Revised and Enhanced from SAS Global Forum Paper 323-2009

Modifying ODS Statistical Graphics Templates in SAS® 9.2

Warren F. Kuhfeld, SAS Institute Inc., Cary, NC

Abstract

With the release of SAS 9.2, over sixty statistical procedures use ODS Statistical Graphics to produce graphs as automatically as they produce tables. This paper reviews the basics of ODS Statistical Graphics and focuses on programming techniques for modifying the default graphs. Each graph is controlled by a template, which is a SAS program written in the Graph Template Language (GTL). This powerful language specifies graph layouts (lattices, overlays), types (scatter plots, histograms), titles, footnotes, insets, colors, symbols, lines, and other graph elements. SAS provides the default templates for graphs, so you do not need to know any details about templates to create statistical graphics. However, with some understanding of the GTL, you can modify the default templates to permanently change how certain graphs are created. Alternatively, you can make immediate and ad hoc changes to specific graphs by using the point-and-click ODS Graphics Editor. This paper presents examples to help you navigate the complexity of the default templates and safely customize elements such as titles, axis labels, colors, lines, markers, ticks, grids, axes, reference lines, and legends.

Introduction

Effective graphics are indispensable for modern statistical analysis. They reveal patterns, differences, and uncertainty that are not readily apparent in tabular output. Graphics provoke questions that stimulate deeper investigation, and they add visual clarity and rich content to reports and presentations. With the release of SAS 9.2, over sixty statistical procedures produce graphs when ODS Graphics is enabled with the following statement:

```
ods graphics on;
```

A few samples of graphs that are automatically produced appear on the next two pages. Each graph has a template, which is a SAS program that provides instructions for creating the graph. SAS provides a template for each graph it creates, so you do not need to know anything about templates to create statistical graphics. However, with just a little knowledge of the graph template language (GTL), you can modify templates to change how your graphs are created. For example, you can change titles, axes, colors, symbols, lines, and other graph elements. Figure 2 displays a scree plot from the FACTOR procedure. Displaying and modifying its template is simple. Figure 3 shows how the plot looks after you modify the template to replace the title “Scree Plot” with “Eigenvalue (λ) Plot”, the X axis label “Factor” with “Factor Number”, and the Y axis label “Eigenvalue” with “λ”. The modification process is explained in detail in the section “Changing Titles and Axis Labels” on page 7.

This paper begins by reviewing some basic principles of ODS and ODS Graphics. Next, it shows you several templates that are used by SAS/STAT® procedures. The earlier templates were chosen because they are among the simplest templates in SAS/STAT, and they illustrate basic principles that can be applied to more complex templates. The remainder of the paper shows examples of modifying more complicated templates. More information can be found in Chapter 21, “Statistical Graphics Using ODS” (*SAS/STAT User’s Guide*).

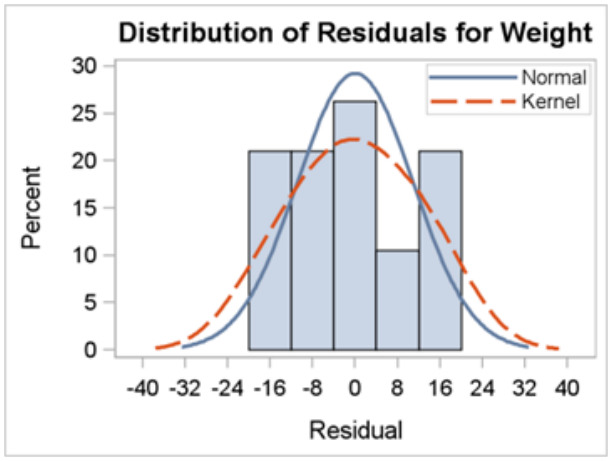
ODS and ODS Graphics Background Material

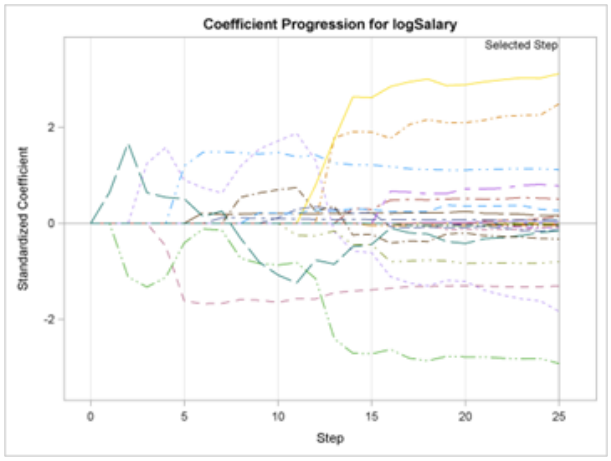
This section presents some background material that you need to know before you change templates.

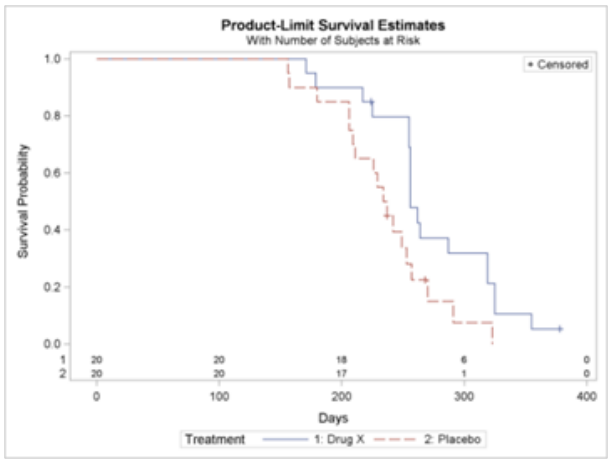
Templates, Libraries, and Item Stores

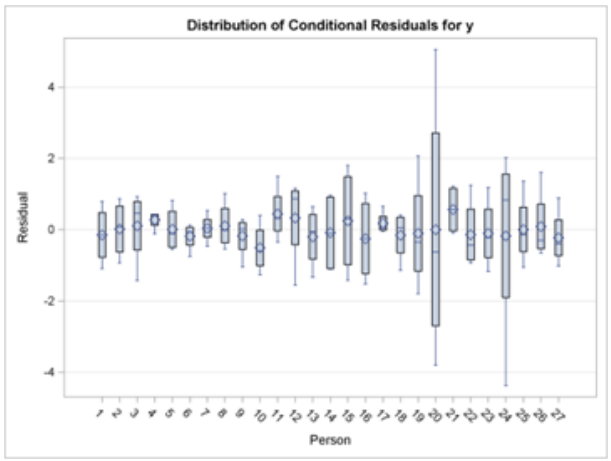
This section uses the KDE (kernel density estimation) procedure to illustrate properties of templates and show where compiled templates are stored. The output from a typical SAS procedure includes a series of tables and graphs. You can use the ODS TRACE statement to list the names and labels of the tables and graphs along with their templates as follows:

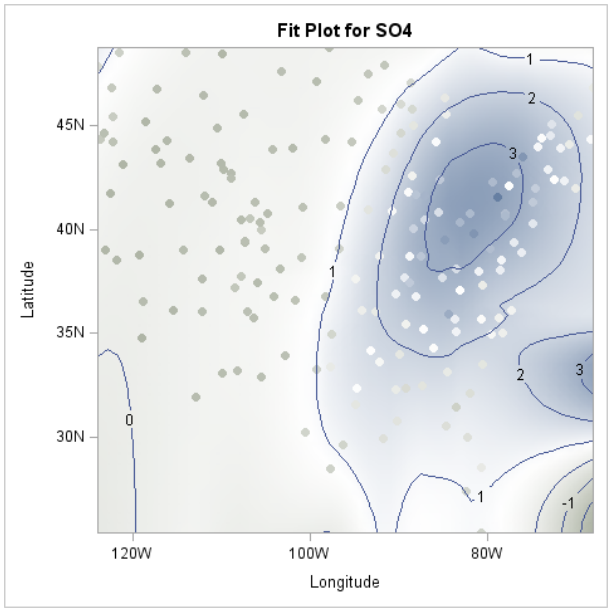
```
ods graphics on;
ods trace on;

proc kde data=sashelp.class;
   bivar height weight / plots=scatter;
run;
```

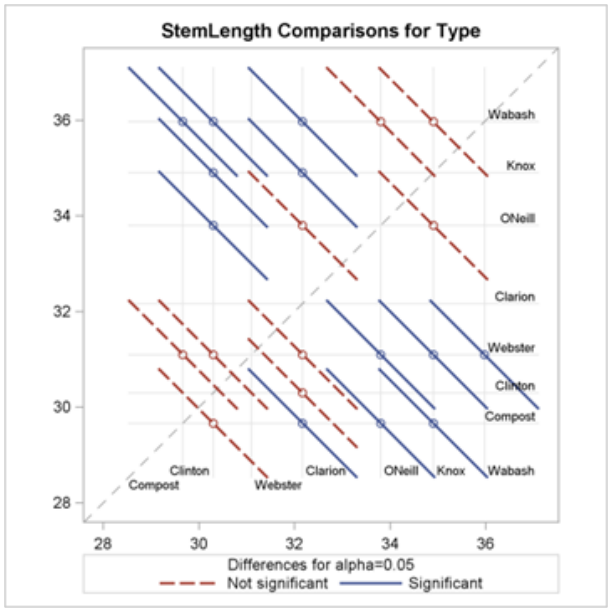

Distribution of Residuals for Weight
Normal
Kernel
Percent
30
25
20
15
10
5
0
-40 -32 -24 -16 -8 0 8 16 24 32 40
Residual

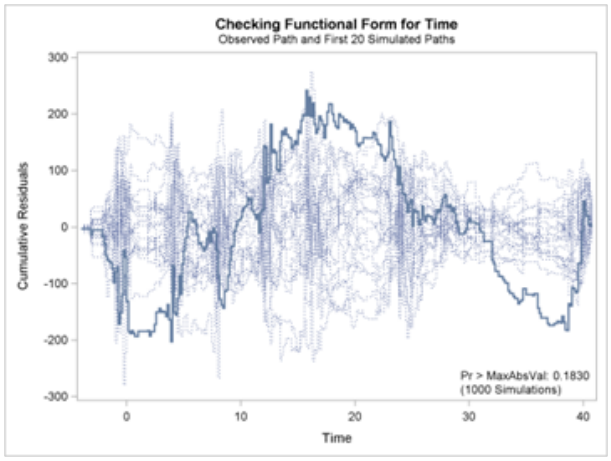

Coefficient Progression for logSalary
Selected Step
Standardized Coefficient
2
0
-2
0 5 10 15 20 25
Step


Product-Limit Survival Estimates
With Number of Subjects at Risk
+ Censored
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
1 20 20 18 6 0
2 20 20 17 1 0
0 100 200 300 400
Days
Treatment 1: Drug X 2: Placebo

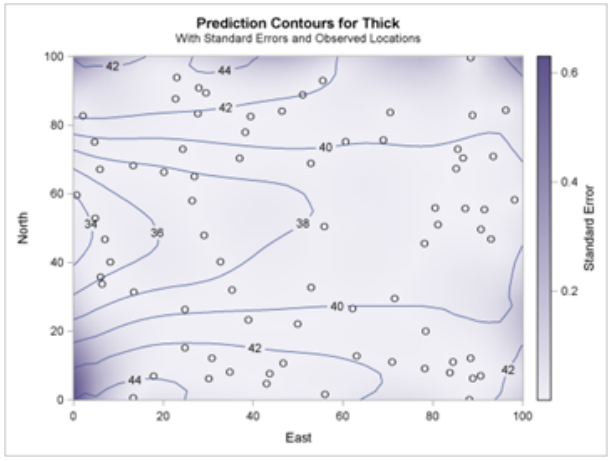

Distribution of Conditional Residuals for y
Residual
4
2
0
-2
-4
Person

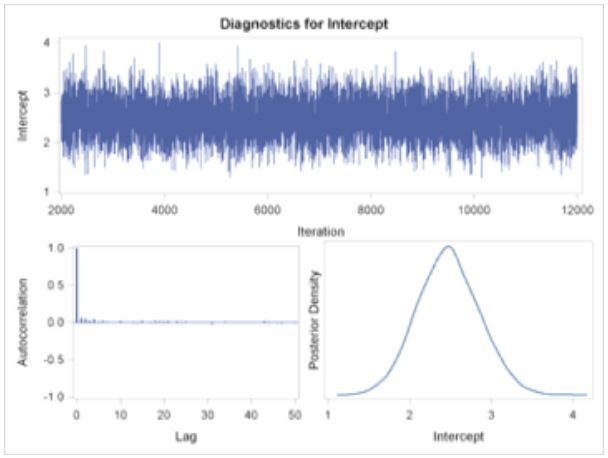

Fit Plot for SO4
45N
40N
35N
30N
Latitude
120W 100W 80W
Longitude

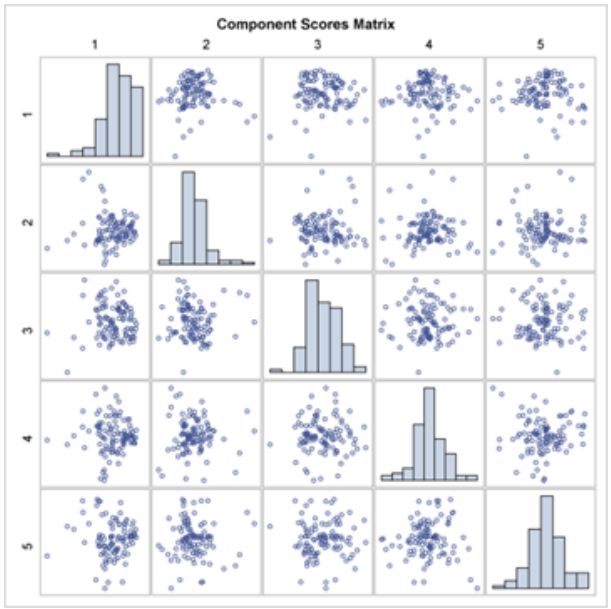

StemLength Comparisons for Type
36
34
32
30
28
Wabash
Knox
ONeill
Clarion
Webster
Clinton
Compost
Differences for alpha=0.05
Not significant
Significant

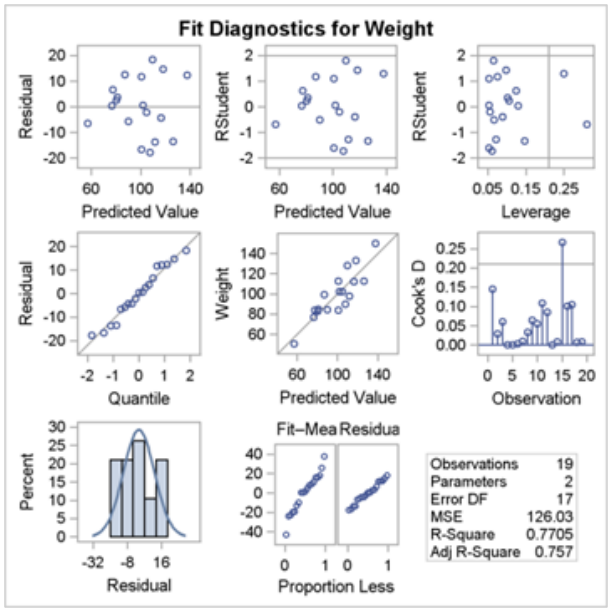

Checking Functional Form for Time
Observed Path and First 20 Simulated Paths
Cumulative Residuals
Time
Pr > MaxAbsVal: 0.1830
(1000 Simulations)


Component Scores
Component 3 (7.733%)
Component 2 (13.64%)
Component 1


Prediction Contours for Thick
With Standard Errors and Observed Locations
North
East
Standard Error


Diagnostics for Intercept
Intercept
Iteration
Autocorrelation
Lag
Posterior Density
Intercept


Component Scores Matrix


Fit Diagnostics for Weight
Residual
Predicted Value
RStudent
Leverage
Quantile
Weight
Cook's D
Observation
Percent
Fit–Mea Residua
Proportion Less
Observations 19
Parameters 2
Error DF 17
MSE 126.03
R-Square 0.7705
Adj R-Square 0.757

A portion of the trace output, which appears in the SAS log, is as follows:

```
Name:        Controls
Template:    Stat.KDE.Controls
Path:        KDE.Bivar1.Height_Weight.Controls

Name:        ScatterPlot
Label:       Scatter Plot
Template:    Stat.KDE.Graphics.ScatterPlot
Path:        KDE.Bivar1.Height_Weight.ScatterPlot
```

A typical graph template name has the form *product.procedure*`.Graphics.`*name*. You can list the source statements for any template by using the TEMPLATE procedure as follows:

```
proc template;
   source Stat.KDE.Graphics.ScatterPlot;
run;
```

The template source statements are as follows:

```
define statgraph Stat.KDE.Graphics.ScatterPlot;
   dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
   BeginGraph;
      EntryTitle "Distribution of " _VAR1NAME " by " _VAR2NAME;
      layout Overlay / xaxisopts=(offsetmin=0.05 offsetmax=0.05)
         yaxisopts=(offsetmin=0.05 offsetmax=0.05);
         ScatterPlot x=X y=Y / markerattrs=GRAPHDATADEFAULT;
      EndLayout;
   EndGraph;
end;
```

The statements and options in this template are discussed in the next section.

