

**It's All in the Presentation**

Jeff Cartier, SAS Institute Inc., Cary, NC

**ABSTRACT**

In Version 9, ODS styles have been extended to include elements that affect graphical procedure output as well as tabular output. This paper will show how to easy it is to apply any of the new supplied style definitions to SAS/GRAPH, SAS/STAT, SAS/ETS output. You will also see how SAS/GRAPH coding and supplied STATGRAPH templates interact with information supplied by a style. By adjusting source programs, you can control exactly the level of style information that contributes to final output.

**INTRODUCTION**

Today, most SAS users are taking advantage of ODS to produce documents containing output from SAS procedures. Users are aware of the existence of ODS styles and how a style can be specified to alter the fonts, colors, and other appearance aspects of their tabular output. The good news is that in Version 9, graphical output can now be formatted in a similar fashion with an ODS style.

**ODS AND SAS/GRAPH OUTPUT**

Many of the new styles offer graphical visual effects such color gradients, transparency, texture maps, shadow effects and anti-aliasing on text. To see style effects for SAS/GRAPH procedures, the graphics device driver must be set to ACTIVEEX, JAVA, ACTXIMG, or JAVAIMG (the first two drivers create interactive controls and the last two drivers create images). The following program illustrates how easy it is to use a style with ODS and how the style produces a coordinated visual effect on both graphical and tabular output:

```
ods html file='class.html' style=default;
goptions reset=all border device=actximg;
proc gchart data=sashelp.class;
  vbar3d sex / sumvar=height type=mean
  outside=mean;
run; quit;
proc means data=sashelp.class maxdec=1
  nonobs mean;
  class sex;
  var height;
run;
ods html close;
```

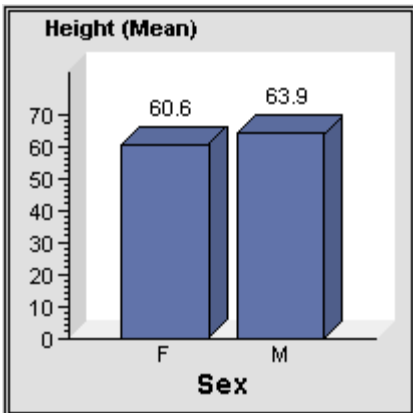


Figure 1: Graph – DEFAULT Style

Analysis Variable : Height	
Sex	Mean
F	60.6
M	63.9

Figure 2: Table – DEFAULT Style

By changing only the value for the STYLE= option, you can create an entirely different appearance for both the graph and table. This is the result for STYLE=RSVP:

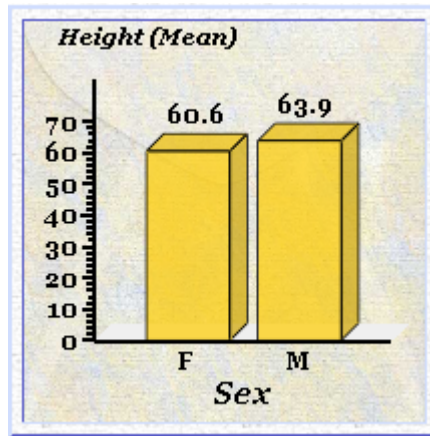


Figure 3: Graph – RSVP Style

Analysis Variable : Height	
Sex	Mean
F	60.6
M	63.9

Figure 4: Table – RSVP Style

Notice that the SAS/GRAPH coding did not include any options that specified fonts or colors to be used. This information was all derived from the style definition. If such options were present, the colors or fonts in the program would be used in place of the corresponding style values.

The example program use ODS HTML destination, but it could have used any other ODS destination just as well, such as PDF, RTF, or PRINTER. All produce different output files with the same visual content.

If you have not used any of the four client drivers before, here are some other things you should know:

- A client technology (ActiveX or Java) is used to render the graph, not SAS/GRAPH. Consequently, there may be some differences in appearance between client and non-client drivers.
- The only supported procedures are GCHART, GPLOT, GMAP, GCONTOUR, and G3D. You can also use SAS/GRAPH annotation coding with these procedures.
- Titles and footnotes appear but are not part of the graph.
- The interactive client drivers (ACTIVEVX and JAVA) enable you to change the graphical display via context menus.
- There are some differences in which SAS/GRAPH options are supported by Java and ActiveX technologies. See the SAS/GRAPH documentation for details.

