Visual Styles for V9 SAS® Output

Jeff Cartier, SAS Institute Inc., Cary, NC

ABSTRACT

In Version 9, SAS Institute will provide an expanded set of Output Delivery System (ODS) styles that can be applied to graphical output as well as text output. This paper illustrates how easy it is to apply these new styles to your SAS/Graph programs. It will also provide specific recommendations on how to adjust your SAS/Graph coding to take full advantage of style definitions. You will also see how to create your own style definitions.

USING ODS STYLES

In addition to the 17 ODS styles provided in Version 8.2, there are 16 new styles in V9:

Analysis	Curve	Magnify	Sketch
Astronomy	Education	Money	Statistical
Banker	Electronics	RSVP	Torn
BlockPrint	Gears	Science	Watercolor

Many of the new styles offer graphical visual effects such color gradients, transparency, texture maps, shadow effects and anti-aliasing on text. To see graphical style effects, the graphics device driver must be set to ACTIVEX or JAVA (or to ACTXIMG or JAVAIMG, the new non-interactive versions of the interactive ACTIVEX and JAVA drivers). This simple program illustrates the how styles affect both graphical and tabular output:

```
ods html file='class.html' style=curve;

goptions reset=all dev=activex;
axis1 label=("Height (Mean)");
proc gchart data=sashelp.class;
  vbar3d age / sumvar=height type=mean
               discrete raxis=axis1;
  run;
quit;

proc means data=sashelp.class maxdec=1;
  class age;
  var height;
run;
ods html close;
```

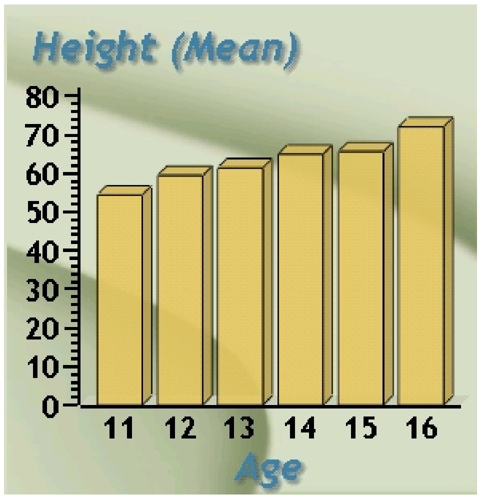
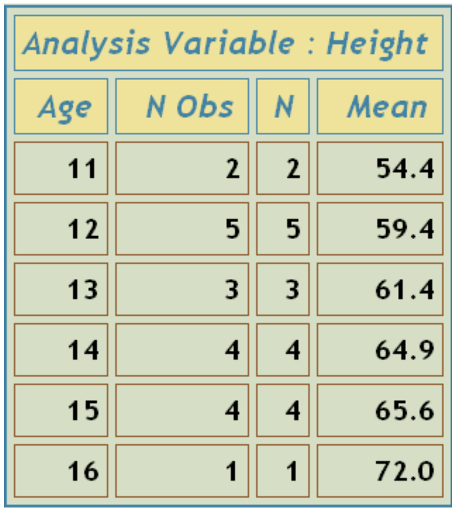


Figure1: Graphical Output – Curve Style

The Curve style uses a background image for the entire graphics area. Notice there is some transparency set on the chart which allows the image to "bleed" through. There is also a drop-shadow effect employed on the axis labels and thicker axis lines and tick marks. Notice that the fonts and colors used for the tabular output coordinate nicely with the graph:

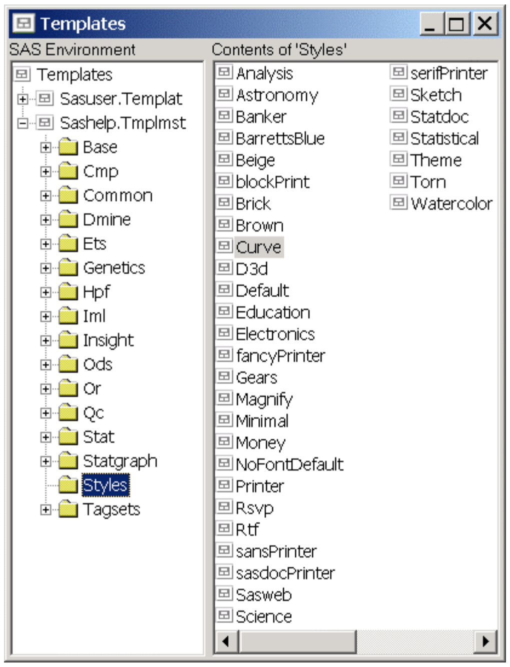


Analysis Variable : Height			
Age	N Obs	N	Mean
11	2	2	54.4
12	5	5	59.4
13	3	3	61.4
14	4	4	64.9
15	4	4	65.6
16	1	1	72.0