You can modify a template by editing the statements, adding a PROC TEMPLATE statement, and submitting the template source to SAS. The compiled templates are stored in a special SAS file called an item store. The default templates that SAS provides are stored in the Sashelp library and the Tmplmst item store. By default, if you submit a template, it is stored in the Sasuser library and the Templat item store. A template stored in this item store persists until you delete the template, the item store, or the entire library.

By default, ODS first searches Sasuser.Templat for templates, and then it searches Sashelp.Tmplmst if it does not find the requested template in Sasuser.Templat. You can modify the path and insert a Work item store in front of the default path in either of the following equivalent ways:

```
ods path work.templat(update) sasuser.templat(update) sashelp.tmplmst(read);
ods path (prepend) work.templat(update);
```

You can see the list of template item stores by submitting the following statement:

```
ods path show;
```

The results are as follows:

```
Current ODS PATH list is:

1. WORK.TEMPLAT(UPDATE)
2. SASUSER.TEMPLAT(UPDATE)
3. SASHELP.TMPLMST(READ)
```

With this path, any template that you submit is stored in Work.Templat, which is deleted at the end of your SAS session. You can see a list of all of the templates that you have modified as follows:

```
proc template;
   list / store=work.templat;
   list / store=sasuser.templat;
run;
```

The following statements illustrate how you can specify a permanent item store for your use and for the use of others:

```
libname mytpl 'C:\MyTemplateLibrary';
ods path (prepend) mytpl.templat(update);
```

Now, when you run PROC TEMPLATE, SAS creates an item store in the directory you specified in the LIBNAME statement. The following statements illustrate how you can list the templates in your item store:

```
proc template;
   list / store=mytpl.templat;
run;
```

You can restore the default template path in either of the following equivalent ways:

```
ods path sasuser.templat(update) sashelp.tmplmst(read);
ods path reset;
```

You can delete any template that you modified (so that ODS finds the default template that SAS supplied) by specifying it in a DELETE statement, as in the following example:

```
proc template;
   delete Stat.KDE.Graphics.ScatterPlot;
run;
```

The Sashelp library is always read-only, so you can safely submit the preceding step even if the template you specify does not exist in Work.Templat or Sasuser.Templat. You can submit the following statements to delete all of the customized templates in Sasuser.Templat so that ODS uses only the templates supplied by SAS:

```
ods path sashelp.tmplmst(read);
proc datasets library=sasuser;
   delete templat(memtype=itemstor);
run; quit;
ods path reset;
```

Before you can delete the item store, you must first remove it from the path. You can optionally restore the path to the default setting when you are finished. It is good practice to store temporary template modifications in Work.Templat so that they are not unexpectedly used in a later SAS session. Alternatively, you can store them in Sasuser.Templat and explicitly delete them when you are done with them. You can find more information about item stores in the documentation section “The Default Template Stores and the Template Search Path” (Chapter 21, *SAS/STAT User’s Guide*).

Data Objects

SAS procedures produce data objects. A data object is a rectangular arrangement of the values (data, statistics, and so on) that are needed to produce a table or a graph. ODS applies a template to the information in a SAS data object to produce a graph. You can output the underlying data object to a SAS data set by using the ODS OUTPUT statement, and you can use the CONTENTS and PRINT procedures to display the results:

```
proc kde data=sashelp.class;
   ods output ScatterPlot=myscatter;
   bivar height weight / plots=scatter;
run;

proc contents p;
   ods select position;
run;

proc print data=myscatter(obs=3);
run;
```

The results are displayed in Figure 1.

Figure 1 Contents of a Data Object

The CONTENTS Procedure

Variables in Creation Order

#	Variable	Type	Len	Label
1	VarNames	Char	17	
2	X	Num	8	Height
3	Y	Num	8	Weight

Obs	VarNames	X	Y
1	Height and Weight	69.0	112.5
2	Height and Weight	56.5	84.0
3	Height and Weight	65.3	98.0

The column names, column labels, and values in the data object provide the variable names, variable labels, and values in the SAS data set.

Styles

ODS styles control the colors and general appearance of all graphs and tables. SAS provides several styles that are recommended for use with statistical graphics. A style is specified in a destination statement as follows:

```
ods listing style=statistical;
```

The STATISTICAL style is the default style in SAS/STAT documentation and in this paper. The default style that you see when you run SAS depends on the ODS destination: the default style for the LISTING destination is LISTING, the default style for the HTML destination is DEFAULT, and the default style for the RTF destination is RTF. You can see the source statements for these style definitions by submitting the following step:

```
proc template;
   source styles.default;
   source styles.statistical;
   source styles.listing;
   source styles.rtf;
run;
```

This step produces a great deal of output, so the full results are not shown. However, a small portion of the DEFAULT style is displayed next:

```
    'gcdata' = cx000000
    'gdata' = cxB9CFE7
    'gconramp3cend' = cxFF0000
    'gconramp3cneutral' = cxFF00FF
    'gconramp3cstart' = cx0000FF

class GraphDataDefault /
   endcolor = GraphColors('gramp3cend')
   neutralcolor = GraphColors('gramp3cneutral')
   startcolor = GraphColors('gramp3cstart')
   markersize = 7px
   markersymbol = "circle"
   linethickness = 1px
   linestyle = 1
   contrastcolor = GraphColors('gcdata')
   color = GraphColors('gdata');
```

This part of the style definition defines the `GraphDataDefault` style element and some of the colors that it uses. This style element defines the default marker, marker size, line style, line thickness, marker and line colors, and other colors.

The marker or symbol size is 7 pixels, the marker is a circle, lines are 1 pixel thick, the line style is 1 (solid), the color is light blue, and the contrast color is black. Colors apply to filled areas, and contrast colors apply to markers and lines.

Figure 2 Default Scree Plot

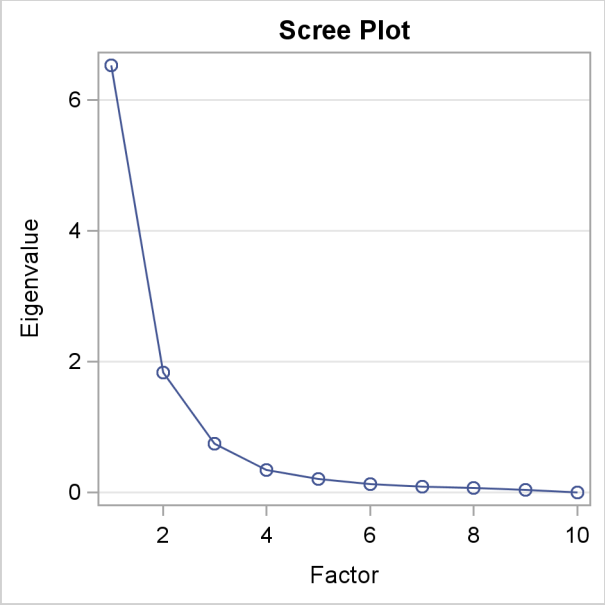
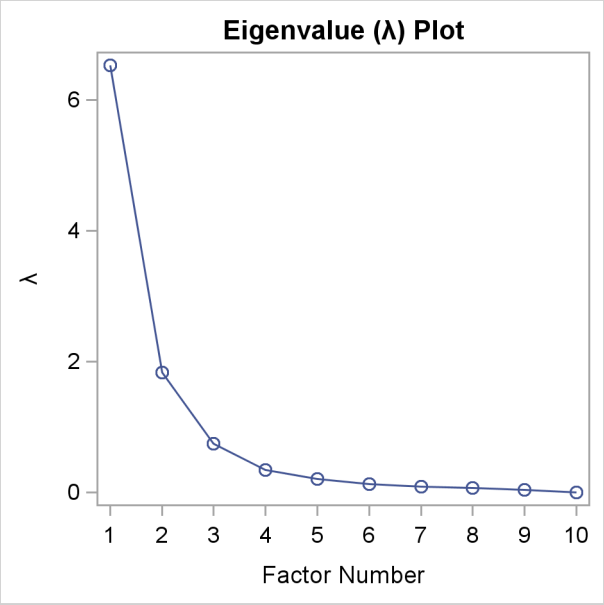


Figure 3 Scree Plot with Modified Title and Axis Labels



A color ramp is used to display a third variable in a scatter plot via color, and the colors range from blue to magenta to red. A value **cx***rrggbb* specifies colors in terms of their red, green, and blue components in hexadecimal where *rr* ranges from 00 (0 = black) to FF (255 = red), *gg* ranges from 00 (0 = black) to FF (255 = green), and *bb* ranges from 00 (0 = black) to FF (255 = blue). You can change styles, create new styles from scratch, and create new styles based on old styles, all by creating or editing a style template. See the documentation section "Styles" (Chapter 21, *SAS/STAT User's Guide*) for more about styles.

SAS/STAT Graph Templates and the Basics of the GTL

This section examines some templates, shows what they have in common and what is different, and explains some common GTL statements and options. The templates that are discussed in this section are chosen because they are small, and they provide complementary insights into template modification. Most templates are more complicated. The rest of this paper assumes that the following statements from the preceding sections are in effect:

```
ods path (prepend) work.templat(update);
ods graphics on;
ods trace on;
```

Changing Titles and Axis Labels

The following step runs PROC FACTOR and produces the eigenvalue (or scree) plot displayed in Figure 2:

```
proc factor data=sashelp.cars plots(unpack)=scree;
run;
```

The PLOTS(UNPACK)=SCREE option produces the scree plot by itself—"unpacked" from its usual location as part of a two-panel display with the variance-explained plot.

The trace output for the scree plot is as follows:

```
Name:        ScreePlot
Label:       Scree Plot
Template:    Stat.Factor.Graphics.ScreePlot1
Path:        Factor.InitialSolution.ScreeAndVarExp.ScreePlot
```

The following statements display the graph template for the scree plot:

```
proc template;
   source Stat.Factor.Graphics.ScreePlot1;
run;
```

The template source statements are as follows:

```
define statgraph Stat.Factor.Graphics.ScreePlot1;
   notes "Scree Plot for Extracted Eigenvalues";
   BeginGraph / designwidth=DefaultDesignHeight;
      Entrytitle "Scree Plot" / border=false;
      layout overlay / yaxisopts=(label="Eigenvalue" gridDisplay=auto_on)
         xaxisopts=(label="Factor" linearopts=(integer=true));
         seriesplot y=EIGENVALUE x=NUMBER / display=ALL;
      endlayout;
   EndGraph;
end;
```

This template has the following components:

- The graph template definition begins with a DEFINE statement of the form: **define statgraph** *template-name*;. An END statement ends the template definition.
- The NOTES statement provides a description or label for the template.
- A block that begins with the BEGINGRAPH statement and ends with the ENDGRAPH statement. Here, the BEGINGRAPH statement has an option that specifies that the outer box which contains the graph should be a square whose width is equal to the default graph height.
- The EntryTitle statement provides the graph title, in this case “Scree Plot”. The BORDER=FALSE option specifies that the title is displayed without a border. In fact, this is the default behavior, so the option is unnecessary. However, it is not unusual to see default specifications in the templates.
- A layout (in this case a LAYOUT OVERLAY statement) is at the heart of the graph template. It ends with an ENDLAYOUT statement, and it is often specified with options. One or more LAYOUT and ENDLAYOUT statement pairs are required. The OVERLAY layout is the most common layout in SAS/STAT templates. Other common layout types are discussed later. This layout provides the label “Eigenvalue” for the vertical or Y axis, provides the label “Factor” for the horizontal or X axis, specifies that grid lines should be produced for the Y axis when the output style favors grids, and specifies that the X axis ticks must be integers. The LINEAROPTS= option is used for options specific to standard axes that depict a linear scaling (as opposed to LOGOPTS=, which is used for log-scale axes).
- One or more statements inside the layout provide the details about what to graph. In this case, the graph is a series plot, produced with the SERIESPLOT statement, which provides a piecewise linear (“connect the dots”) plot. The Y axis column in the ODS data object is named EigenValue, and the X axis column in the ODS data object is named Number (the factor number). The standard series plot display is a series of lines, but the DISPLAY=ALL specification additionally displays the markers (in this case circles) for the data values.

Notice that the title and the axis labels are all specified directly as literal character strings in this template. You can change any of them and submit the results to SAS. From then on, until you change or delete your custom template in Work.Templat or until you end your SAS session, you will see your customization whenever you run PROC FACTOR.