It should be emphasized that when using SAS/GRAPH procedures with ODS, a SAS/GRAPH device driver is always in effect. You must use one of the "client drivers" (ACTIVEVX, JAVA, ACTXIMG, or JAVAIMG) to see the effect of a style. All other drivers are "style unaware". For example, if you were to use any of the GIF family of drivers, the ODS output would look like just like the GRSEG output, but as a GIF image. Its visual appearance is affected only by SAS/GRAPH coding and not by any ODS style.

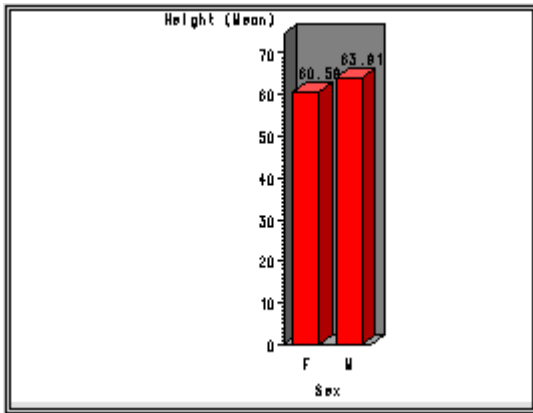


Figure 5: Graph using DEVICE=GIF260

### ODS AND STATGRAPH OUTPUT

In Version 9, SAS/STAT and SAS/ETS procedures can produce graphs when used with ODS. Here is an example of using PROC LIFETEST to produce a survival plot showing the Hall-Wellner band.

```
ods html file='lifetest.html' style=mystyle;
ods graphics on;
proc lifetest data=mydata;
  time Months;
  survival confband=all plots=(hwb) ;
run;
ods graphics off;
ods html close;
```

Here are some things to know about the graphs produced by SAS/STAT and SAS/ETS:

- Graphs are produced by entirely Java technology. They do not require installation of SAS/GRAPH and do not support any form of SAS/GRAPH coding, including device drivers.
- Graphs are not produced by default. You must enable / disable graphics with the ODS GRAPHICS statement.
- Statistical procedures supply one or more ODS STATGRAPH templates that specify a predefined graph. You simply instruct the procedure which graphs to produce.
- The supplied STATGRAPH templates use ODS styles to set colors, fonts, and as well as other appearance features such as markers and line styles.

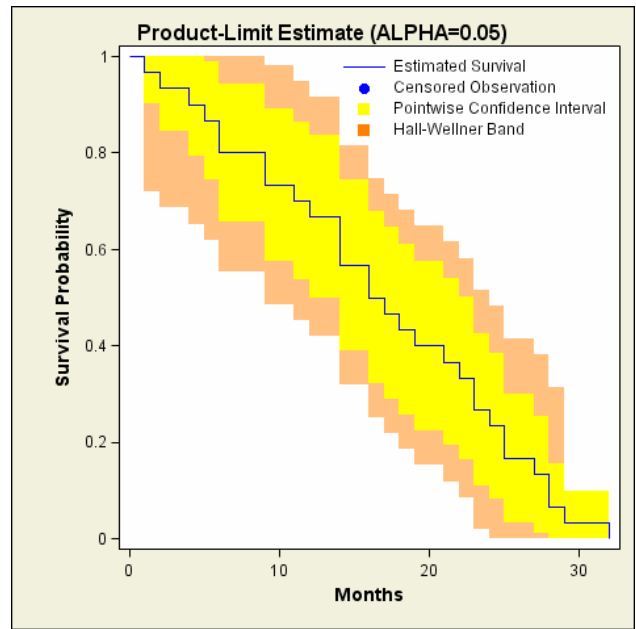


Figure 6: StatGraph - Custom Style

### 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 STYLES to display the contents of this folder. In addition to the 17 ODS styles provided in Version 8, there are 16 new styles in Version 9:

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

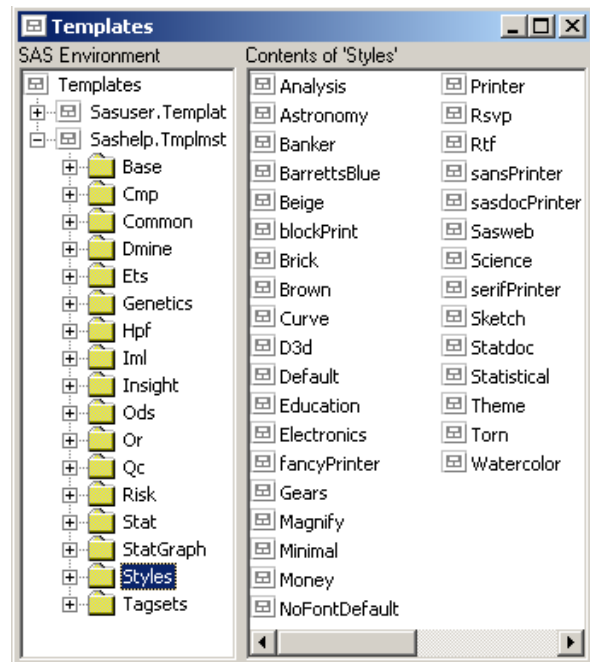


Figure 7: Templates Window – Supplied Styles

## ANATOMY OF AN ODS STYLE DEFINITION

An ODS style is defined by a SAS program. You can browse or edit the source program for any style from the Templates window. The code for defining styles is not complex but it can be lengthy. To modify or create a style it is important to understand the structure of a style program and how it can inherit information from other styles.

### TEMPLATE PROCEDURE

PROC TEMPLATE is used to create various kinds of template stores. Examples of template stores are STYLE, TABLE, and STATGRAPH. All ODS templates stores can be viewed from the Template window. This paper will only deal with STYLE and STATGRAPH template types.

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. Notice that the DEFINE statement requires an END statement. For example, to define a style named MYCURVE:

```
define style styles.mycurve;
  /* 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 is pretty straightforward. What makes styles 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;
```

Most 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 more elements, these elements are picked up from the parent. Learning how to 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 custom style.

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

```
Container (root of all containers)
Output (output presentation)
Table (tabular output)
Graph (graphical output)
```

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

```
define style Container /*;
define style Output from Container /*;
define style Table from Output /*;
define style Graph from Output /*;
```

This form of inheritance allows you to define a new element and automatically include all the attributes of a parent element. As mentioned before, if you do not declare an element, the same named parent template element is used. If you do 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.

### DEFINING COLORS AND FONTS

A major portion of any style definition establishes colors and fonts for specific areas of the output. A style establishes lists of colors and fonts and assigns each value an "abstract" name. These names are referenced in other style elements. Here are some shortened examples of such lists:

```
style fonts "Fonts for style" /
  'docFont' = ("Arial, Helvetica, Helv",3);

style GraphFonts "Fonts for graphs" /
  'GraphValueFont' = ("Arial",10pt)
  'GraphLabelFont' = ("Arial",14pt,Bold);

style color_list "Colors for default style" /
  'fgA1' = cx000000 /*foreground */
  'bgA1' = cxF0F0F0 /* background */
  'fgA' = cx002288
  'bgA' = cxE0E0E0;

style colors "Abstract colors" /
  'tableborder' = color_list('fgA1')
  'tablebg' = color_list('bgA1')
  'docfg' = color_list('fgA')
  'docbg' = color_list('bgA');

style GraphColors "Abstract graph colors" /
  'glabel' = cx000000
  'gaxis' = cx000000
  'gdata1' = cx6173A9
  'gdata2' = cx8DA642
  'gdata3' = cx98341C
  'gdata4' = cxFDC861;
```

Here are some examples of how these lists are used:

```
style container /
  font = Fonts('DocFont')
  foreground = colors('docfg')
  background = colors('dogbg');

style Table from output /
  background = colors('tablebg');

style GraphBackground /
  background = colors('docbg');

style GraphAxisLines /
  foreground = GraphColors('gaxis');

style GraphLabelText /
  font = GraphFonts('GraphLabelFont');
```

Notice that various style elements may reference the same color or font. If you want to change fonts or colors in a style, it is recommended that you change only the font or color values (but not their abstract names) in elements Fonts, GraphFonts, Color\_List, and GraphColors. This ensures a consistent effect is created across tables and graphs. Color values can be specified in many ways including SAS color names, RGB or HLS. Consult the ODS documentation for examples. When testing the appearance of modified colors and fonts, you should include both graphs and tables to assure that you get the desired consistency for both forms of output.