Figure2: Tabular Output – Curve Style

SUPPLIED STYLES

To view the supplied ODS styles, issue the ODSTEMPLATE command from your Display Manager session. If you have not created any of your own styles, you will see a single node for SASHELP.TMPLMST under the TEMPLATES tree. Expand this node to see all supplied template folders. Select the STYLES folder to display its contents.



Templates

SAS Environment

- Templates
  - Sasuser.Templat
  - Sashelp.Tmplmst
    - Base
    - Cmp
    - Common
    - Dmine
    - Ets
    - Genetics
    - Hpf
    - Iml
    - Insight
    - Ods
    - Or
    - Qc
    - Stat
    - Statgraph
    - Styles
    - Tagsets

Contents of 'Styles'

Analysis, Astronomy, Banker, BarrettsBlue, Beige, blockPrint, Brick, Brown, Curve, D3d, Default, Education, Electronics, fancyPrinter, Gears, Magnify, Minimal, Money, NoFontDefault, Printer, Rsvp, Rtf, sansPrinter, sasdocPrinter, Sasweb, Science, serifPrinter, Sketch, Statdoc, Statistical, Theme, Torn, Watercolor

Figure 3: Templates Window – Supplied Styles

ANATOMY OF A STYLE DEFINITION

A style is defined by a SAS program. You can double-click on the any style item to view a program that can create it. The program code for styles is not complex but it can be lengthy. To modify or create styles it is important to understand the structure of these programs.

TEMPLATE PROCEDURE

PROC TEMPLATE is used to create various kinds of template stores (more on this later). Examples of template stores are STYLE, TABLE, and TAGSET. This paper will only deal with STYLE templates.

The general form of a program that creates a style is this:

```
proc template;
   define style <directory.>styleName;
      parent = <directory.>parentStyle;

      replace elementName /
           attribute = value
           attribute = value
           ...
      ;

      style elementName <from parentElement> /
           attribute = value
           attribute = value
           ...
      ;
   end;
run;
```

The DEFINE statement creates a new template. STYLE is type of template we are creating. The name of the style comes next (more on this later). Notice that the DEFINE statement requires an END statement. For example, to define a style named CURVE:

```
   define style styles.curve;
       /* sub-statements */
   end;
```

The sub-statements most commonly used within the DEFINE STYLE block are STYLE, REPLACE, and PARENT.

STYLE STATEMENT

The STYLE statement defines a ***style element*** which is a named set of logically-related style attributes. A ***style attribute*** is a name-value pair. (ODS uses the terms *element* and *attribute* in the same way markup languages like HTML and XML do.)
For example:

```
   style Table /
      background = colors('tablebg')
      rules = ALL
      frame = BOX
      cellpadding = 5
      cellspacing = 5
      bordercolor = colors('tableborder')
      borderwidth = 2
   ;
```

Here the TABLE element is being defined. The forward slash begins the declaration of its attributes. All the attribute names used here are reserved and documented. The attributes RULES and FRAME have only a few possible values which are also reserved and documented. The syntax for assigning color values will be explained shortly.

So far, this pretty straightforward. What make templates very interesting is that they support inheritance.

PARENT STATEMENT

Inheritance provides a mechanism for one template definition to use another template definition.

```
   parent = styles.default;
```

All supplied styles include this PARENT statement (except for STYLES.DEFAULT which has no parent). When defining your own styles, you do not need to use inheritance, but it certainly makes your work easier if you do. There are over 100 style *elements* in STYLES.DEFAULT. Each of the other supplied styles overrides specific elements definitions rather than redefining all the style elements from scratch. If the current style does not define one or elements, these elements are picked up from the parent. Learning how exploit inheritance will make your style definitions much shorter and more readable. Any existing style can be used as a parent. It is recommended that you become familiar with the supplied styles and pick one of them as the parent of your style.

