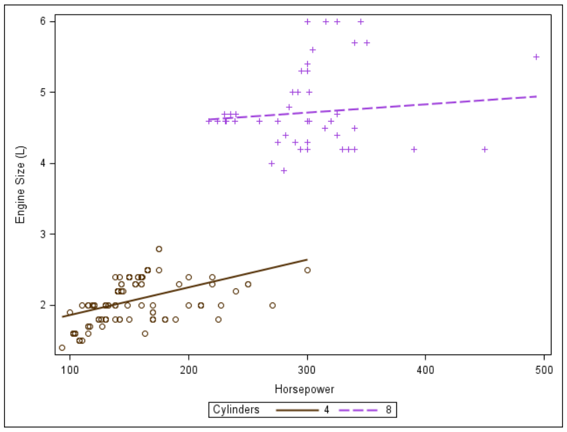


Figure 15. Missing groups and style modifications

Note how the 8-cylinder group changed from pea green to purple. This is because the 8-cylinder group is now the second group in the legend, and hence gets its attributes from GraphData2.

Discrete Attribute Maps

```
data MyAttrMap;
   retain id "myreg";
   input value $ markercolor $;
   linecolor = markercolor;
   datalines;
4 cx543005
6 cx9D3cDB
8 cx7F8E1F
;
run;

proc sgplot data=work.cars
            dattrmap=MyAttrMap;
   reg x=horsepower y=enginesize /
      group=cylinders attrid=myreg;
   where cylinders in (4 8);
run;
```

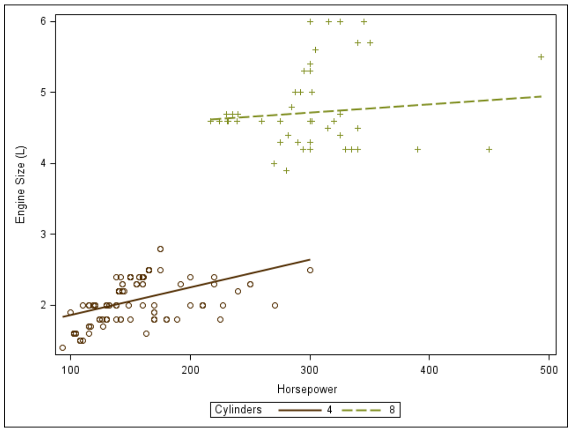


Figure 16. Missing groups and attribute maps

Note how the 8-cylinder group remained pea green. This is because the color pea green is associated with the value 8 in the discrete attribute map.

And the winner is?

If you prefer that specific values of the group variable be associated with specific colors, then use discrete attribute maps. However, if you prefer that specific colors are used in a specific order, independent of the values of the group variable, then use style modifications.

THE STYLEATTRS STATEMENT

Beginning in 9.4, SAS introduced the STYLEATTRS statement. This statement allows you to specific your colors as a space-separated list.

```
proc sgplot data=work.cars;
   styleattrs datacontrastcolors=
      (cx543005 cx9D3CDB cx7F8E1F);
   reg x=horsepower y=enginesize /
      group=cylinders;
   where cylinders in (4 6 8);
run;
```

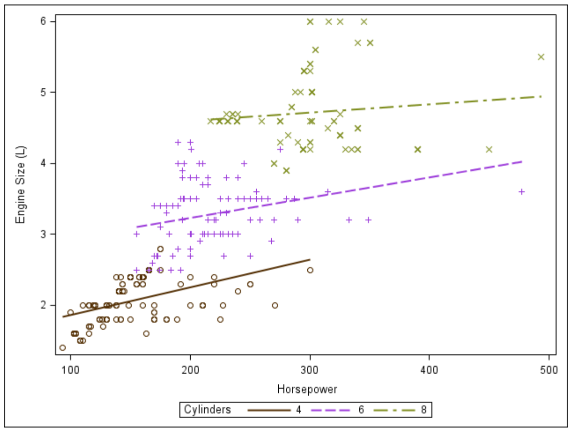


Figure 17. Target colors through styleattrs

The STYLEATTRS statement is conceptually similar to the %modstyle statement in that (a) it modifies the style and (b) you are not allowed to reference element + attribute pairs.

GROUPED DATA SUMMARY

For grouped data we looked at four ways to change colors.