### 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 the appearance of graphical and tabular output. There are tables at the end of the paper that summarize the new style elements and style attributes. Use these tables to help understand which style elements affect which part of the graph. Figure 8 shows the names of some of the graphical style elements indicates the areas of a graph affected by each.

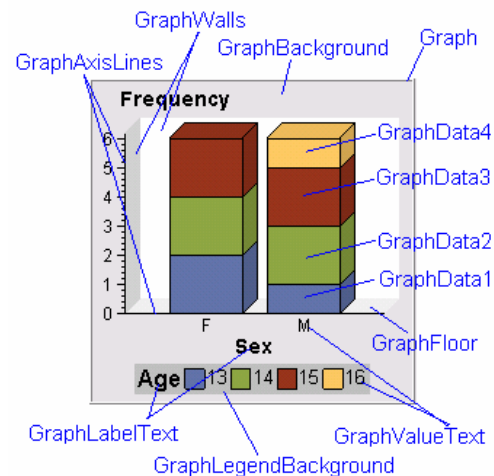


Figure 8: Commonly Used Graphical Style Elements

Most of the element names are self-explanatory. The elements GraphData1 – GraphData12 are used to associate a style attributes with sets of data values. Figure 8 shows a subgrouped bar chart. The properties for each level of the subgroup variable are obtained from the GraphData elements. These elements can specify not only colors, but also line and marker properties for plots.

The remainder of this paper will show how customize the appearance of graphs (both SAS/GRAPH and STATGRAPH) in your ODS output by adapting supplied styles.

We will modify the supplied STYLES.CURVE as our starting point (parent) and name our style STYLES.MYCURVE:

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

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

  end;
run;
```

## CHANGING GRAPH SIZE

By adding the OUTPUTWIDTH and OUTPUTHEIGHT attributes to the GRAPH element, you can change the size of all graphs produced with this style. The defaults for these attributes are OUTPUTWIDTH=640px and OUTPUTHEIGHT=480px. For SAS/GRAPH output these attributes are overridden if you include values for GOPTIONS XPIXELS= YPIXELS=.

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

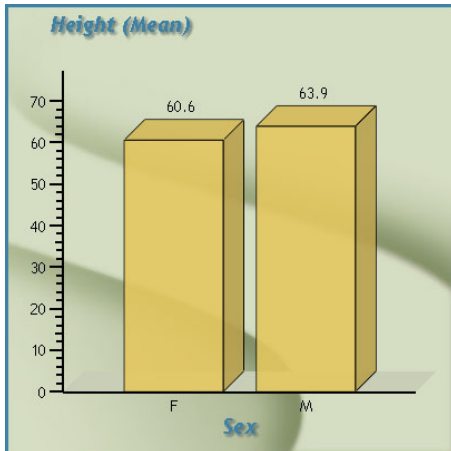


Figure 9: Curve Style

## CHANGING GRAPH TEXT ATTRIBUTES

The two style elements that affect most graphical text are  
 GraphLabelText – color and font for axis and legend labels  
 GraphValueText – color and font for axis and legend values

The CURVE style set these attributes for GraphLabelText:

```
style GraphLabelText from GraphLabelText /
  dropshadow = on
  /* these are inherited from default */
  foreground = GraphColors('glabel')
  font = fonts('GraphLabelFont');
```

The DropShadow element defines a shadow color and some offsets for shadow size. Rather than change these, you could sharpen the text appearance by simply disabling the drop shadow effect:

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

The CURVE style did not enable the drop shadow effect for the GraphValueText element so no changes are necessary.

## 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 employs transparency with two elements:

```
style GraphCharts from GraphCharts /
  transparency = 0.1;
style GraphWalls from GraphCharts /
  transparency = 1.0;
```

The closer the transparency is to 1, the more you will see through to the graph background.

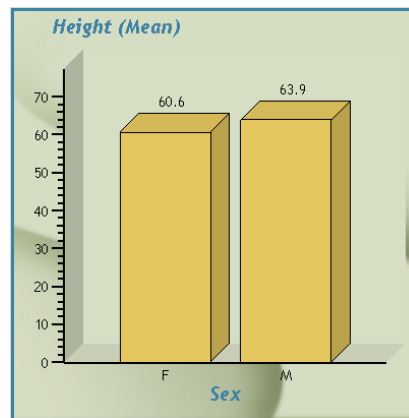


Figure 10: Transparency = 0 for Chart and Walls

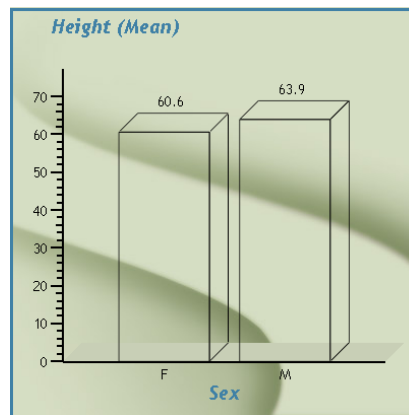


Figure 11: Transparency = 1 for Chart and Walls

## CHANGING GRAPH BACKGROUND

The CURVE style uses this definition for GraphBackground:

```
replace GraphBackground /
  background = colors('docbg')
  image = "Curve.jpg"
  just = Right
  vjust = Bottom;
```

CURVE.JPG is one of several image files supplied with base SAS that are used with style definitions. The location of these files is defined by the system option TEXTURELOC=. You can add your own images to the TEXTURELOC path, and refer to them without path information or you can include the fully-qualified name (or URL) to your own image. Filetypes are not restricted to JPG. Figure 12 shows a corporate logo used for the IMAGE attribute.

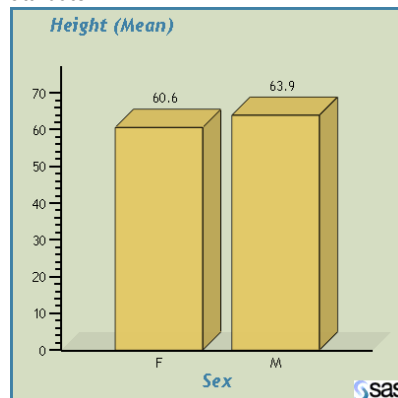


Figure 12: Custom Image for Background

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 BACKGROUNDIMAGE.

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

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

There are three attributes affecting a gradient. The GRADIENT\_DIRECTION can be vertical ("Yaxis") or horizontal ("Xaxis"). The graph in Figure 13 also set these attributes for GraphWalls and GraphData1-GraphData12.

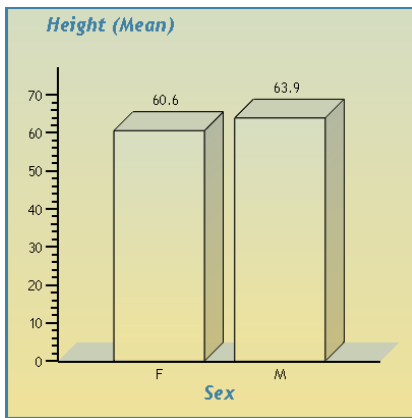


Figure 13: Using a Vertical Gradient for Background

## ADJUSTING SAS/GRAPH PROGRAMS FOR STYLES

Recall the SAS/GRAPH coding of our original program:

```
goptions reset=all border dev=actximg;
proc gchart data=sashelp.class;
  vbar3d sex / sumvar=height type=mean
  outside=mean;
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 were to appear in the program, they would have precedence over any style attribute that may address the same feature.