Inheritance is used not only at the template level, but also at the element level. Here a partial listing of a few existing styles elements within STYLES.DEFAULT (indentation implies inheritance):

```
   Container  (abstract: root of all containers)
      Output (abstract: output presentation)
         Table (table output)
         Graph (graph output)
      Cell (abstract: tables cells)
         Data (data in table cells)
      HeadersAndFooters (abstract: for table)
         Header (table headers)
         Footer  (table footers)
   Fonts (list of fonts)
   Color_List (list of color names & RGB values)
   Colors (links parts of output & color names)

   GraphCharts (all charts in graph)
   GraphAxisLines (all axis lines)
```

The keyword FROM indicates inheritance syntactically. The element following FROM is the parent element. For example:

```
   define style Output from Container /..;
   define style Table from Output /..;
   define style Cell from Container /..;
   define style Data from Cell / ..;
```

This form of inheritance allows you to define a new element and include all the attributes of a parent element.

As mentioned before, if you do not a declare an element, the parent's element is used. If you declare an element, you should decide whether you want inheritance or not.

```
   /* inheritance:                     */
   /* element picks up any additional */
   /* attributes from parent element  */

   style Table from Table /
      rules = COLS
      borderwidth=1;
```

```
/* no inheritance:           */
/* element is self-contained */

style Table /
   background = colors('tablebg')
   rules = COLS
   frame = BOX
   cellpadding = 5
   cellspacing = 5
   bordercolor = colors('tableborder')
   borderwidth = 1
;
```

What happens if you don't include all possible attributes and you don't inherit them? Some default value will be used. Even if the default for an attribute is documented, it is recommended that you completely redefine the element when not using inheritance.

REPLACE STATEMENT

Both STYLE and REPLACE sub-statements control style element inheritance. They augment or override the attributes of a particular style element. You can think of the REPLACE statement as replacing the definition for the like-named element in the parent style definition. The REPLACE statement doesn't actually change the parent style definition, but PROC TEMPLATE builds the child style definition as if it had changed the parent. All style elements that inherit attributes from this style element inherit the ones that are specified in the REPLACE statement, not the ones that are used in the parent style definition. The REPLACE statement can further reduce element coding but it provides no unique functionality that can't be obtained with STYLE statements.

TEMPLATE STORES AND STYLE NAMES

In Version 7, SAS introduced a new file type called an item store (which uses the .SASBITM extension). Item stores are utility files that are used to persist hierarchical information not suitable for SAS data files or SAS Catalog entries. The SAS Registry is an item store that is manipulated with the Registry Editor and Proc Registry. A template store is a specific kind of item store managed with the Template window and Proc Template. SAS supplies one template store named SASHELP.TMPLMST as part of base SAS. This file contains all supplied style definitions (as well as other types of templates used by ODS). You can think of this file as containing internal directories (and subdirectories) that contain various item types. One of the directories in TMPLMST is STYLES and within this directory are a number of items of type STYLE. See Figure 3.

When you refer to an item in a template store, you specify just the internal directory path and item name, but not the name of the template store itself. For example, STYLES.CURVE refers to the internal directory STYLES and the item named CURVE. Is there anything special about the directory named STYLES? No, we could define CORP.DEPT.STYLE1 that refers to a style named STYLE1 in DEPT directory which is a subdirectory of CORP. As a matter of fact, any template store could have styles with these names.

So how does ODS know which template store to use? The ODS PATH statement queries or sets a search sequence for template stores. This statement shows the current search sequence:

```
ods path show;
```

It displays this text in the SAS log:

```
Current ODS PATH list is:
 1. SASUSER.TEMPLAT(UPDATE)
 2. SASHELP.TMPLMST(READ)
```

What does this mean? If you specify a style say, STYLE=STYLES.CURVE, ODS will look for it first in SASUSER.TEMPLAT and then in SASHELP.TMPLMST. This is true whether simply accessing an existing style or creating a new one:

```
proc template;
  define style Styles.Curve;
     /* sub-statements */
  end;
```

This means to store STYLES.CURVE in SASUSER.TEMPLAT because it is the first template store that can be updated.