Style Modifications

```
proc template;
   define style MyStyle;
      parent = Styles.Listing;
      class GraphData1 / contrastcolor = GraphColors('gcdata4');
      class GraphData2 / contrastcolor = GraphColors('gcdata5');
      class GraphData3 / contrastcolor = GraphColors('gcdata6');
   end;
run;

ods listing style=MyStyle;

proc sgplot data=work.cars;
   reg x=horsepower y=enginesize / group=cylinders;
   where cylinders in (4 6 8);
run;
```

The %Modstyle Macro

```
%modstyle
   (name=MyStyle
   ,parent=Listing
   ,type=CLM
   ,colors=cx543005 cx9D3CDB cx7F8E1F
   );

ods listing style=MyStyle;

proc sgplot data=work.cars;
   reg x=horsepower y=enginesize / group=cylinders;
   where cylinders in (4 6 8);
run;
```

Discrete Attribute Maps

```
data MyAttrMap;
   retain id "myreg";
   input value $ markercolor $;
   linecolor = markercolor;
   datalines;
4 cx543005
6 cx9D3cDB
8 cx7F8E1F
;
run;

proc sgplot data=work.cars dattrmap=MyAttrMap;
   reg x=horsepower y=enginesize / group=cylinders attrid=myreg;
   where cylinders in (4 6 8);
run;
```

STYLEATTRS

```
proc sgplot data=work.cars;
   styleattrs datacontrastcolors=(cx543005 cx9D3CDB cx7F8E1F);
   reg x=horsepower y=enginesize / group=cylinders;
   where cylinders in (4 6 8);
run;
```

CONCLUSION

Color management in ODS Graphics is not quite as straightforward as it was back in the day with classic SAS/GRAPH. And while there aren't literally 50 ways to change your colors (yet!), there are quite a few ways to go about managing your colors. Each of the color management techniques is useful given the right set of circumstances and no one method is better than all the others. Do not pick a single method and try to force it to work in every situation. Instead, make the initial investment to learn each of these color management methods and it will pay off in the long run.

REFERENCES

Kuhfeld, Warren. 2010. *Statistical Graphics in SAS: An Introduction to the Graph Template Language and the Statistical Graphics Procedures*: Cary, NC: SAS Institute Inc.

SAS Online Documentation: Style Elements and Attributes

SAS Online Documentation: Style Template Modification Macro

SAS Online Documentation: SG Attribute Map Data Sets

SAS Online Documentation: STYLEATTRS Statement

Graphically Speaking blog: Consistent Group Colors

CONTACT INFORMATION

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

Name:	Shane Rosanbalm
Enterprise:	Rho, Inc.
Address:	6330 Quadrangle Drive
City, State ZIP:	Chapel Hill, NC 27514
Work Phone:	919-595-6273
E-mail:	shane_rosanbalm@rhoworld.com
Web:	graphics.rhoworld.com

APPENDIX 1

RGB (red-green-blue) color codes define a color by combining red, green, and blue colors in different ratios. Color names are in the form CXrrggbb where:

- CX indicates to SAS that this is an RGB color specification.
- rr is the red component.
- gg is the green component.
- bb is the blue component.

The components are given as hexadecimal numbers in the range 00 through FF (0% to 100%). Each hexadecimal number indicates how much of the red, green, or blue is included in the color. Lower percentage values are darker and higher values are lighter. This scheme allows for up to 256 levels of each color component (more than 16 million different colors).

Examples of RGB color values include:

Color	RGB Value
Red	CXFF0000
Green	CX00FF00
Blue	CX0000FF
White	CXFFFFFF
Black	CX000000

APPENDIX 2

The TYPE= parameter in the %modstyle macro specifies how the new style cycles through colors, markers, and line styles listed. Consider the following call to the macro:

```
%modstyle
   (name=MyStyle
   ,parent=Listing
   ,type=CLM
   ,colors=blue purple orange
   ,markers=star triangle diamond
   );
```

The result of this call would be GraphData1 having blue stars, GraphData2 having purple triangles, and GraphData3 having orange diamonds. Using different values of TYPE will result in different color/marker combinations for each GraphData*n*. For instance:

TYPE=	Description	Results
CLM	Cycles through colors, line styles, and markers simultaneously.	Blue star, purple triangle, orange diamond, *repeat these 3*
LMbyC	Fixes line style and markers, cycles through colors, and then moves to the next line style and marker.	Blue star, purple star, orange star, blue triangle, purple triangle, orange triangle, blue diamond, purple diamond, orange diamond, *repeat these 9*
CbyLM	Fixes color, cycles through line style and marker, and then moves to the next color.	Blue star, blue triangle, blue diamond, purple star, purple triangle, purple diamond, orange star, orange triangle, orange diamond, *repeat these 9*