The following example adds a PROC TEMPLATE statement and a RUN statement, changes the title and axis labels, specifies explicit tick values, and removes the grid and the unnecessary BORDER= option:

```
proc template;
   define statgraph Stat.Factor.Graphics.ScreePlot1;
      notes "Scree Plot for Extracted Eigenvalues";
      BeginGraph / designwidth=DefaultDesignHeight;
         Entrytitle "Eigenvalue ((*ESC*){Unicode Lambda}) Plot";
         layout overlay / yaxisopts=(label="(*ESC*){Unicode Lambda}")
            xaxisopts=(label="Factor Number"
                       linearopts=(tickvaluelist=(1 2 3 4 5 6 7 8 9 10)));
            seriesplot y=EIGENVALUE x=NUMBER / display=ALL;
         endlayout;
      EndGraph;
   end;
run;
```

Figure 4 Default Box Plots

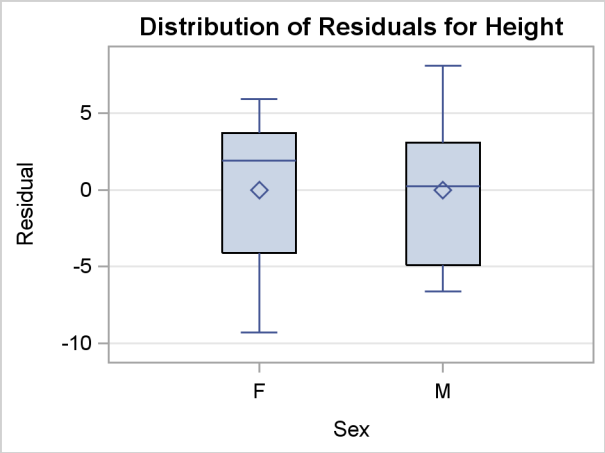
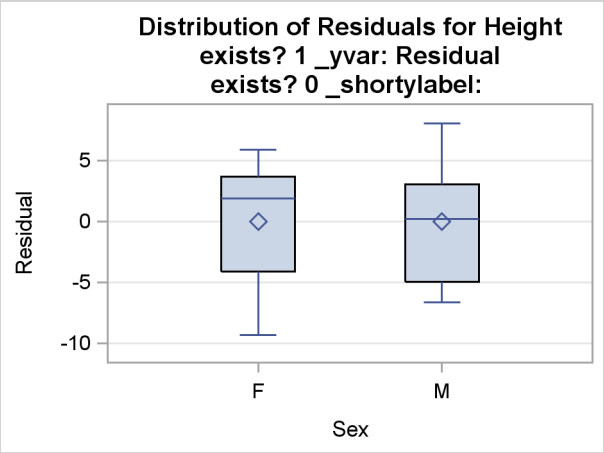


Figure 5 Examining Dynamic Variables



Both the title and the Y axis now contain the Greek letter λ, which is specified as an escape sequence followed by a Unicode specification. The tick value list is specified in full because the GTL does not accept standard SAS shorthand lists. The only output from this step is the following log note:

```
NOTE: STATGRAPH 'Stat.Factor.Graphics.ScreePlot1' has been saved to: WORK.TEMPLAT
```

The following step uses the new template to create a scree plot and produces Figure 3:

```
proc factor data=sashelp.cars plots(unpack)=scree;
run;
```

The following step restores the default template:

```
proc template;
   delete Stat.Factor.Graphics.ScreePlot1;
run;
```

The only output from this step is the following log note:

```
NOTE: 'Stat.Factor.Graphics.ScreePlot1' has been deleted from: WORK.TEMPLAT
```

Changing Titles and Axis Labels Set by Dynamic Variables

In the previous example, the title and both axis labels are specified directly as literal strings in the template. In this example, the procedure provides the title and axis labels at run time. The following step uses the GLIMMIX procedure to create the box plot in Figure 4:

```
ods graphics on;
proc glimmix data=sashelp.class plots=boxplot;
    class sex;
    model height = sex;
run;
```

The trace output for the box plot is as follows:

```
Name:       BoxPlot
Label:      Residuals by Sex
Template:   Stat.Glimmix.Graphics.BoxPlot
Path:       Glimmix.Boxplots.BoxPlot
```

The following statements display the graph template for the box plot:

```
proc template;
   source Stat.Glimmix.Graphics.BoxPlot;
run;
```

The template source statements are as follows:

```
define statgraph Stat.Glimmix.Graphics.BoxPlot;
   dynamic _TITLE _YVAR _SHORTYLABEL;
   BeginGraph;
      entrytitle _TITLE;
      layout overlay / yaxisopts=(gridDisplay=auto_on shortlabel=_SHORTYLABEL)
         xaxisopts=(discreteopts=(tickvaluefitpolicy=rotatethin));
         boxplot y=_YVAR x=LEVEL / labelfar=on datalabel=OUTLABEL
            primary=true freq=FREQ;
      endlayout;
   EndGraph;
end;
```

The procedure uses dynamic variables to provide text strings and option values to the template. In this case, the dynamic variables are a title, a variable name for the Y axis, and a short variable label for the Y axis. Note that dynamic variables cannot provide any arbitrary syntax. For example, they can provide title text and values of options, but not option names, statement names, layout names, and so on.

Axes can have labels and optionally short labels. The label is displayed if there is sufficient space. Otherwise, the short label is used instead. Axis labels (and short labels) can be specified in the template with a literal string, in the template through a dynamic variable, or implicitly. The axis label comes from the first source that provides a value: the LABEL= option in the template (or the SHORTLABEL= option), the data object column label, or the data object column name.

As a SAS user, you cannot peek into the SAS procedure code to see how the dynamic variables, column names, and column labels are set. However, you can do a bit of detective work to learn about these things. The following steps illustrate one approach:

```
proc template;
   define statgraph Stat.Glimmix.Graphics.BoxPlot;
      dynamic _TITLE _YVAR _SHORTYLABEL;
      BeginGraph;
         entrytitle _TITLE;
         entrytitle "exists? " eval(exists(_yvar)) " _yvar: " _yvar;
         entrytitle "exists? " eval(exists(_shortylabel))
                    " _shortylabel: " _shortylabel;
         layout overlay / yaxisopts=(gridDisplay=auto_on shortlabel=_SHORTYLABEL)
            xaxisopts=(discreteopts=(tickvaluefitpolicy=rotatethin));
            boxplot y=_YVAR x=LEVEL / labelfar=on datalabel=OUTLABEL
               primary=true freq=FREQ;
         endlayout;
      EndGraph;
   end;

proc glimmix data=sashelp.class plots=boxplot;
    class sex;
    ods output boxplot=bp;
    model height = sex;
run;

proc contents p;
   ods select position;
run;
```

The graph is displayed in Figure 5, and the data object contents are displayed in Figure 6. The first title is unmodified and simply displays the value of the _Title dynamic variable. Following that, this template is temporarily modified by adding two new EntryTitle statements to report on both the existence and the value of two of the dynamic variables. The expression **eval(exists(***dynamic-variable***))** resolves to 1 (for true) when the dynamic variable is set by the procedure and 0 (for false) when it is not set. It is not unusual for a procedure to conditionally set dynamic variables. A specification of *option=dynamic* is ignored when the dynamic variable does not exist. After the existence information is displayed, the value (if any) is displayed.

Figure 6 Contents of a Data Object

The CONTENTS Procedure

Variables in Creation Order

#	Variable	Type	Len	Format	Label
1	BOX__YVAR_X_LEVEL_DATALABEL_O__Y	Num	8		Residual
2	BOX__YVAR_X_LEVEL_DATALABEL_O_ST	Char	10		
3	BOX__YVAR_X_LEVEL_DATALABEL_O__X	Char	1		Sex
4	BOX__YVAR_X_LEVEL_DATALABEL_O_DL	Num	8	BEST8.	Index
5	Residual	Num	8		
6	Level	Char	1		Sex
7	OutLabel	Num	8	BEST8.	Index

Figure 5 shows that the Y axis column is Residual and the short Y axis label is undefined. The PROC CONTENTS information confirms that the data object has a column called Residual for the Y axis and a column called Level with a label of "Sex" for the X axis. The Y axis column name and the X axis column label become the axis labels. The contents information also displays other columns in the data object.[1]

You have a number of options for modifying templates beyond simply adding or replacing text. For example, you can use the dynamic variables that are provided in creative ways. For example, you could use the title as a label for the Y axis as follows: `yaxisopts=(label=_title . . .)`. This next example will instead replace the X axis label and add a footnote (horizontally aligned on the left using a font that is appropriate for footnotes or second title lines), both through macro variables, as follows:

```
proc template;
   define statgraph Stat.Glimmix.Graphics.BoxPlot;
      dynamic _TITLE _YVAR _SHORTYLABEL;
      mvar datetag xlabel;
      BeginGraph;
         entrytitle _TITLE;
         entryfootnote halign=left textattrs=graphvaluetext datetag;
         layout overlay / yaxisopts=(gridDisplay=auto_on shortlabel=_SHORTYLABEL)
            xaxisopts=(label=xlabel
                       discreteopts=(tickvaluefitpolicy=rotatethin));
            boxplot y=_YVAR x=LEVEL / labelfar=on datalabel=OUTLABEL
               primary=true freq=FREQ;
         endlayout;
      EndGraph;
   end;
run;

%let DateTag = Acme 01Apr2008;
%let xlabel  = Gender;

proc glimmix data=sashelp.class plots=boxplot;
    class sex;
    ods output boxplot=bp;
    model height = sex;
run;
```

The new graph is displayed in Figure 7.

In this example, there is a new footnote, which comes from the value of the macro variable DateTag. DateTag is named in the MVAR (macro variable) statement. When the MVAR statement is used, the template is compiled and the value of the macro variable is substituted when the template is used by the procedure. This approach lets you modify and compile the template once and then use it repeatedly with different values of the macro variable without ever having to recompile the template. You usually use this approach when you make persistent changes in Sasuser.Templat or some other permanent item store. An alternate approach is to use the following statement without using an MVAR statement:

```
entryfootnote "&datetag";
```

[1] Data objects come in many varied forms. You should not expect them to be pretty or well organized for display or subsequent processing. Although you can process them in any way you choose, they are designed for input to one or more templates and very little else. On some occasions, extra columns or extra dynamic variables might be created but not used. These represent cases where the procedure writer recognized possibilities for future processing and tried to facilitate them. They might be helpful when they occur, but most data objects or templates do not have such information. This data object has a number of manufactured and verbose names. You often see names like these when the values that are plotted are statistics of some sort or are computed by ODS Graphics.

Figure 7 Modified Box Plots

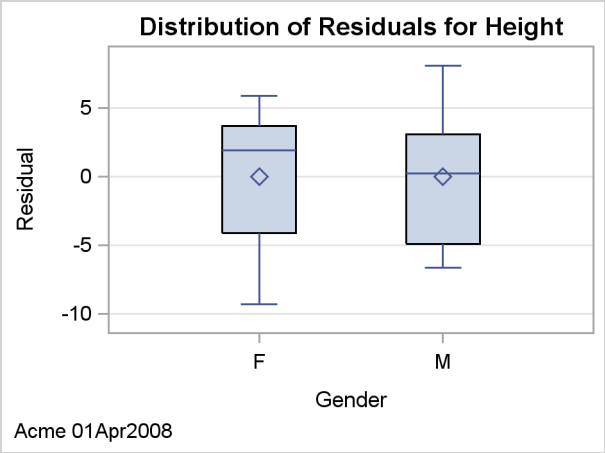
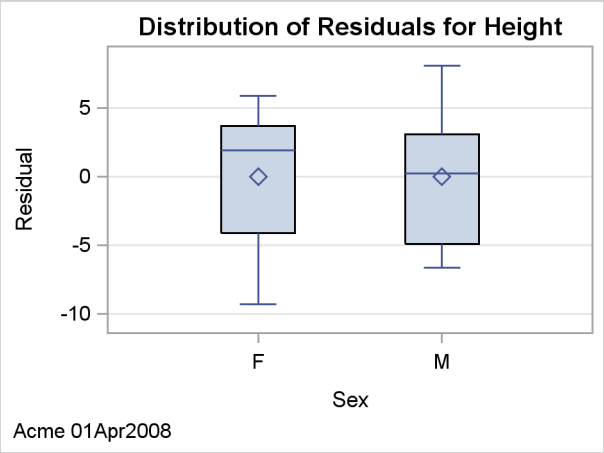


Figure 8 Footnote Added with a Macro



In this approach, the template is compiled and the value of the macro variable is substituted at compile time. The value cannot change in this approach unless you recompile the template. In this case, the approach does not matter because the template is compiled and immediately used.

The X axis label is set to “Gender” using the macro variable xlabel. The X axis change is ad hoc, so changes such as this are usually made temporarily.

The following step restores the default template:

```
proc template;
   delete Stat.Glimmix.Graphics.BoxPlot;
run;
```

If your only goal is to add or change a footnote or title, there is an easier, alternative mechanism. SAS provides a new autocall macro, **ModTmplt**, that you can use for this purpose. This macro is used in the following example:

```
title;
footnote 'halign=left textattrs=graphvaluetext "Acme 01Apr2008"';
%modtmplt(template=Stat.Glimmix.Graphics.BoxPlot, steps=t,
          options=titles noquotes)
footnote;

proc glimmix data=sashelp.class plots=boxplot;
    class sex;
    ods output boxplot=bp;
    model height = sex;
run;

%modtmplt(template=Stat.Glimmix.Graphics.BoxPlot, steps=d)
```

The TITLE statement clears the default title of “The SAS System”. The FOOTNOTE statement provides the footnote along with options to place the footnote on the left using the font that is used for values. The **ModTmplt** macro modifies the box plot template. Only one macro step is run: the template modification step (STEPS=T). OPTIONS=TITLES adds SAS system titles and footnotes (those specified in TITLE and FOOTNOTE statements) to the existing graph titles and footnotes. OPTIONS=NOQUOTES moves the footnote from the FOOTNOTE statement to the EntryFootNote statement but without the outer quotes. You must specify this option if you want to specify EntryTitle or EntryFootNote options in your TITLE or FOOTNOTE statement. See the option “NOQUOTES” on page 33 for more information about this option. The next FOOTNOTE statement clears the footnote so that it affects only the box plot template and does not otherwise affect the analysis. PROC GLIMMIX makes the plot. The final call to the macro deletes the modified template (STEPS=D). The results are displayed in Figure 8. For further discussion of the **ModTmplt** macro, see the section “ModTmplt Macro” on page 31.

Figure 9 Transposed Scatter Plot with Labels

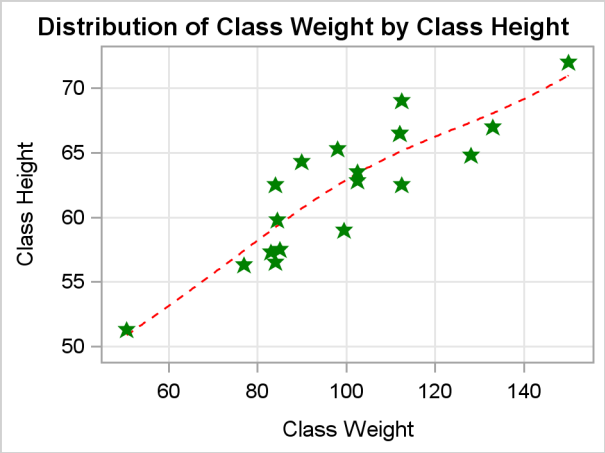
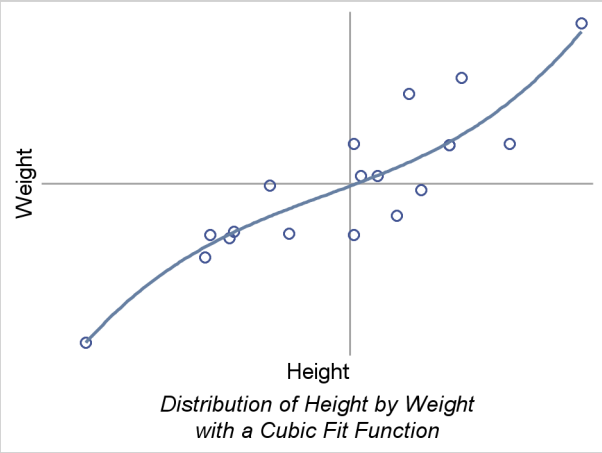


Figure 10 Scatterplot with Numerous Modifications



Modifying Colors, Lines, Markers, Axes, and Reference Lines

The following template source appeared in a previous section:

```
define statgraph Stat.KDE.Graphics.ScatterPlot;
   dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
   BeginGraph;
      EntryTitle "Distribution of " _VAR1NAME " by " _VAR2NAME;
      layout Overlay / xaxisopts=(offsetmin=0.05 offsetmax=0.05)
         yaxisopts=(offsetmin=0.05 offsetmax=0.05);
         ScatterPlot x=X y=Y / markerattrs=GRAPHDATADEFAULT;
      EndLayout;
   EndGraph;
end;
```

Here, the entry title is a mix of constant text and dynamic variables that provide variable names. The procedure writer has provided you with additional dynamic variables that provide the variable labels. This template also has offset options specified. These options are frequently used in scatter plots and other graphs. They add a small amount of white space to the left or bottom (OFFSETMIN=) and to the right or top (OFFSETMAX=) of the specified axis. The SCATTERPLOT statement has a MARKERATTRS=GRAPHDATADEFAULT specification, which references a style element. In addition, this style element can be specified with lines to control line color and thickness.