So how do you define a style so that others can use? Typically, you designate a SAS library for the template store and then use the PATH statement to indicate the template store:

```
libname dept "some-public-location";
proc template;
  path dept.sasstyles(write);
  define style Styles.Curve;
       /* sub-statements */
  end;
```

This assures that STYLES.CURVE physically resides in DEPT.SASSTYLES (without the PATH statement, STYLES.CURVES would be written to SASUSER.TEMPLAT).

To give people access to the style, you need to establish the search sequence with the ODS PATH statement:

```
libname dept "some-public-location"
             access=readonly;
libname corp "some-other-public-location"
             access=readonly;
ods path reset;
ods path (prepend) corp.styles(read)
                   dept.sasstyles(read);
```

This code could be placed in an autoexec file. It establishes this search path:

```
 1. CORP.STYLES(READ)
 2. DEPT.SASSTYLES(READ)
 3. SASUSER.TEMPLAT(UPDATE)
 4. SASHELP.TMPLMST(READ)
```

Now, if an ODS statement specifies a style like

```
ods html file='html-file' style=styles.curve;
```

ODS will search these template stores in order and use the first definition of STYLES.CURVE it finds.

GRAPHICAL STYLE FEATURES IN VERSION 9

Everything that has been said about Proc Template syntax applies to Versions 8 and 9. What has changed in Version 9 is the addition of 16 styles mentioned earlier. All of these new styles incorporate a large number of graphically-related style elements and attributes that better coordinate graphical and tabular output. There are two tables at the end of the paper that summarize the new style elements and style attributes.

The remainder of this paper will show how customize the appearance of graphs in your ODS output by adapting supplied styles.

We will create a new style STYLES.MYCURVE using the supplied STYLES.CURVE as our starting point (parent).

```
   proc template;
      define style Styles.myCurve;
         parent = styles.Curve;

        /* style statements */
        /* defined below    */

      end;
   run;
```

CHANGING GRAPH SIZE

This is the definition for the GRAPH element in STYLES.DEFAULT which is inherited by all 16 new styles (none of them override this definition). The GRAPH element also can control the size of the graph area using the attributes OUTPUTWIDTH and OUTPUTHEIGHT. The defaults for these attributes are 640px and 480px.

```
   style Graph from Output /
      cellpadding = 0
      cellspacing = 0
      background = colors('docbg');
```

By adding these attributes to the GRAPH element, you can change the size of all graphs:

```
   /* add to mycurve definition */
   style Graph from Graph /
      outputwidth = 500px
      outputheight = 400px;
```

CHANGING GRAPH TEXT ATTRIBUTES

There are two style elements that affect graphical text:

GraphLabelText – axis and legend labels
GraphValueText – axis and legend values

The CURVE style set these attributes for GraphLabelText:

```
   style GraphLabelText from GraphLabelText /
      dropshadow = on
      font = fonts('GraphLabelFont');
```

This had the effect of making the text somewhat “fuzzy”. To sharpen the text appearance, disable DROPSHADOW:

```
   /* add to mycurve definition */
   style GraphLabelText from GraphLabelText /
      dropshadow = off;
```

The CURVE style did not set DROPSHADOW = ON for the GraphValueText element so we will make no changes.

CHANGING CHART ATTRIBUTES

One of the more interesting attributes is TRANSPARENCY. This affects how much you can “see through” portions of a chart. The CURVE style uses transparency (see Figure 1):

```
   style GraphCharts from GraphCharts /
      transparency = 0.1;
```

The other elements that use transparency are

GraphWalls – axis walls (in 3D mode)
GraphFloor – axis floor (in 3D mode)
GraphLegendBackground