In general, a style does not enable a SAS/GRAPH feature – you must do this in your SAS/GRAPH program. Examples of this include GOPTIONS BORDER | NOBORDER to enable or disable a border around the graph. If you enable the border, the Output and Graph styles elements control its visual characteristics (which are coordinated with the table border in the supplied styles). Another example is the FRAME | NOFRAME option used by GCHART and GPLOT action statements. In general, you only need to enable or disable this feature. If you use CFRAME to turn on the frame you will also override the color defined in the style. Here is a list of some common SAS/GRAPH options that affect the same graph features that graphical styles do:

```
GOPTIONS
  COLORS 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/VBAR3D/HBAR3D
  CAXIS CFRAME COUTLINE CTEXT IFRAME
  LAUTOREF CAUTOREF
GPLOT – PLOT
  CAXIS CFRAME CTEXT FRAME
  CAUTOHREF CAUTOVREF LAUTOVREF LAUTOHREF
```

## STATGRAPH TEMPLATES AND STYLES

As mentioned earlier, you will be able to create one or more graphs for statistical procedures, independent of SAS/GRAPH.

```
ods graphics on;
ods html file="robustreg.html"
  style=mystatistical;

proc robustreg data=growth HISTplot DDplot;
  model GDP = LFG GAP EQP NEQ /
  diagnostics(all) leverage;
run;

ods html close;
ods graphics off;
```

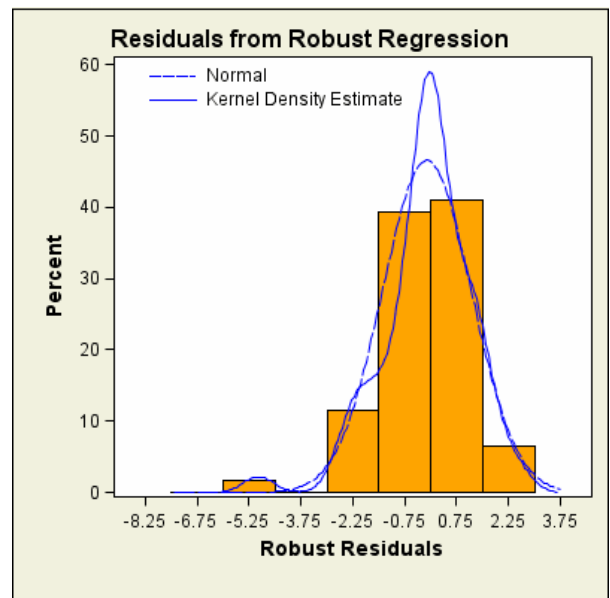


Figure 13: HISTPlot Output from PROC ROBUSTREG

STATGRAPH output uses the same graphical style elements and attributes that SAS/GRAPH does (there are a few style attributes that STATGRAPH does not support, such as those for image and gradient backgrounds). In the DEFAULT style there are several additional elements that apply only to STATGRAPH. Two of these elements are shown here:

```
style StatGraphData from GraphComponent /
  markersize = 3px
  markersymbol = "CircleFilled"
  linestyle = 1
  contrastcolor = GraphColors('gcdata')
  foreground = GraphColors('gdata');

style StatGraphFitLine from GraphComponent /
  transparency = 0.00
  linethickness = 2px
  linestyle = 1
  contrastcolor = GraphColors('gcfit')
  foreground = GraphColors('gfit');
```