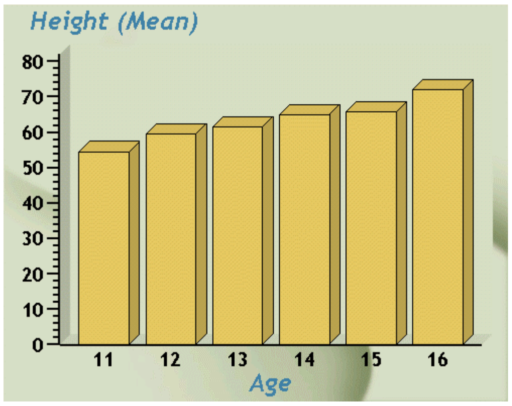


Figure 4: Transparency = 0 for Chart, Walls and Floor

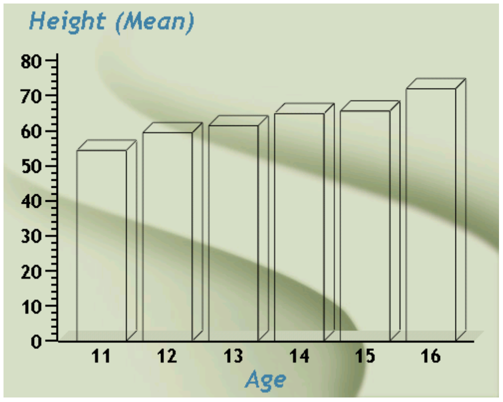


Figure 5: Transparency = 1 for Chart, Walls and Floor

As you can see, the close the transparency is to 1, the more you will see the graph background.

CHANGING GRAPH BACKGROUND

The CURVE style uses this definition for GraphBackground:

```
replace GraphBackground /
   background = colors('docbg')
   image = "C:/Program Files/SAS/
              SAS System/9.0/common/textures/Curve.jpg"
    just = Right
    vjust = Bottom;
```

CURVE.JPG is one of several image files supplied with base SAS. The location of these files is defined by the new system option TEXTURELOC=. By simply changing the image to one of your own, you can get many interesting effects. Figure 5 shows a small corporate logo appearing

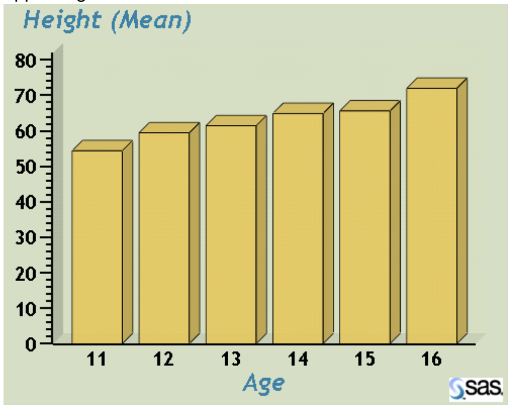


Figure 5: Using an Image for Background

The IMAGE attribute displays an image using its actual size. The JUST attribute (LEFT, CENTER, RIGHT) and VJUST attribute (TOP, MIDDLE, BOTTOM) control its position. A related attribute is BACKGROUNDIMAGE. This differs from IMAGE in that it specifies an image to be stretched to fit the entire background. VJUST and JUST do not apply to this.

Another possible background effect is create a ***gradient***.

```
   replace GraphBackground /
      gradient_direction = "YAxis"
      endcolor = colors('docbg')
      startcolor = colors('headerbg');
```

There are three attributes affecting a gradient. The GRADIENT_DIRECTION can be vertical (“Yaxis”) or horizontal (“Xaxis”). You could replace the GRAPHWALLS element with same attributes to obtain an even better transition effect.

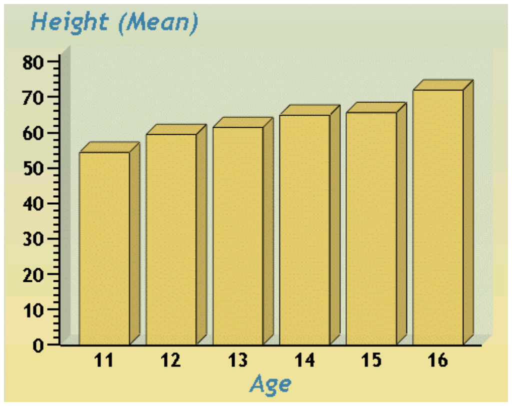


Figure 6: Using a Vertical Gradient for Background

CHANGING GRID LINES

You can control the line style, color and width of grid lines. The values used for LINESTYLE are the same as SAS/GRAPH uses. This element does not turn on grid lines; it only controls their display properties when they are displayed. You must turn on the gridlines with a SAS/GRAPH chart option such as AUTOREF.

```
   style GraphGridLines /
      lineStyle=3
      foreGround = dmgray
      outputWidth = 2;
```