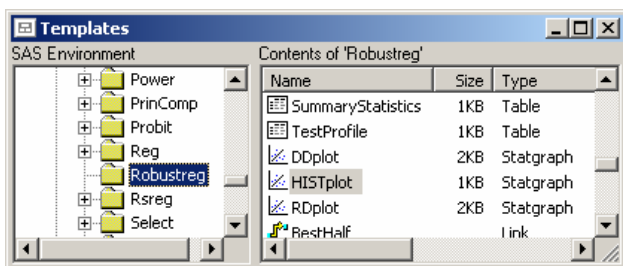


Figure 14: Templates for PROC ROBUSTREG

If you look at a STATGRAPH template that defines a graph, you will see visual features being set by references to style attributes (**element : attribute**). This form of coding enables a user to modify the style definition to get a customized presentation without modifying the STATGRAPH template.

```
proc template;
  define statgraph Stat.Robustreg.HISTplot;
    layout Gridded;
      EntryTitle
        "Residuals from Robust Regression";
    layout Overlay /
      xaxisopts=(label="Robust Residuals")
      yaxisopts=(label="Percent");

      Histogram RRESIDUAL / fill=true
        fillcolor=StatGraphData:foreground;

      Density RRESIDUAL / KERNEL() name="kern"
        linecolor=
          StatGraphFitLine:contrastcolor
        linethickness=
          StatGraphFitLine:linethickness
      LegendLabel=
        "Kernel Density Estimate";

      Density RRESIDUAL / name="norm"
        linecolor=
          StatGraphFitLine:contrastcolor
        linethickness=
          StatGraphFitLine:linethickness
      linepattern=dashlong
      LegendLabel="Normal";

      DiscreteLegend "norm" "kern" /
        hAlign=left vAlign=top;
    EndLayout;
  EndLayout;
```

```
end;
run;
```

## 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 easily adjust your programs to use more (or less) of the style definition in any particular program.

## CONTACT INFORMATION

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

Jeff Cartier  
 SAS Institute Inc.  
 Cary NC 27513  
 Work Phone:  
 Email: [Jeff.Cartier@sas.com](mailto:Jeff.Cartier@sas.com)

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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

**Table 1 Version 9 Graphical Style Elements**

Note: Style elements include all recognized attributes.  
Style elements do not have to define all attributes.

Style Attribute	Type	Affects	Examples
<b>OutputWidth</b>	dimension	width of graph; line thickness	OutputWidth=400px; OutputWidth=2
<b>OutputHeight</b>	dimension	height of graph	OutputHeight=300px
<b>Transparency</b>	number: 0.0=opaque 1.0=transparent	Chart, walls, floor and legend backgrounds	Transparency=0.2
<b>Background</b>	color	background color of the graph, walls, or floor	Background=colors('docbg');
<b>Foreground</b>	color	color of text, data fill item	Foreground=colors('docfg');
<b>ContrastColor</b>	color	alternate color for maps; marker color	ContrastColor=red
<b>LineStyle</b>	integer: 1 = solid line 2-46= dashed line	borders, axis lines, grid, reference, model, confidence lines	LineStyle=2
<b>LineThickness</b>	color	color of line	LineColor=blue
<b>DropShadow</b>	boolean: On or Off	drop shadow color for text	DropShadow=on DropShadow=off
<b>BackgroundImage</b>	string: image file (including path)	image that can be stretched, but not positioned in graph, chart, walls, floor	Image="//server/images/myimage.gif"
<b>Image</b>	string: image file (including path)	image that can be positioned, but not stretched in graph, chart, walls, floor	Image="//server/images/myimage.gif"
<b>Just</b>	justification: center, left, or right	image horizontal positioning	Just=left
<b>Vjust</b>	justification: top, middle, bottom	image vertical positioning	Vjust=bottom
<b>Gradient_Direction</b>	string: use "Xaxis" for left-to-right; "Yaxis" for top-to-bottom	graph background, legend background, charts, walls, floors	Gradient_Direction="Xaxis"
<b>StartColor</b>	color: initial color used with gradient	graph background, legend background, charts, walls, floors	StartColor=yellow
<b>EndColor</b>	color: final color used with gradient	graph background, legend background, charts, walls, floors	StartColor=red
<b>MarkerSymbol</b>	string	markers related to data values	MarkerSymbol="circle"; MarkerSymbol="square"
<b>MarkerSize</b>	dimension	marker size related to data values	MarkerSize=5px; MarkerSize=3%
<b>Font_Face</b>	string	value text, label text	Font_Face="Helvetica"
<b>Font_Size</b>	fontsize: 1 to 7 or dimension	value text, label text	Font_Size=3; Font_Size=10pt
<b>Font_Width</b>	fontwidth: normal, narrow, wide, etc.	value text, label text	Font_Width=narrow
<b>Font_Weight</b>	fontweight: light, medium, bold, etc.	value text, label text	Font_Weight=bold
<b>Font_Style</b>	fontstyle: italic, roman, slant	value text, label text	Font_Style=italic
<b>Font</b>	Aggregate definition in parentheses	value text, label text	Font=("arial, helvetica", 4, medium roman)

**Table 2 Version 9 Graphical Style Attributes**