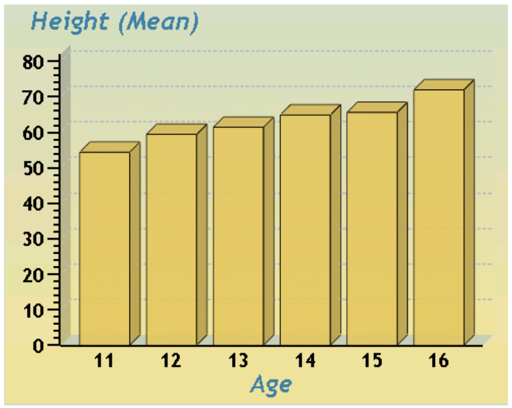


Figure 7: Setting Grid Line Style

ADJUSTING SAS/GRAPH PROGRAMS FOR STYLES

Recall the SAS/GRAPH coding of our original program:

```
   goptions reset=all dev=activex;
   axis1 label=("Height (Mean)");
   proc gchart data=sashelp.class;
     vbar3d age / sumvar=height type=mean
                  discrete raxis=axis1;
     run;
   quit;
```

Notice that this program does NOT contain any of the numerous SAS/GRAPH options that change colors or fonts of the output. If any of these options did appear in the program, they would have precedence over any style attribute that may address the same feature.

Here a list of some common SAS/GRAPH options that affect the same graph features that graphical styles do.

GOPTIONS
    HSIZE VSIZE XPIXELS YPIXELS IBACK
    CTEXT CTITLE CBY CBACK CSYMBOL CPATTERN
    FTEXT FTITLE FBY HTEXT HTITLE HBY

AXIS
    COLOR STYLE WIDTH LABEL=(COLOR FONT HEIGHT)
    VALUE=(COLOR FONT HEIGHT)

LEGEND
    CBACK CFRAME CBORDER CSHADOW FWIDTH
    LABEL=(COLOR FONT HEIGHT)
    VALUE=(COLOR FONT HEIGHT)

SYMBOL
    CO CI CV FONT VALUE HEIGHT WIDTH

PATTERN
    COLOR IMAGE

TITLE / FOOTNOTE
    COLOR FONT HEIGHT JUSTIFY

GCHART – VBAR/HBAR
    CAXIS CFRAME COUTLINE CTEXT IFRAME
    LAUTOREF CAUTOREF

GPLOT – PLOT
    CAXIS CFRAME CTEXT FRAME
    CAUTOHREF CAUTOVREF LAUTOVREF LAUTOHREF

CONCLUSION

In Version 9, you will be able to control the appearance of graphs as well as tables in your ODS output. SAS will provide 16 new styles. You can define your own styles to create many interesting effects.

The two tables that follow document the Version 9 style elements and attributes. These tables also relate style elements and attributes to SAS/GRAPH syntax features so you can more easy adjust your programs to use more (or less) of the style definition in any particular program.

Style Element	Affects	Style Attributes	SAS/Graph Override
Graph	entire graphics area	**OutputWidth OutputHeight**	GOPTIONS XPIXELS= YPIXELS=
GraphCharts	all charts in graphics area	**Transparency**	
GraphBackground	background color or image of the graph	**Gradient_Direction, StartColor, EndColor, ForeGround; BackGround, BackGroundImage, Image, Vjust, Just**	GOPTIONS CBACK= IBACK= IMAGESTYLE=
GraphLegendBackground	background color or image of the legend	**Gradient_Direction, StartColor, EndColor, ForeGround; BackGround, BackGroundImage, Image, Vjust, Just**	LEGEND statement
DropShadowStyle	drop shadow color for text	**DropShadow, ForeGround**	
GraphLabelText	text for axis labels and legend title	**ForeGround, DropShadow, Font_Face, Font_Size, Font_Weight, Font_Style**	AXIS statement LABEL=( ) suboptions; LEGEND statement; LABEL=( ) suboptions
GraphValueText	text for axis tick marks values and legend entries	**ForeGround, DropShadow, Font_Face, Font_Size, Font_Weight, Font_Style**	AXIS statement LABEL=( ) suboptions; LEGEND statement; LABEL=( ) suboptions
GraphGridLines	grid / reference lines	**ForeGround, LineStyle, OutputWidth**	AXIS statement COLOR= , STYLE=, WIDTH= options
GraphAxisLines	axis lines and tick marks	**ForeGround, LineStyle, OutputWidth**	Procedure CAXIS=; AXIS statement COLOR=, STYLE=, WIDTH=
GraphBorderLines	frame around axis area and legend	**ForeGround, LineStyle, OutputWidth**	Chart FRAME option, LEGEND statement CBORDER=
GraphOutlines	lines that outline bars, map regions, etc.	**ForeGround, LineStyle, OutputWidth**	PATTERN statement
GraphWalls	wall color or image	**Transparency, StartColor, EndColor, Gradient_Direction, Background, BackgroundImage, Image**	Procedure action statement IFRAME= IMAGESTYLE= CFRAME= options
GraphFloor	floor color or image	**Transparency, StartColor, EndColor, Gradient_Direction, Background, BackgroundImage, Image**	
TwoColorRamp	Maps with continuous response	**StartColor, EndColor**	
GraphData1 – GraphData12	graphics primitives related to data items: color, fill, marker	**Foreground, ContrastColor, MarkerSymbol, MarkerSize, LineStyle, OutputWidth**	GOPTIONS COLORS=( ); SYMBOL statement; PATTERN statement

Table1 Version 9 Graphical Style Elements

Style Attribute	Type	Affects	Examples	SAS/Graph Override
OutputWidth	dimension	width of graph; line thickness	OutputWidth=400px; OutputWidth=2	GOPTIONS XPIXELS=
OutputHeight	dimension	height of graph	OutputHeight=300px	GOPTIONS YPIXELS=
Transparency	number: 0.0=opaque 1.0=transparent	Graphic background s area	Transparency=0.2	
Background	color	background color of the graph, walls, or floor	Background= colors('docbg'); Background='blue'	GOPTIONS CBACK=
Foreground	color	color of text, data item	Foreground= colors('docfg');	GOPTIONS CBACK=
ContrastColor	color	alternate color for maps	ContrastColor="red"	
LineStyle	integer: 1 = solid line 2-46= dashed line	background color or image of the legend	LineStyle=2	SYMBOL statement LINE= option; AXIS statement STYLE=option
DropShadow	boolean: On or Off	drop shadow color for text	DropShadow,=on DropShadow=off	
BackGroundImage	string: image file (including path)	image that can be stretched, but not positioned in graph, chart, walls, floor	Image="//server/images/ myimage.gif"	GOPTIONS IBACK= IMAGESTYLE=FIT
Image	string: image file (including path)	image that can be positioned, but not stretched in graph, chart, walls, floor	Image="//server/images/ myimage.gif"	GOPTIONS IBACK= IMAGESTYLE=TILE
Just	justifcation: center, left, or right	image horizontal positioning	Just=left	
Vjust	justifcation: top, middle, bottom	image vertical positioning	Vjust=bottom	
Gradient_Direction	string: use "Xaxis" for left-to-right; "Yaxis" for top-to-bottom	graph background, legend background, charts, walls, floors	Gradient_Direction= "Xaxis"	
StartColor	color: initial color used with gradient	graph background, legend background, charts, walls, floors	StartColor="yellow"	
EndColor	color: final color used with gradient	graph background, legend background, charts, walls, floors	StartColor="red"	
MarkerSymbol	string	markers related to data values	MarkerSymbol="circle"; MarkerSymbol="square"	SYMBOL statement VALUE= option
MarkerSize	dimension	marker size related to data values	MarkerSize=5px; MarkerSize=3%	SYMBOL statement HEIGHT= option
Font_Face	string	value text, label text	Font_Face="Helvetica"	PATTERN statement
Font_Size	fontsize: 1 to 7 or dimension	value text, label text	Font_Size=3; Font_Size=10pt	
Font_Width	fontwidth: normal, narrow, wide, etc.	value text, label text	Font_Width=narrow	
Font_Weight	fontweight: light, medium, bold, etc.	value text, label text	Font_Weight=bold	
Font_Style	fontstyle: italic, roman, slant	value text, label text	Font_Style=italic	
Font	Aggregate definition in parentheses	value text, label text	Font=("arial, helvetica", 4, medium roman)	

Table2 Version 9 Graphical Style Attributes