The following statements switch the Y and X axis variables, use variable labels instead of variable names in the title, change the marker characteristics, add a nonlinear penalized B-spline fit function, and add grids:

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
      BeginGraph;
         EntryTitle "Distribution of " _VAR2LABEL " by " _VAR1LABEL;
         layout Overlay /
            xaxisopts=(offsetmin=0.05 offsetmax=0.05 griddisplay=on)
            yaxisopts=(offsetmin=0.05 offsetmax=0.05 griddisplay=on);
            pbsplineplot x=y y=x / lineattrs=(color=red pattern=2 thickness=1);
            ScatterPlot  x=y y=x / markerattrs=(color=green size=5px
                                   symbol=starfilled weight=bold);
         EndLayout;
      EndGraph;
   end;
run;

proc kde data=sashelp.class;
   label height = 'Class Height' weight = 'Class Weight';
   bivar height weight / plots=scatter;
run;
```

The results are displayed in Figure 9.

Note that the addition of the variable labels in the PROC KDE step also changes the axis labels because no axis labels are explicitly specified in the template. The PBSPLINEPLOT statement includes the LINEATTRS= option which specifies the color (red), pattern (2, dashed), and thickness (1 pixel) of the fit function. The SCATTERPLOT statement includes the MARKERATTRS= option which specifies the color (green), size (5 pixels), symbol (a filled star), and weight (bold) of the marker or symbol.

The starting point for the next step is the original PROC KDE scatter plot template, rather than the modified template. The goal in this example is to produce the scatter plot displayed in Figure 10 with axes passing through the center of the data. The following steps make a highly modified scatter plot:

```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
      BeginGraph;
         EntryFootNote "Distribution of " _VAR1NAME " by " _VAR2NAME;
         EntryFootNote "with a Cubic Fit Function";
         layout Overlay / walldisplay=none
            xaxisopts=(display=(label))
            yaxisopts=(display=(label));
            referenceline y=eval(mean(y));
            referenceline x=eval(mean(x));
            ScatterPlot x=X y=Y / markerattrs=GRAPHDATADEFAULT;
            regressionplot x=x y=y / degree=3;
         EndLayout;
      EndGraph;
   end;
run;

proc kde data=sashelp.class;
   bivar height weight / plots=scatter;
run;
```

The template is modified by adding an MVAR statement to use the mean height and mean weight. Specifically, a reference line is displayed at the mean for each axis. The title is changed to a footnote, and a second footnote is added. The LAYOUT OVERLAY statement now has a WALLDISPLAY=NONE option to suppress the axes, and only the labels are displayed on each axis. A cubic-polynomial fit function is also added. The results are displayed in Figure 10.

The LAYOUT OVERLAY block has four statements in it. The statements are executed in the order in which they appear in the LAYOUT OVERLAY. Reference lines are displayed first. Therefore any point or function that coincides with the reference line is displayed on top of the reference line. Similarly, the fit function is displayed rather than the points in the places where they coincide. You can vary the order of the statements if you prefer some other effect. The plot has no axes, no ticks, no tick labels, and no wall. (The wall is the area inside the plot axes, which can be a different color from the background color outside of the axes.) Instead, the plot simply has reference lines at the means and axis labels. Many variations can be tried. In the interest of space, several variations are discussed but their results are not shown.

The following statement displays a left axis and a bottom axis (but no top axis or right axis), and the color outside the axes matches the color inside:

```
layout Overlay / walldisplay=none;
```

The following statement displays all four axes, and the color outside the axes matches the color inside:

```
layout Overlay / walldisplay=(outline);
```

The following statement suppresses all axis information (the axes, the ticks, the tick labels, the axis labels, and the wall):

```
layout Overlay / walldisplay=none
   xaxisopts=(display=none) yaxisopts=(display=none);
```

Repositioning Legends

This section creates a plot with a legend. The following step runs the GLM procedure and produces a residual histogram:

```
proc glm plots=diagnostics(unpack) data=sashelp.class;
   model weight = height;
   ods output residualhistogram=hr;
run;

proc contents p;
   ods select position;
run;
```

This type of graph (shown in Figure 12) is used in many procedures.

The trace output for the residual histogram is as follows:

```
Name:       ResidualHistogram
Label:      Residual Histogram
Template:   Stat.GLM.Graphics.ResidualHistogram
Path:       GLM.ANOVA.Weight.DiagnosticPlots.ResidualHistogram
```

The following statements display the graph template for the residual histogram:

```
proc template;
   source Stat.GLM.Graphics.ResidualHistogram;
run;
```

The template source statements are as follows:

```
define statgraph Stat.GLM.Graphics.ResidualHistogram;
   notes "Residual Histogram with Overlayed Normal and Kernel";
   dynamic Residual _DEPNAME;
   BeginGraph;
      entrytitle "Distribution of Residuals" " for " _DEPNAME;
      layout overlay / xaxisopts=(label="Residual")
         yaxisopts=(label="Percent");
         histogram RESIDUAL / primary=true;
         densityplot RESIDUAL / name="Normal" legendlabel="Normal"
            lineattrs=GRAPHFIT;
         densityplot RESIDUAL / kernel () name="Kernel" legendlabel="Kernel"
            lineattrs=GRAPHFIT2;
         discretelegend "Normal" "Kernel" / across=1 location=inside
            autoalign=(topright topleft top);
      endlayout;
   EndGraph;
end;
```

This graph template consists of a histogram of residuals. On top of the histogram is a normal density plot, and on top of both is a kernel density plot. Additionally, a legend is positioned inside the graph. The preferred position is in the top right, but ODS Graphics automatically repositions the legend in the top left or top center if there are conflicts between the legend and the histogram or functions in the top right.

The EntryTitle statement specifies the title, which consists of literal text and a dynamic variable that contains the dependent variable name. The LAYOUT OVERLAY statement specifies the labels for both axes. Since the labels never vary in this template, they are specified directly in the template. The HISTOGRAM statement creates a histogram from the data object column named Residual. It is the primary statement in the overlay. The data columns from the primary statement determine the default axis types and default axis labels. By default, the first graph statement is the primary statement. Hence, in this case the PRIMARY= option is not needed. However, in many other cases it is needed. You must specify PRIMARY=TRUE when you want a statement other than the first to control the axes. The color, width, and line style for the normal density plot comes from the GRAPHFIT style element (blue and solid in this style), and for the kernel density plot comes from the GRAPHFIT2 style element (red and dashed in this style). All graphs are based on the same data object column, Residual, and the kernel density plot uses default options for finding the kernel density.

The contents of the data object are displayed in Figure 11. From the original input variable Residual, six other variables are created by the HISTOGRAM and the two DENSITYPLOT statements. The X and Y axis variables for the density plot are BIN_RESIDUAL___X and BIN_RESIDUAL___Y; for the normal density plot they are NORMAL_RESIDUAL___X and

NORMAL_RESIDUAL___Y; and for the kernel density plot they are KERNEL_RESIDUAL___X and KERNEL_RESIDUAL___Y. If you display the output data set created from this data object, you will see that the variables do not have the same number of nonmissing values. Some, such as the histogram values, have many fewer than the others. In this case, the computed density values have many more values than the raw residuals. Data objects are often constructed from pieces of very different sizes.

Figure 11 Contents of the Data Object

The CONTENTS Procedure

Variables in Creation Order

#	Variable	Type	Len	Label
1	Dependent	Char	8	
2	BIN_RESIDUAL___X	Num	8	Residual
3	BIN_RESIDUAL___Y	Num	8	Percent
4	NORMAL_RESIDUAL___X	Num	8	Residual
5	NORMAL_RESIDUAL___Y	Num	8	Percent
6	KERNEL_RESIDUAL___X	Num	8	Residual
7	KERNEL_RESIDUAL___Y	Num	8	Percent
8	Residual	Num	8	

All of the remaining options concern the legend. The discrete legend is produced by the DISCRETELEGEND statement. In contrast, a continuous legend is used to produce a color “thermometer” legend when point or surface colors vary continuously as a function of a third variable. The legend is constructed from the statements named “Normal” and “Kernel” by the NAME= option in each of the two DENSITYPLOT statements. The labels for these two legend components come from the LEGENDLABEL= options. The legend has only one component in each row due to the ACROSS=1 option.

The following steps modify the graph by moving the legend outside the graph and by removing the ACROSS= option, which for this graph produces a legend with one row and two entries:

```
proc template;
   define statgraph Stat.GLM.Graphics.ResidualHistogram;
      notes "Residual Histogram with Overlayed Normal and Kernel";
      dynamic Residual _DEPNAME;
      BeginGraph;
         entrytitle "Distribution of Residuals" " for " _DEPNAME;
         layout overlay / xaxisopts=(label="Residual")
            yaxisopts=(label="Percent");
            histogram RESIDUAL / primary=true;
            densityplot RESIDUAL / name="Normal"
               legendlabel="Normal" lineattrs=GRAPHFIT;
            densityplot RESIDUAL / kernel ()
               name="Kernel" legendlabel="Kernel" lineattrs=GRAPHFIT2;
            discretelegend "Normal" "Kernel";
         endlayout;
      EndGraph;
   end;
run;

proc glm plots=diagnostics(unpack) data=sashelp.class;
   model weight = height;
run;
```

The results are displayed in Figure 13.

Understanding the Lattice Layout and Panels

The examples so far have been simple in that they produce a graph with one panel. The templates so far consist of a single LAYOUT OVERLAY with one or more plotting statements inside. However, many graphs consist of two or more panels within a single display. For example, the scree plot displayed in Figure 2 is, by default, part of a two-panel display. It is produced when you run PROC FACTOR without the UNPACK option as follows:

```
proc factor data=sashelp.cars plots=scree;
run;
```

Figure 12 Default Residual Histogram

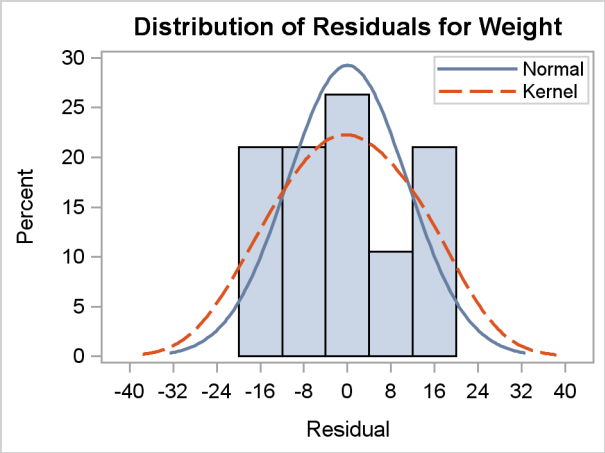
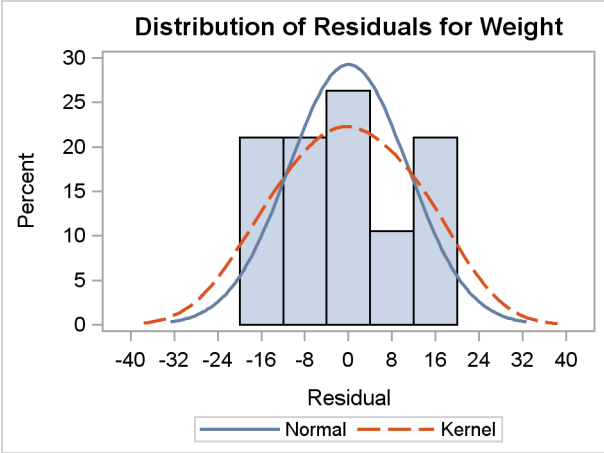


Figure 13 Residual Histogram with Repositioned Legend



The graph is displayed in Figure 14.

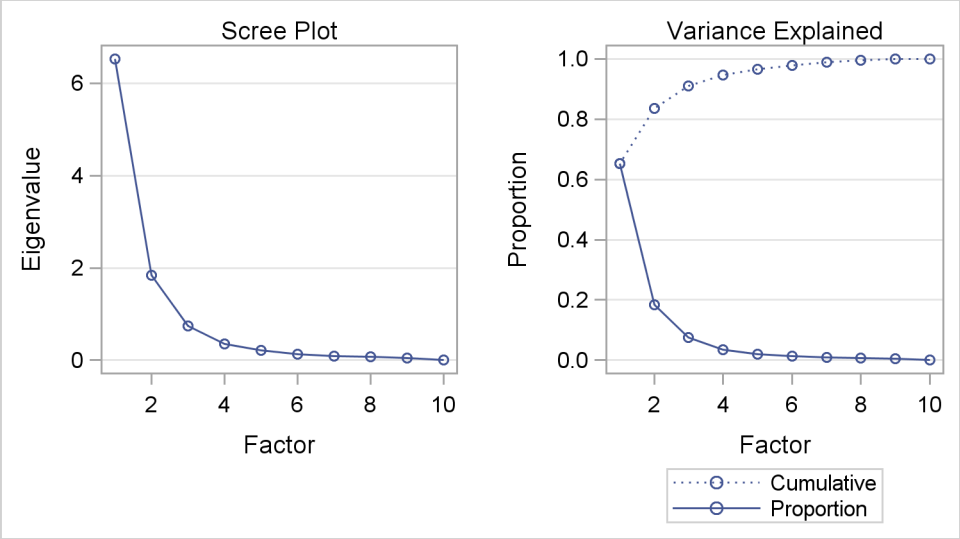
The trace output (not shown) shows that the template is called `Stat.Factor.Graphics.ScreePlot2`. A slight simplification of the template source statements is as follows:

```
define statgraph Stat.Factor.Graphics.ScreePlot2;
   notes "Scree and Proportion Variance Explained Plots";
   BeginGraph / DesignHeight=360px;
      layout lattice / rows=1 columns=2 columngutter=30;
         layout overlay / yaxisopts=(label="Eigenvalue" gridDisplay=auto_on)
            xaxisopts=(label="Factor" linearopts=(integer=true));
            entry "Scree Plot" / textattrs=GRAPHLABELTEXT location=outside;
            seriesplot y=EIGENVALUE x=NUMBER / display=ALL;
         endlayout;
         layout overlay / yaxisopts=(label="Proportion" gridDisplay=auto_on)
            xaxisopts=(label="Factor" linearopts=(integer=true));
            entry "Variance Explained" / textattrs=GRAPHLABELTEXT
                                         location=outside;
            seriesplot y=PROPORTION x=NUMBER / display=ALL legendlabel=
               "Proportion" name="Proportion";
            seriesplot y=CUMULATIVE x=NUMBER / lineattrs=GRAPHDATADEFAULT (
               pattern=dot) display=ALL LegendLabel="Cumulative" name=
               "Cumulative" primary=true;
            DiscreteLegend "Cumulative" "Proportion" / across=1 border=1;
         endlayout;
      endlayout;
   EndGraph;
end;
```

The template begins with a BEGINGRAPH statement. Most templates do not contain a specific numerical size for the overall graph area. This template does, so that the two resulting plots are approximately square. At the default size, the plots are tall and thin. Note that size is a “design height” rather than a hardcoded size. The graph is designed at a height of 360 pixels, but it can be stretched or shrunk to other sizes while preserving the aspect ratio.

The next layer is a LAYOUT LATTICE that creates a display with one row and two columns. Row and column gutters are frequently specified in lattice layouts. The COLUMNGUTTER=30 specification ensures that there are 30 pixels between the two columns of graphs. (By default, the plots are closer than that.) Inside of the lattice layout are two LAYOUT OVERLAY blocks, one for each graph. Each individual LAYOUT OVERLAY block is designed in much the same way it would be designed if it were in a one-panel display. However, in practice it is not unusual for an “unpacked graph” (a graph produced in a single panel) to be different from the same graph packed into a display with other graphs.

Figure 14 Graph with Two Panels



In some cases, the unpacked plot might have additional graph elements due to the increased room in the unpacked plot. One difference in the paneled plot is the title. In this case, the goal is to have two titles, one for each plot with no overall title. Hence, there is no EntryTitle statement. Instead, there are two ENTRY statements, which place the title outside (and above) each plot by using the GRAPHLABELTEXT style element. By default, without this style specification, the text would be smaller and would not look like other titles. The LOCATION=OUTSIDE option is an example of one of the undocumented options that appear in some templates.

The last SERIESPLOT statement specifies the option: **`lineattrs=GRAPHDATADEFAULT(pattern=dot)`**. The properties of the line produced by this statement are controlled by the GRAPHDATADEFAULT style element. However, one aspect of the style (namely the line pattern) is overridden and a dotted line is used instead. PATTERN= is one of the options in the LINEATTRS= option, rather than the name of a style element (which is **`MarkerSymbol`**). Since GRAPHDATADEFAULT is the default style for the first series plot in the second overlay layout, the specification in the second series plot ensures that the two series have identical styles (except for one aspect) so that they can be distinguished in the legend. Most SAS/STAT templates do not hardcode graph elements such as this (the dotted line); usually they strictly use style elements. However, on occasion you will see explicit specifications. For example, the LOESS and TRANSREG procedures use the specification **`MarkerAttrs=GraphData1(symbol=star size=15)`** to mark the minimum of a function that is being optimized. This specification produces a large star. Numerical sizes like SIZE=15 are design sizes; they can be stretched or shrunk to other sizes while preserving the aspect ratio.

You can do many things to modify this template. You can change the titles, axis labels, colors, markers, and so on. All of these are illustrated in other parts of this paper. You can switch the order of the layouts and put the variance-explained plot first; you can provide an overall title; and so on. However, rather than perform familiar or obvious changes, the rest of the paper concentrates on understanding other aspects of the GTL and the complicated templates that you might encounter. For further discussion of complicated templates, see the section “Appendix I: Template Complexity” on page 38.

Understanding Conditional Template Logic

This example assumes that you know how to run the procedure and find the name of the template. It concentrates on explaining the layout of a template with conditional logic and nested IF statements. You might need to understand conditional template logic when you determine which part of a template to modify. The survival estimate plot from the LIFETEST procedure has a long and complicated template, **`Stat.Lifetest.Graphics.ProductLimitSurvival`**, which includes nested IF statements. A very small part of this template is as follows:

```
define statgraph Stat.Lifetest.Graphics.ProductLimitSurvival;
   dynamic . . .;
   BeginGraph;
      if (NSTRATA=1)
         if (EXISTS(STRATUMID))
         entrytitle "Product-Limit Survival Estimate" " for " STRATUMID;
      else
         entrytitle "Product-Limit Survival Estimate";
      endif;
      if (PLOTATRISK)
         entrytitle "with Number of Subjects at Risk" / textattrs=
         GRAPHVALUETEXT;
      endif;
      layout overlay . . .;
          . . .
      endlayout;
      else
         entrytitle "Product-Limit Survival Estimates";
      if (EXISTS(SECONDTITLE))
         entrytitle SECONDTITLE / textattrs=GRAPHVALUETEXT;
      endif;
      layout overlay . . .;
         . . .
      endlayout;
      endif;
   EndGraph;
end;
```

This layout is confusing even when displayed like this with most details removed. In its original form at 145 lines, it is even more confusing. To understand the layout of this template, you must carefully evaluate the IF, ELSE, ENDIF, structure. The following step shows the structure manually re-indented, with additional details removed and additional white space added:

```
define statgraph Stat.Lifetest.Graphics.ProductLimitSurvival;
   dynamic . . .;
   BeginGraph;
      if (NSTRATA=1)

         if (EXISTS(STRATUMID)) entrytitle . . .;
         else entrytitle . . .;
         endif;

         if (PLOTATRISK) entrytitle . . .;
         endif;

         layout overlay ...;
             . . .
         endlayout;

      else

         entrytitle . . .;

         if (EXISTS(SECONDTITLE)) entrytitle . . .;
         endif;

         layout overlay . . .;
            . . .
         endlayout;

      endif;
   EndGraph;
end;
```

The IF and ELSE statements do not perform as do the similarly named statements in the DATA step. There are no DO and END statements. When the first IF condition is true, the statements under the first IF statement are executed until control reaches the ELSE statement at the same indentation level. The statements in the ELSE block include everything

below the ELSE and through the ENDIF at the same level. Inside the first IF block, a title is provided if a condition is true. Otherwise a different title is provided, and that block ends with the first ENDIF statement.

If there is one stratum (NSTRATA=1), then the graph consists of one of two conditional titles followed by a conditional second title followed by a graph defined in a layout. With more than one stratum, the graph consists of an unconditional title, a conditional second title, and a graph defined in a different layout. Sometimes the easiest way to understand a template structure is to do precisely what is done here: copy the template and remove details until you are left with an outline of the overall structure. Then use that knowledge to go back and evaluate and modify the full template.

The following provides a similar manual re-edit and pruning, this time concentrating on the titles:

```
define statgraph Stat.Lifetest.Graphics.ProductLimitSurvival;
   dynamic . . .;
   BeginGraph;
      if (NSTRATA=1)
         if (EXISTS(STRATUMID))
            entrytitle "Product-Limit Survival Estimate" " for " STRATUMID;
         else
            entrytitle "Product-Limit Survival Estimate";
         endif;
         if (PLOTATRISK)
            entrytitle "with Number of Subjects at Risk" / textattrs=GRAPHVALUETEXT;
         endif;
       . . .
      else
         entrytitle "Product-Limit Survival Estimates";
         if (EXISTS(SECONDTITLE))
            entrytitle SECONDTITLE / textattrs=GRAPHVALUETEXT;
         endif;
      . . .
      endif;
   EndGraph;
end;
```

You can see that titles can come from literal strings, dynamic variables, or both. If you are unclear about which title appears in the output, you can temporarily change the titles by adding some text to clearly show which is which. The following statements show how:

```
entrytitle "(1) Product-Limit Survival Estimate" " for " STRATUMID;
entrytitle "(2) Product-Limit Survival Estimate";
entrytitle "(3) with Number of Subjects at Risk" / textattrs=GRAPHVALUETEXT;
entrytitle "(4) Product-Limit Survival Estimates";
entrytitle "(5)" SECONDTITLE / textattrs=GRAPHVALUETEXT;
```

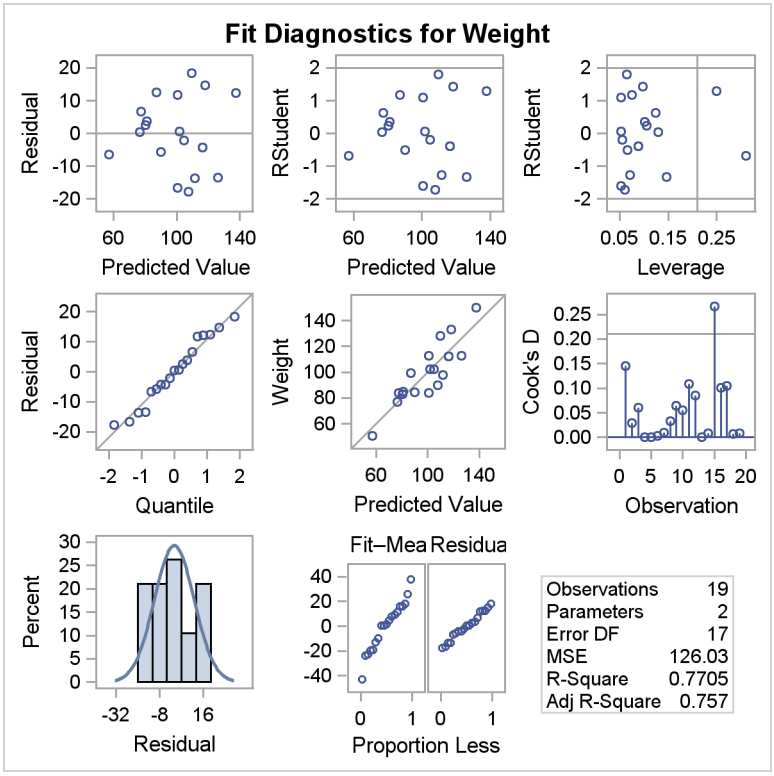
If you submit the full template with titles like these, you can clearly see whether a title is used and where it is used. You can apply the same technique to axis labels, legend labels, and any other text in the template. Then you can remove the identification numbers, modify the text of interest, and submit the modified template. For example, you might wish to change the first, second, and fourth title to “Kaplan-Meier Plot”. Note that it is not always sufficient to find and change the first entry title in a template.

In a previous example, an ENTRY statement specified the style element GRAPHLABELTEXT so that the entry text would look like a title. In this template, an EntryTitle statement specifies the style element GRAPHVALUETEXT so that second titles are subordinate (less bold or smaller according to the style) to the first title lines.

IF, ELSE, and ENDIF statements cannot be used in arbitrary ways. The GTL code that is conditional must be complete. For example, the following statements produce an error:

```
if ( exists(SQUAREPLOT) )                    /* Wrong! */
    layout overlayequated / equatetype=square; /* Wrong! */
 else                                        /* Wrong! */
    layout overlay;                          /* Wrong! */
 endif;                                      /* Wrong! */
    scatterplot x=XVAR y=YVAR;               /* Wrong! */
 endlayout;                                  /* Wrong! */
```

Figure 15 Diagnostics Panel for the REG Procedure



The following statements are correct:

```
if (exists(SQUAREPLOT))
   layout overlayequated / equatetype=square;
      scatterplot x=XVAR y=YVAR;
   endlayout;
else
   layout overlay;
      scatterplot x=XVAR y=YVAR;
   endlayout;
endif;
```

The incorrect example attempts to conditionally execute a complete statement, but only complete layouts (not merely complete layout statements) can be conditionally executed. Also note that IF conditions determine what is rendered in the plot rather than what is computed for the data object. For example, the following step attempts to compute a LOESS fit whether or not the LOESSPLOT dynamic variable is defined:

```
if (exists(LOESSPLOT))
   loessplot y=LOESS x=X;
endif;
```

Since the LOESS fit is computationally expensive, procedure writers use a different approach to conditionally compute results only when needed. If either LOESS or X is a dynamic variable that is not defined to a data object column name, then the computation is not performed.

Working with Templates for Paneled Displays

This example discusses how to understand the overall structure of a paneled display with many graphs and how to isolate individual graphs that you might want to modify. The following step fits a regression model and displays a set of model fit diagnostics:

```
proc reg data=sashelp.class;
   model weight = height;
run; quit;
```

The diagnostics panel is displayed in Figure 15.

The rendered version of the diagnostics panel template, **`Stat.Reg.Graphics.DiagnosticsPanel`**, has 271 lines. A very small portion of it is as follows:

```
define statgraph Stat.Reg.Graphics.DiagnosticsPanel;
   notes "Diagnostics Panel";
   dynamic . . .;
   BeginGraph / designheight=defaultDesignWidth;
      entrytitle . . .;
      layout lattice / columns=3 rowgutter=10 columngutter=10
         shrinkfonts=true rows=3;
         layout overlay . . . scatterplot . . . endlayout;
         layout overlay . . . scatterplot . . . endlayout;
         layout overlay . . . scatterplot . . . endlayout;
         layout overlay . . . scatterplot . . . endlayout;
         layout overlayequated . . . scatterplot . . . endlayout;
         layout overlay . . . needleplot . . . endlayout;
         layout overlay . . . histogram . . . densityplot . . . endlayout;
         layout lattice / columns=2 rows=1 rowdatarange=unionall columngutter=0;
            . . .
            layout overlay . . . scatterplot . . . endlayout;
            layout overlay . . . scatterplot . . . endlayout;
            . . .
         endlayout;
         . . .
         layout overlay;
            layout gridded / columns=_NSTATSCOLS valign=center border=TRUE
               BackgroundColor=GraphWalls:Color Opaque=true;
               . . . entry halign=left "Observations" / valign=top;
               . . . entry halign=right eval (PUT(_NOBS,BEST6.)) / valign=top;
               . . .
            endLayout;
         endif;
      . . .
      endlayout;
   EndGraph;
end;
```

The paneled display is large and square (although greatly reduced from the default size for this paper) and is designed with a height equal to the default width. It has a single overall title for the display. It consists of a 3×3 lattice of nine entries. The first eight panels are graphs, and the last is a table of statistics. The graphs that are defined in the overlay layouts fill in the display in order from left to right and from top to bottom. The first four graphs are ordinary scatter plots; the fifth is an equated scatter plot where both axes are equated to represent the same data range; the sixth is a needle plot; the seventh is an overlay of a histogram and a density plot; the eighth is another lattice, this one consisting of two scatter plots; and the ninth and final panel in the outer lattice is a grid that contains statistic names and their values. The outermost lattice specifies the SHRINKFONTS=TRUE option. This option is commonly specified in outer lattices and specifies that fonts can be scaled down when the graph is reduced in size. Without this option, the text is typically too large in reduced versions of displays such as Figure 15.

Even when the overall template is huge, you can often find and isolate small template components that are easily understood. For example, the first plot, which displays residuals and predicted values, is created from the following layout:

```
layout overlay / xaxisopts=(shortlabel='Predicted');
   referenceline y=0;
   scatterplot y=RESIDUAL x=PREDICTEDVALUE / primary=true datalabel=
      _OUTLEVLABEL rolename=(_tip1=OBSERVATION _id1=ID1 _id2=ID2 _id3=
      ID3 _id4=ID4 _id5=ID5) tip=(y x _tip1 _id1 _id2 _id3 _id4 _id5);
endlayout;
```

The DATALABEL= option provides labels for the markers when the dynamic variable _OutLevLabel exists. The ROLENAME= and TIP= options create tooltips in HTML. Tooltips are text boxes that appear in HTML output when your mouse pointer hovers over a part of the plot. Tips are produced for the Y axis column, the X axis column, and additional columns _tip1 and _id1 through _id5. The columns x and y have predefined roles as axis variables. In contrast, the other tips are

provided for columns that are identified through the ROLENAME= option. You must provide role names for columns that do not have automatic roles (such as the axis columns) and use the role names rather than the column names in the TIP= option. You can modify the tooltips by adding, deleting, or changing columns named in these lists. These options usually appear in templates for graphs that display data or computed values with a one-to-one correspondence with the data (for example, independent variable, dependent variable, predicted values, residuals, leverage, and variables specified in the procedure's ID statement).

Part of the gridded layout that composes the ninth panel is as follows (after some manual indentation adjustments):

```
if (_SHOWNOBS^=0)
   entry halign=left "Observations" / valign=top;
   entry halign=right eval (PUT(_NOBS,BEST6.)) / valign=top;
endif;
if (_SHOWTOTFREQ^=0)
   entry halign=left "Total Frequency" / valign=top;
   entry halign=right eval (PUT(_TOTFREQ,BEST6.)) / valign=top;
endif;
if (_SHOWNPARM^=0)
   entry halign=left "Parameters" / valign=top;
   entry halign=right eval (PUT(_NPARM,BEST6.)) / valign=top;
endif;
```

Do not rely on the indentation provided by PROC TEMPLATE and the SOURCE statement to see the structure of a template. Re-indent the template yourself to make it clearer. Each statistic is added to the display conditional on a dynamic variable. First, a label is displayed on the left followed by a value on the right. In a table such as this, you could change the labels, change the formats, remove statistics, or reorder them.

The layout for the fourth graph, the normal quantile plot of the residuals, which is displayed in the second row and first column of the panel, is as follows:

```
layout overlay / yaxisopts=(label="Residual" shortlabel="Resid")
   xaxisopts=(label="Quantile");
   lineparm slope=eval (STDDEV(RESIDUAL)) y=eval (MEAN(RESIDUAL)) x=0
      / extend=true lineattrs=GRAPHREFERENCE;
   scatterplot y=eval (SORT(DROPMISSING(RESIDUAL))) x=eval (
      PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL))) -0.375)/
      (0.25 + N(RESIDUAL)))) / markerattrs=GRAPHDATADEFAULT
      primary=true rolename=(s=eval (SORT(DROPMISSING(RESIDUAL)))
      nq=eval (PROBIT((NUMERATE(SORT(DROPMISSING(RESIDUAL)))
      -0.375)/(0.25 + N(RESIDUAL))))) tiplabel=(nq="Quantile" s="Residual")
      tip=(nq s);
endlayout;
```

Again, this code has been manually reformatted. The PROC TEMPLATE SOURCE statement struggles with complicated code like this. Besides having indentation problems, it sometimes breaks lines in the middle of names. These problems must be fixed manually before the generated code can be compiled again by PROC TEMPLATE.

This template differs from others shown previously due to the heavy reliance on expression evaluation. The GTL provides a series of functions that can be used to make plots. The LINEPARM statement produces a diagonal reference line whose slope is the standard deviation of the residuals. A line is determined given a slope and a point. The Y= option provides the Y coordinate of a point, which is the mean of the residuals. The X= option provides the X coordinate of that same point, which is 0. When X=0, then Y= provides the intercept. The scatter plot consists of the sorted residuals (ignoring missing values) on the Y axis and normal quantiles on the X axis. These quantities are also provided as tooltips. Functions and expressions must always be wrapped in the EVAL function.

Style Modifications

ODS styles control the colors and general appearance of all graphs and tables. This section provides a series of examples of modifying ODS styles. Styles are composed of style elements that control specific aspects of graphs (for example, the font of a title, the color of a reference line, the style of a regression fit line, and so on). One particular set of style elements allows you to modify how groups of observations are distinguished in a graph. These are the **`GraphData`***n* style elements (**`GraphData1`** through **`GraphData12`**). In most cases, it is easiest to modify these elements through the **`ModStyle`** SAS autocall macro instead of directly modifying a style. The first examples illustrate using this macro. The macro is documented later in this paper and in the macro header. Also see the documentation section "Some Common

Style Elements" (Chapter 21, *SAS/STAT User's Guide*).

Figure 16 STATISTICAL Style

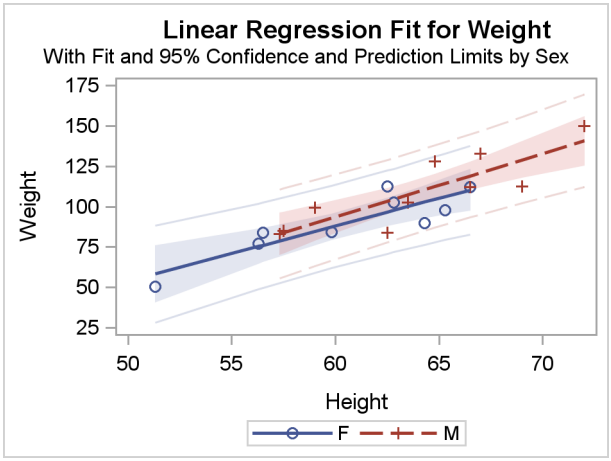
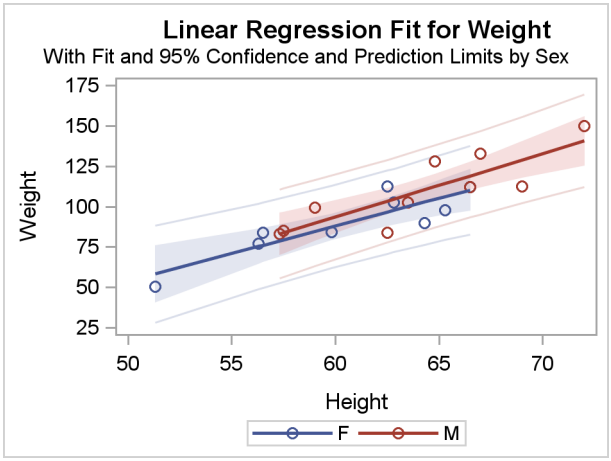


Figure 17 A Color-Based Style



An All-Color Style

Many styles are designed to make color plots in which you can distinguish lines, functions, and groups of observations even when you send the plot to a black-and-white printer. Hence, lines and markers differ not only in color but also in pattern and the symbol. You can use the **ModStyle** autocall macro to create a new style (for example, STATCOLOR) by modifying a parent style and reordering the colors, line patterns, and marker symbols in the **GraphData***n* style elements.

When you only specify a parent style and a new style name, and you use the defaults for all other options, the **ModStyle** macro creates a new style that uses only color to distinguish the groups. Lines and markers in subsequent groups match the lines and markers in the first group. The following example creates a plot with two groups, first with the STATISTICAL style, and then with a color-only style created by the default use of the **ModStyle** macro:

```
proc transreg data=sashelp.class;
   model identity(weight) = class(sex / zero=none) | identity(height);
run;

%modstyle(parent=statistical, name=StatColor)

ods listing style=StatColor;

proc transreg data=sashelp.class;
   model identity(weight) = class(sex / zero=none) | identity(height);
run;
```

The graph using the STATISTICAL style is displayed in Figure 16, and the graph using the new style is displayed in Figure 17. In both plots, the females are represented by blue circles and a blue solid line. In the first plot, the males are represented by red pluses and a red dashed line, whereas in the second plot, the males and female groups differ only by color.

An Ad Hoc Style Modification for Groups

Styles are general; they are not made for specific types of data that have common color associations such as blue with male. The following step modifies the line styles, markers, and colors of the STATISTICAL style and creates a new style where the points for males are displayed in blue:

```
%modstyle(parent=statistical, name=GenderStyle, type=CLM,
          colors=red blue, fillcolors=red blue,
          markers=plus circle, linestyles=solid solid)
```

Figure 18 Gender Style

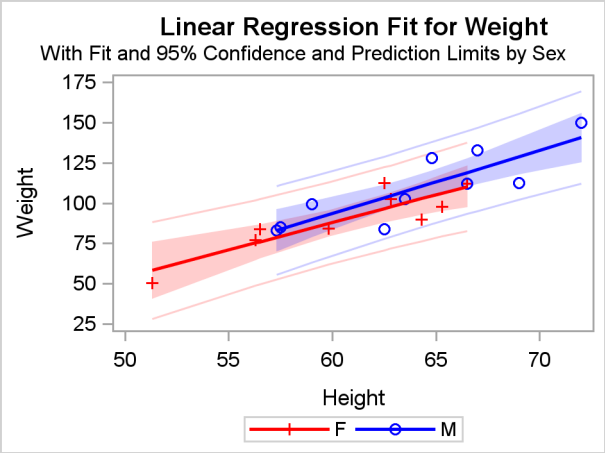
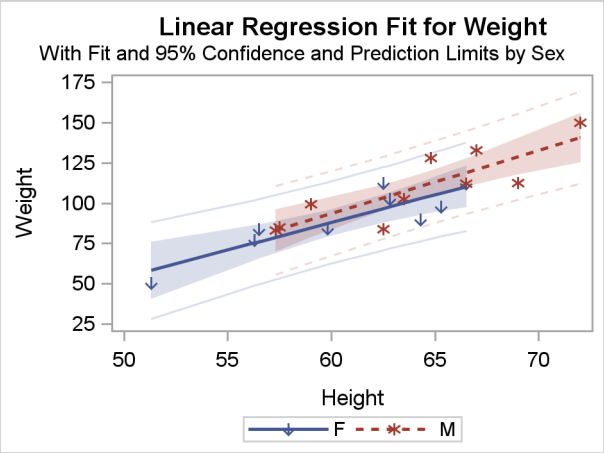


Figure 19 An Ad Hoc Style



The colors for the first group (females, since 'F' comes before 'M') are set to red, and the colors for the second group are set to blue. The COLORS= option controls the line and marker colors, and the FILLCOLORS= option controls the colors for the confidence limits. The actual colors for the prediction and confidence limits are lighter due to the application of transparency. The marker for females is a plus, and the marker for males is a circle. The line style for both groups is a solid line. The TYPE=CLM option specifies that colors (C), lines (L), and markers (M) all vary simultaneously. The following step uses the new style and creates Figure 18:

```
ods listing style=GenderStyle;

proc transreg data=sashelp.class;
   model identity(weight) = class(sex / zero=none) | identity(height);
run;
```

The following step modifies the line styles, markers, and colors of the STATISTICAL style, creates a new style, and uses it to create Figure 19:

```
%modstyle(parent=statistical, name=GenderStyle, type=CLM,
          colors=GraphColors("gcdata1") GraphColors("gcdata2")
                 green cx543005 cx9D3CDB cx7F8E1F cx2597FA cxB26084
                 cxD17800 cx47A82A cxB38EF3 cxF9DA04 magenta,
          fillcolors=colors,
          linestyles=Solid ShortDash MediumDash LongDash MediumDashShortDash
                  DashDashDot DashDotDot Dash LongDashShortDash Dot
                  ThinDot ShortDashDot MediumDashDotDot,
          markers=ArrowDown Asterisk Circle CircleFilled Diamond
                  DiamondFilled GreaterThan Hash HomeDown Ibeam Plus
                  Square SquareFilled Star StarFilled Tack Tilde
                  Triangle TriangleFilled Union X Y Z)

ods listing style=GenderStyle;

proc transreg data=sashelp.class;
   model identity(weight) = class(sex / zero=none) | identity(height);
run;
```

More style elements are changed than are displayed by the graph. However, the **ModStyle** macro options in this example illustrate some of the flexibility that you have for changing the **GraphData***n* elements. You can specify colors by name or by **cx***rrggbb* value, or you can use the colors that are predefined with the style. You can specify FILLCOLORS=COLORS when you want the fill colors to match what you specified for the COLORS= option. A number of different line styles and markers are available for you to use.

Figure 20 Survival Plot

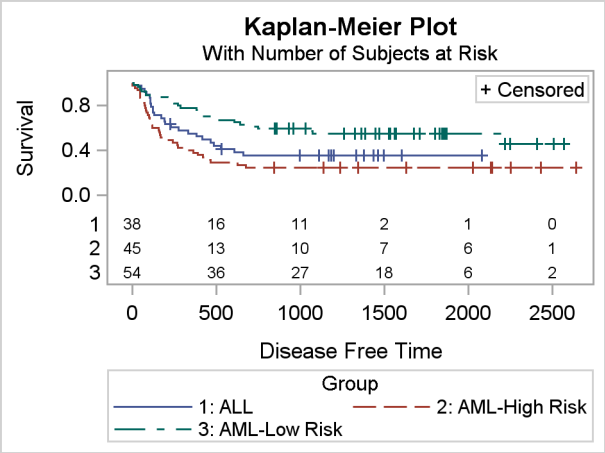
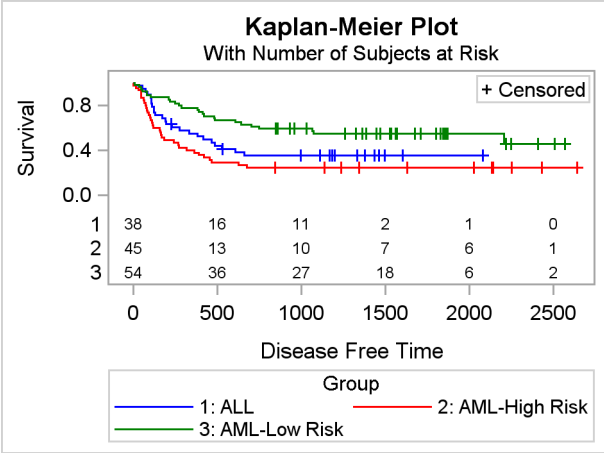


Figure 21 Modified Survival Plot



Color Changes in a Survival Plot

The section "Understanding Conditional Template Logic" on page 18 discussed changing the title of the survival estimate plot to "Kaplan-Meier Plot" by changing three EntryTitle statements. This section assumes that change has been made. The following example uses PROC LIFETEST to produce a survival plot with the number of subjects at risk and multiple comparisons of survival curves:

```
proc format;
   value risk 1='ALL' 2='AML-Low Risk' 3='AML-High Risk';
run;

data BMT;
   input Group T Status @@;
   format Group risk.;
   label T='Disease Free Time';
   datalines;
1 2081 0 1 1602 0 1 1496 0 1 1462 0 1 1433 0
1 1377 0 1 1330 0 1  996 0 1  226 0 1 1199 0

   ... more lines ...

3  113 1 3  363 1
;

proc lifetest data=BMT plots=survival(atrisk=0 to 2500 by 500);
   time T * Status(0);
   strata Group / test=logrank adjust=sidak;
   run;
```

The results are displayed in Figure 20.

The following steps use the **ModStyle** macro to change the colors of the survival curves to pure shades of blue, red, and green and to re-create the plot:

```
%modstyle(parent=statistical, name=SurvivalStyle,
          colors=blue red green)

ods listing style=SurvivalStyle;

proc lifetest data=BMT plots=survival(atrisk=0 to 2500 by 500);
   time T * Status(0);
   strata Group / test=logrank adjust=sidak;
   run;
```

The results are displayed in Figure 21.

For further discussion of the **ModStyle** macro, see the section "ModStyle Macro" on page 29.

Direct Style Modifications

This example directly modifies a style template. First, use PROC TEMPLATE to display the style that you want to change:

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

The first two lines of the results are as follows:

```
define style Styles.Statistical;
   parent = styles.default;
```

The STATISTICAL style inherits many of its elements from the DEFAULT style. The preceding lines are followed by many lines that specify differences between the parent DEFAULT style and the STATISTICAL style. To see more fully how the STATISTICAL style is defined, you also need to look at its parent, as follows:

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

A small part of the results are as follows:

```
class GraphColors
   'gcdata' = cx000000
   'gdata' = cxB9CFE7
   . . .
   'gcdata1' = cx2A25D9
   . . .
   'gdata1' = cx7C95CA;
   . . .
class GraphDataDefault /
   endcolor = GraphColors('gramp3cend')
   neutralcolor = GraphColors('gramp3cneutral')
   startcolor = GraphColors('gramp3cstart')
   markersize = 7px
   markersymbol = "circle"
   linethickness = 1px
   linestyle = 1
   contrastcolor = GraphColors('gcdata')
   color = GraphColors('gdata');
class GraphData1 /
   markersymbol = "circle"
   linestyle = 1
   contrastcolor = GraphColors('gcdata1')
   color = GraphColors('gdata1');
class GraphData2 /
   markersymbol = "plus"
   linestyle = 4
   contrastcolor = GraphColors('gcdata2')
   color = GraphColors('gdata2');
. . .
```

The style elements displayed in the preceding results are not specified directly in the STATISTICAL style, so they come from the parent DEFAULT style. The `GraphDataDefault` class defines the default marker, marker size, line style, line thickness, marker and line colors, and other colors. The `GraphData1` and `GraphData2` classes define the default appearance of the first two groups. You can convert the hexadecimal colors to decimal as follows:

```
data x;
   input cx $2. (Red Green Blue) (hex2.);
   datalines;
cx2A25D9
cx7C95CA
;

proc print;
run;
```

The results are displayed in Figure 22. (Alternatively, you can convert decimal to hex by using PROC PRINT and the statement: `format red green blue hex2.;`)

Figure 22 Colors from the Style

Obs	cx	Red	Green	Blue
1	cx	42	37	217
2	cx	124	149	202

You can see that both colors are dominated by the blue component. The first value (the color that is applied to filled areas) is a purer shade of blue; the contrast color (which is applied to markers and lines) has greater contributions from the other colors.

You can make a new style that changes aspects of style elements. For example, the following statements change the `GraphDataDefault` style element:

```
proc template;
   define style Styles.MyStyle;
      parent = Styles.statistical;
      class GraphDataDefault /
         endcolor = GraphColors('gramp3cend')
         neutralcolor = GraphColors('gramp3cneutral')
         startcolor = GraphColors('gramp3cstart')
         markersize = 7px
         markersymbol = "square"
         linethickness = 1px
         linestyle = 1
         contrastcolor = blue
         color = cyan;
   end;
run;
```

The following steps use the old and new style with the LISTING destination:

```
ods listing style=statistical;


proc transreg data=sashelp.class;
   model identity(weight) = pbspline(height);
run;

ods listing style=MyStyle;

proc transreg data=sashelp.class;
   model identity(weight) = pbspline(height);
run;
```

The results are displayed in Figure 23 and Figure 24. You can see in the plot that the style affects the first group of observations, the females. Although you can make any style changes that you want, be aware that ad hoc changes such as these might not go well with the other colors and elements in the style.

Figure 23 Default `GraphDataDefault`

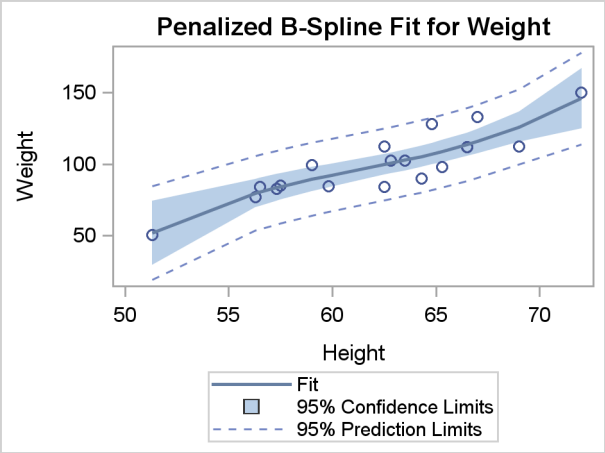
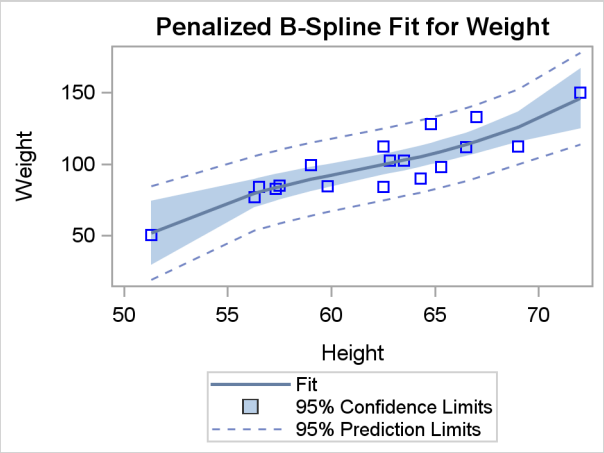


Figure 24 Modified `GraphDataDefault`



ModStyle Macro

The **`ModStyle`** macro was written by Robert E. Derr at SAS to provide easy ways to customize the style elements (**`GraphData1`**—**`GraphData`***n*) that control how groups of observations are distinguished. The **`ModStyle`** macro has the following options:

COLORS=*color-list*
specifies a space-delimited list of colors for markers and lines. If you do not specify this option, then the colors from the parent style are used. You can specify the colors using any SAS color notation such as **`cx`***rrggbb*.

COLORS=GRAYS generates seven distinguishable grayscale colors from blackest to whitest. The colors should be mixed up to be more easily distinguished when you need fewer colors, but you can do that with your own COLORS= list. The HLS (hue/light/saturation) coding generates colors by setting hue and saturation to 0 and incrementing the lightness for each gray. You can also use the keywords BLUES, PURPLES, MAGENTAS, REDS, ORANGES, YELLOWS, GREENS, and CYANS to generate seven colors with a fixed hue and a saturation of AA (hex).

COLORS=SHADES INT generates seven colors as described previously, except that you specify an integer $0 \leq \text{INT} < 360$. See *SAS/GRAPH: Reference*. The available hues include: GRAY, GREY, BLUE=0, PURPLE=30, MAGENTA=60, RED=120, ORANGE=150, YELLOW=180, GREEN=240, and CYAN=300.

DISPLAY=*n*
specifies whether to display the generated template. By default, the template is not displayed. Specify DISPLAY=1 to display the generated template.

FILLCOLORS=*color-list*
specifies a space-delimited list of colors for bands and fills. If you do not specify this option, then the colors from the parent style are used.

Fill colors from the parent style are designed to work well with the colors from the parent style. If you specify a COLORS= list, then you might want to redefine the FILLCOLORS= list as well. You need to have at least as many fill colors as you have colors (any extra fill colors are ignored). Two shortcuts are available: FILLCOLORS=COLORS uses the COLORS= colors for the fills (your confidence bands should have transparency for this to be useful) and FILLCOLORS=LIGHTCOLORS modifies the lightness associated with each color generated by COLORS=SHADES (this is allowed only with COLORS=SHADES).

LINESTYLES=*line-style-list*
specifies a space-delimited list of line style numbers. The default is:

```
LineStyles=Solid MediumDash MediumDashShortDash LongDash DashDashDot LongDashShortDash
           DashDotDot Dash ShortDashDot MediumDashDotDot ShortDash
```

Line style numbers can range from 1 to 46. Some line styles have names associated with them. You can specify either the name or the number for the following number/name pairs: 1 Solid, 2 ShortDash, 4 MediumDash, 5 LongDash, 8 MediumDashShortDash, 14 DashDashDot, 15 DashDotDot, 20 Dash, 26 LongDashShortDash, 34 Dot, 35 ThinDot, 41 ShortDashDot, 42 MediumDashDotDot.

MARKERS=*marker-list*

specifies a space-delimited list of marker symbols. By default, **`Markers=Circle Plus X Triangle Square Asterisk Diamond`**. The available marker symbols are listed in *SAS/GRAPH: Graph Template Language Reference*. Two shortcuts are available: MARKERS=FILLED is an alias for the specification **`Markers=CircleFilled TriangleFilled SquareFilled DiamondFilled StarFilled HomeDownFilled`**, and MARKERS=EMPTY is an alias for the specification **`Markers=Circle Triangle Square Diamond Star HomeDown`**.

NAME=*style-name*

specifies the name of the new style that you are creating. This name is used when you specify the style in an ODS destination statement (for example, ODS HTML STYLE=*style-name*). The default is NAME=NEWSTYLE.

NUMBEROFGROUPS=*n*

specifies n, the number of **`GraphData`***n* style elements to create. The **`GraphData1`**–**`GraphData`***n* style elements contain n combinations of colors, markers, and line styles. By default, 32 combinations are created.

PARENT=*style-name*

specifies the parent style. The new style inherits most of its attributes from the parent style. The default is PARENT=DEFAULT (which is the default style for HTML).

TYPE=*type-specification*

specifies how your new style cycles through colors, markers, and line styles. The default is TYPE=LMbyC.

These first three methods work well with all plots, because cycling line styles and markers together ensures that both scatterplot markers and series plot lines are distinguishable:

CLM

cycles through colors, line styles, and markers simultaneously. The first group uses the first color, line style, and marker; the second group uses the second color, line style, and marker; and so on. This is the method used by the ODS Graphics styles.

LMbyC

fixes line style and marker, cycles through colors, and then moves to the next line style and marker. This is the default and creates a style where the first groups are distinguished entirely by color.

CbyLM

fixes color, cycles through line style and marker, and then moves to the next color. This option uses the smaller of the number of line styles or the number of markers when cycling within a color.

The following two methods might not work well with all plots:

CbyLbyM

fixes color and line style, then cycles through markers, increments line style, and then cycles through markers. After all line styles have been used, then this option moves to the next color and continues.

LbyMbyC

fixes line style and marker, then cycles through colors, increments marker, and then cycles through colors. After all markers have been used, then this option moves to the next line style and continues. This is closest to the legacy SAS/GRAPH method.

The following steps show more examples of how this macro is used:

```
* First few groups are distinguished by line style,
  but later by changing color;

%modstyle(name=NewStatStyle, parent=Statistical,
          type=CbyLbyM, markers=circle)
```

```
* Grayscale with non-transparent confidence bands;

%modstyle(name=NewStatStyle, parent=Statistical, type=LMbyC,
          colors=grays, fillcolors=lightcolors)

* Blue filled circles;

%modstyle(name=NewStatStyle, parent=Statistical, type=LMbyC,
          colors=shades 0, markers=circlefilled)

%modstyle(name=NewStatStyle, parent=Statistical, type=LMbyC,
          colors=shades blue, markers=circlefilled)

%modstyle(name=NewStatStyle, parent=Statistical, type=LMbyC,
          colors=blues, markers=circlefilled)
```

You can use the following artificial data and test program to illustrate the effects of using the **ModStyle** macro:

```
data x;
   do y = 40 to 1 by -1;
      group = 'Group' || put(41 - y, 2. -L);
      do x = 0 to 10 by 5;
         if x = 10 then do; z = 11; l = group; end;
         else           do; z = .;  l = ' ';   end;
         output;
      end;
   end;
run;

ods listing style=NewStatStyle;

proc sgplot data=x;
   title 'Colors, Markers, Lines Patterns for Groups';
   series  y=y x=x / group=group markers;
   scatter y=y x=z / group=group markerchar=l;
run;
```

ModTmplt Macro

You can use the **ModTmplt** macro to insert BY line information, titles, and footnotes in ODS Graphics. You can also use it to remove titles and perform other template modifications. When you want to display BY line information in your graphs, you use the **ModTmplt** macro to both modify the template and run the procedure. In that case, the **ModTmplt** macro requires you to construct a SAS macro called **MyGraph** that contains the SAS procedure that needs to be run. This code must be in a macro so that the **ModTmplt** macro can call it. The following example illustrates this usage of the macro:

```
proc sort data=sashelp.class out=class;
   by sex;
run;

%macro mygraph;
proc transreg data=__bydata;
   model identity(weight) = identity(height);
%mend;

%modtmplt(by=sex, data=class, template=Stat.Transreg.Graphics)
```

Notice that the BY and RUN statements are *not* specified in the **MyGraph** macro. Also notice that you must use DATA=__BYDATA with the procedure call in the **MyGraph** macro and specify the real input data set in the DATA= option of the **ModTmplt** macro.

The **ModTmplt** macro outputs the specified template or templates to a file, adds either an EntryTitle or EntryFootNote statement that adds the BY line information, and then runs the **MyGraph** macro once for each BY group. In the end, it deletes the modified template. Use the STEPS= option if you do not want to have all of these steps performed.

Figure 25 First BY Group

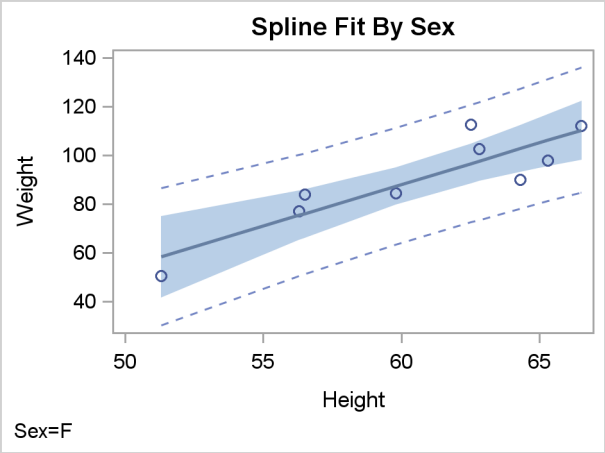
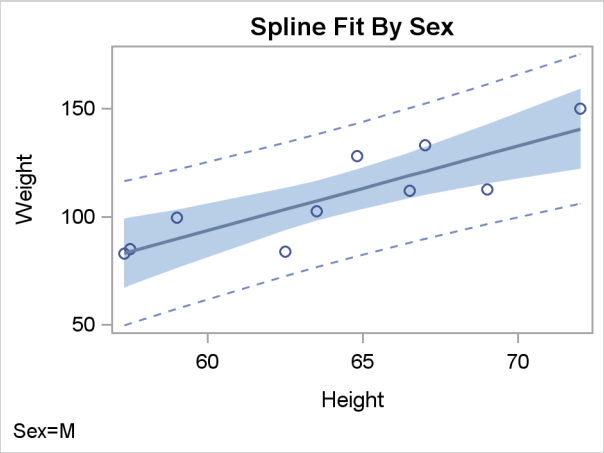


Figure 26 Second BY Group



In contrast, if you only want to add system titles to the graph, run the macro only to modify the template by specifying the STEPS=T option. Then run your procedure in the normal way. You can optionally run the macro again to delete the modified template by specifying the STEPS=D option. The following example uses OPTIONS=REPLACE to replace the default graph titles with the system title:

```
title 'Spline Fit By Sex';

%modtmplt(template=Stat.transreg, options=replace, steps=t)

proc transreg data=sashelp.class;
   model ide(weight) = class(sex / zero=none) | spline(height);
   run;

%modtmplt(template=Stat.transreg, steps=d)
```

The results of this step are not shown; however, see page 12 for a similar example. The following example both replaces the title and adds BY line information as a footnote:

```
ods graphics on / maxlegendarea=0;

title 'Spline Fit By Sex';

proc sort data=sashelp.class out=class;
   by sex;
run;

%macro mygraph;
proc transreg data=__bydata;
   model identity(weight) = identity(height);
   ods select fitplot;
%mend;

%modtmplt(by=sex, data=class, options=replace, template=Stat.Transreg.Graphics)

ods graphics on;
```

This example also uses the option MAXLEGENDAREA=0 to suppress the legend. The results are displayed in Figure 25 and Figure 26.

The **ModTmplt** macro has the following options:

BY=*by-variable-list*
: specifies the list of BY variables. Also see BYLIST=. When graphs are produced (by default or when the STEPS= value contains 'G'), you must specify the BY= option. Otherwise, when you are only modifying the template, you do not need to specify the BY= option.

BYLIST=*by-statement-list*

specifies the full syntax of the BY statement. You can specify a full BY statement syntax including the DESCENDING or NOTSORTED options. If only BY variables are needed, specify only BY=. If you also need options, then specify the BY variables in the BY= option and the full syntax in the BYLIST= option (for example, specify BY=A B and BYLIST=A DESCENDING B).

DATA=*SAS-data-set*

specifies the input SAS data set. If you do not specify the DATA= option, the macro uses the most recently created SAS data set.

FILE=*filename*

specifies the file in which to store the original templates. This is a temporary file. You can specify either a quoted file name or the name from a FILENAME statement that you provide before you call the macro. The default is *"template.txt"*.

OPTIONS=*options*

specifies one or more of the following options (case is ignored):

LOG

displays a note in the SAS log when each BY group has finished.

FIRST

adds the EntryTitle or EntryFootNote statements as the first titles or footnotes. By default, the statements are added after the last titles or footnotes. Most graph templates provided by SAS do not use footnotes; so this option usually affects only entry titles.

NOQUOTES

specifies that the values of the system titles and footnotes are to be moved to the EntryTitle or EntryFootNote statements without the outer quotation marks. With OPTIONS=NOQUOTES, you can specify options in the titles or footnotes in addition to the text. However, you must ensure that you quote the text that provides the actual title or footnote.

The following is an example of an ordinary footnote:

```
footnote "My Footer";
```

With this FOOTNOTE statement and without OPTIONS=NOQUOTES, the macro creates the following EntryFootNote statement:

```
entryfootnote "My Footer";
```

The following footnotes are used with OPTIONS=NOQUOTES:

```
footnote 'halign=left "My Footer"';
footnote2 '"My Second Footer"';
```

With these FOOTNOTE statements and OPTIONS=NOQUOTES, the macro creates the following EntryFootNote statements:

```
entryfootnote halign=left "My Footer";
entryfootnote "My Second Footer";
```

REPLACE

replaces the unconditionally added entry titles and entry footnotes in the templates (those that are not part of IF or ELSE statements) with the system titles and footnotes. The system titles and footnotes are those that are specified in the TITLE or FOOTNOTE statements. You can instead use the TITLES=*SAS-data-set* option to specify titles and footnotes with a data set. If OPTIONS=REPLACE is specified, then OPTIONS=TITLES is ignored.

SOURCE

displays the generated source code. By default, the template source code is not displayed.

TITLES

displays the system titles and footnotes with the graphs. The system titles and footnotes are those that are specified in the TITLE or FOOTNOTE statements. You can instead use the TITLES=*SAS-data-set* option to specify titles and footnotes with a data set. If you also specify OPTIONS=FIRST, the system titles

and footnotes are inserted before the previously existing entry titles and entry footnotes in the templates. Otherwise, they are inserted at the end.

You can specify OPTIONS=TITLES or OPTIONS=REPLACE, or insert BY lines, or do both. If you do both, and you do not like where the BY line is inserted relative to your titles and footnotes, just specify OPTIONS=NOQUOTES and _ByLine0 to place the BY line wherever you choose. The following TITLE statements illustrate:

```
title1 '"My First Title"';
title2 '_byline0';
title3 '"My Last Title"';
```

Also, you can embed BY information in a title or a footnote, again with OPTIONS=NOQUOTES. For example:

```
title '"Spline Fit By Sex, " _byline0';
```

When _ByLine0 is specified in any of the titles or footnotes, then the usual BY line is not added.

The following example removes all titles and footnotes:

```
footnote;
title;
%modtmplt(options=replace, template=Stat.Transreg.Graphics, steps=t)
```

STATEMENT=*entry-statement-fragment*

specifies the statement that contains the BY line that gets added to the template along with any statement options. The default is **`Statement=EntryFootNote halign=left TextAttrs=GraphValueText`**. Other examples include:

```
Statement=EntryTitle
Statement=EntryFootNote halign=left TextAttrs=GraphLabelText
```

STEPS=*steps*

specifies the macro steps to run. Case and white space are ignored. the macro modifies the templates (when 'T' is specified), produces the graphs for each BY group (when 'G' is specified), and deletes the modified templates (when 'D' is specified). The default is STEPS=TGD. You can instead have it perform a subset of these three tasks by specifying a subset of terms in the STEPS= option.

When you use the **`ModTmplt`** macro to add BY lines, you usually do not need to delete the templates before you run your procedure again in the normal way. The template modification inserts the BY line through a macro variable and an MVAR statement. When the macro variable _ByLine0 is undefined, the EntryTitle or EntryFootNote statement drops out as if it were not there at all.

STMTOPTS1= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS2= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS3= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS4= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS5= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS6= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS7= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS8= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS9= *n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >
STMTOPTS10=*n* **ADD** | **REPLACE** | **DELETE** | **BEFORE** | **AFTER** *statement-name* < *options* >

These ten options add or replace options in up to 10 selected statements. The following example produces Figure 27:

```
%modtmplt(template=Stat.glm.graphics.residualhistogram, steps=t,
          stmtopts1=.  add  discretelegend  autoalign=(topleft),
          stmtopts2=1  add  densityplot     legendlabel='Normal Density',
          stmtopts3=2  add  densityplot     legendlabel='Kernel Density',
          stmtopts4=1  add  overlay         yaxisopts=(griddisplay=on)
                            yaxisopts=(label='Normal and Kernel Density'))
```

Figure 27 Modified Residual Histogram

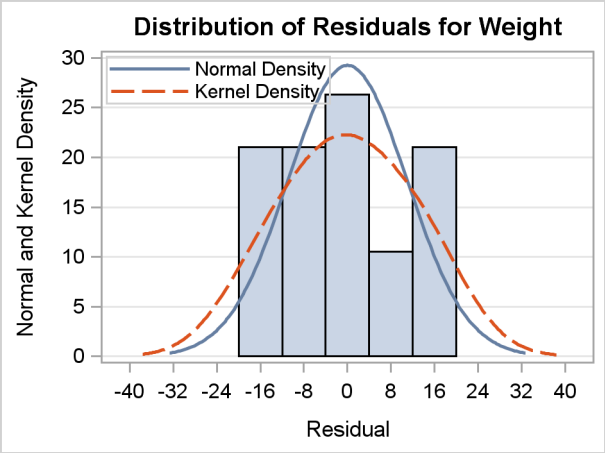
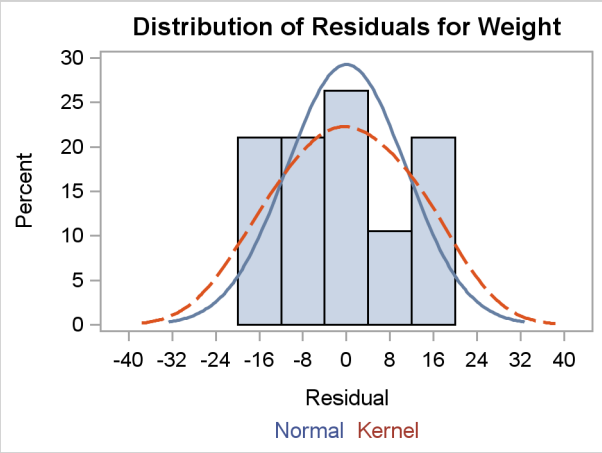


Figure 28 Footnote, No Legend




```
proc glm plots=diagnostics(unpack) data=sashelp.class;
   model weight = height;
run;

%modtmplt(template=Stat.glm.graphics.residualhistogram, steps=d)
```

These options require you to specify a series of values. The first value is the statement number (or missing to modify options on all statements that match the statement name). The second value is: ADD, REPLACE, DELETE, BEFORE, or AFTER. When the second value is ADD or REPLACE, it controls whether you add new options or replace existing options. Alternatively, the second value can be BEFORE or AFTER to add a new statement before or after the named statement. When the value is DELETE, the corresponding statement is deleted. The third value is a statement name. All remaining options are options for the statement named by the third value (with ADD and REPLACE) or for a new statement (with BEFORE and AFTER). In the STMTOPTS1= example, an option is added to all DiscreteLegend statements. In the STMTOPTS2= example, an option is added to the first DensityPlot statement. In the STMTOPTS4= example, an option is added to the LAYOUT OVERLAY statement. In most cases, the statement name is the first name that begins the statement. The LAYOUT statement is an exception. In the case of layouts, specify the second name (OVERLAY, GRIDDED, LATTICE, and so on) for the third value. Note that a statement such as `if (expression) EntryTitle...;` is an IF statement not an EntryTitle statement.

If an option is specified multiple times on a GTL statement, the last specification overrides previous specifications. Hence, you do not need to know and respecify all of the options. You can just add an option to the end, and it overrides the previous value. You can use these options only to modify statements that contain a slash, and only to modify the options that come after the slash. Note that in STMTOPTS4=, the YAXISOPTS= option is specified twice. It could have been equivalently specified once as follows:

```
yaxisopts=(griddisplay=on label='Normal and Kernel Density'))
```

The actual specification adds the GRIDDISPLAY=ON to the Y axis options (which by default has only a label specification). The old label is unchanged until the LABEL= option in the second YAXISOPTS= specification overrides it. In other words, YAXISOPTS=(GRIDDISPLAY=ON) augments the old YAXISOPTS= option; it does not replace it.

The following steps delete the legend and instead provide a footnote and produce Figure 28:

```
%modtmplt(template=Stat.glm.graphics.residualhistogram, steps=t,
          stmtopts1=. delete discretelegend,
          stmtopts2=1 after  begingraph entryfootnote
                      textattrs=GraphLabelText(color=cx445694) 'Normal  '
                      textattrs=GraphLabelText(color=cxA23A2E) 'Kernel')

proc glm plots=diagnostics(unpack) data=sashelp.class;
   model weight = height;
run;

%modtmplt(template=Stat.glm.graphics.residualhistogram, steps=d)
```

Figure 29 First BY Group

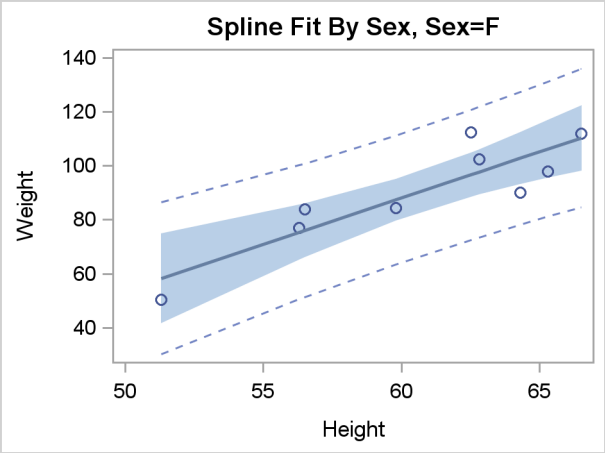
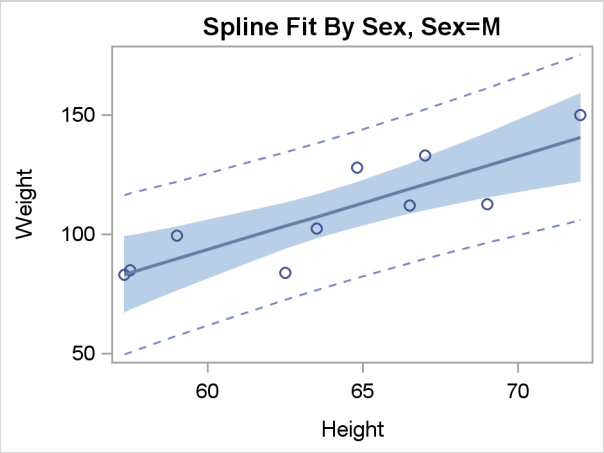


Figure 30 Second BY Group



TEMPLATE=*SAS-template*
specifies the name of the template to modify. You can specify just the first few levels to modify a series of templates. For example, to modify all of PROC REG's graph templates, specify `Template=Stat.Reg.Graphics` This option is required.

TITLES=*SAS-data-set*
specifies a data set that contains titles or footnotes or both. By default, when the system titles or footnotes are used (when OPTIONS=TITLES or OPTIONS=REPLACE is specified), PROC SQL is used to determine the titles and footnotes. You can instead create this data set yourself so that you can set the graph titles independently from the system titles and footnotes. The data set must contain two variables: Type (Type='T' for titles and Type='F' for footnotes), and Text, which contains the titles and footnotes. Other variables are ignored. Specify the titles and footnotes in the order in which you want them to appear.

TITLEOPTS=*entry-statement-options*
specifies the options for system titles and footnotes. For example, you can specify the HALIGN= and TEXTATTRS= options as in the STATEMENT= option. By default, no title options are used. With OPTIONS=NOQUOTES, you can specify options individually.

The following example illustrates the use of the OPTIONS=REPLACE, OPTIONS=NOQUOTES, and TITLES= options:

```
ods graphics on / maxlegendarea=0;

title 'Regression Analysis';

data title;
   type = 'T';
   text = '"Spline Fit By Sex, " _byline0';
run;

proc sort data=sashelp.class out=class;
   by sex;
run;

%macro mygraph;
proc transreg data=__bydata;
   model identity(weight) = identity(height);
   ods select fitplot;
%mend;

%modtmplt(by=sex, data=class, options=replace noquotes, titles=title,
          template=Stat.Transreg.Graphics)

ods graphics on;
```

The results are displayed in Figure 29 and Figure 30. Note that the listing, HTML, and other output have the title "Regression Analysis". The graph titles come from the TITLES=title data set.

The following statements modify the PROC FACTOR scree plot template and produce the same graph that is displayed in Figure 3:

```
%modtmplt(template=Stat.Factor.Graphics.ScreePlot1, steps=t,
          stmtopts1=. replace overlay
             yaxisopts=(label="(*ESC*){Unicode Lambda}")
             xaxisopts=(label="Factor Number"
                linearopts=(tickvaluelist=(1 2 3 4 5 6 7 8 9 10))),
          stmtopts2=. replace
             entrytitle 'Eigenvalue ((*ESC*){Unicode Lambda}) Plot')

proc factor data=sashelp.cars plots(unpack)=scree;
run;

%modtmplt(template=Stat.Factor.Graphics.ScreePlot1, steps=d)
```

The following statements modify the PROC GLIMMIX box plot template and produce the same graph that is displayed in Figure 7:

```
%let DateTag = Acme 01Apr2008;

%modtmplt(template=Stat.Glimmix.Graphics.BoxPlot, steps=t,
          stmtopts1=. before begingraph mvar datetag,
          stmtopts2=. before endgraph entryfootnote halign=left datetag)

proc glimmix data=sashelp.class plots=boxplot;
    class sex;
    model height = sex;
run;

%modtmplt(template=Stat.Glimmix.Graphics.BoxPlot, steps=d)
```

The following statements modify the PROC KDE scatter plot template and produce the same graph that is displayed in Figure 9:

```
%modtmplt(template=Stat.KDE.Graphics.ScatterPlot, steps=t,
          stmtopts1=1 replace entrytitle "Distribution of " _VAR2LABEL
                      " by " _VAR1LABEL,
          stmtopts2=1 replace overlay
                      xaxisopts=(offsetmin=0.05 offsetmax=0.05 griddisplay=on)
                      yaxisopts=(offsetmin=0.05 offsetmax=0.05 griddisplay=on),
          stmtopts3=1 after overlay pbsplineplot x=y y=x /
                      lineattrs=(color=red pattern=2 thickness=1),
          stmtopts4=1 delete scatterplot,
          stmtopts5=1 before endlayout scatterplot
                      x=y y=x / markerattrs=(color=green size=5px
                                symbol=starfilled weight=bold))

proc kde data=sashelp.class;
   label height = 'Class Height' weight = 'Class Weight';
   bivar height weight / plots=scatter;
run;

%modtmplt(template=Stat.KDE.Graphics.ScatterPlot, steps=d)
```

The following statements modify the PROC GLM residual histogram template and produce the same graph that is displayed in Figure 13:

```
%modtmplt(template=Stat.GLM.Graphics.ResidualHistogram, steps=t,
          stmtopts1=. replace discretelegend)

proc glm plots=diagnostics(unpack) data=sashelp.class;
   model weight = height;
   ods output residualhistogram=hr;
run;

%modtmplt(template=Stat.GLM.Graphics.ResidualHistogram, steps=d)
```

There is one limitation of the **ModTmplt** macro. You cannot use it with BY processing and the ODS Document and then replay the graphs. This is because the value of the macro variable that contains the BY group information does not persist for replay.

Conclusions

The GTL is a powerful language for producing modern statistical graphics. SAS provides the default templates for graphs, so you do not need to know any details about templates to create statistical graphics. However, with some understanding of the GTL, you can modify the default templates to permanently change graphs. Although the templates are often large and complicated, with a little knowledge you can easily isolate and modify the relevant parts without understanding the myriad of surrounding details.

Recommended Reading

For a parallel introduction to the GTL and the statistical graphics procedures see:
http://support.sas.com/publishing/authors/kuhfeld.html.

More information about ODS, ODS Graphics, the GTL, and SAS/STAT software is available on the Web at:
http://support.sas.com/documentation/,
http://support.sas.com/documentation/onlinedoc/base/index.html,
http://support.sas.com/documentation/onlinedoc/graph/index.html,
and http://support.sas.com/documentation/onlinedoc/stat/index.html.

To learn more about ODS, see Chapter 20, "Using the Output Delivery System" (*SAS/STAT User's Guide*).

To learn more about ODS Graphics, see Chapter 21, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*).

For introductory information about ODS Graphics, see the documentation section "A Primer on ODS Statistical Graphics" (Chapter 21, *SAS/STAT User's Guide*).

For complete ODS documentation, see the *SAS Output Delivery System: User's Guide*.

For complete GTL documentation, see the *SAS/GRAPH: Graph Template Language User's Guide* and *SAS/GRAPH: Graph Template Language Reference*.

For complete documentation about the Graphics Editor, see the *Getting Started with the SAS/GRAPH Statistical Graphics Editor*.

For information about the statistical graphics procedures, see the *SAS/GRAPH: Statistical Graphics Procedures Guide*.

The **ModTmplt** macro is available from http://support.sas.com/kb/37/503.html and the **ModStyle** macro is available from http://support.sas.com/kb/37/508.html.

Contact Information

Warren F. Kuhfeld
SAS Institute Inc.
S3018 SAS Campus Drive
Cary, NC, 27513
(919) 531-7922
Warren.Kuhfeld@sas.com

Appendix I: Template Complexity

Many graphs produced by ODS Graphics are complicated. They might have multiple panels, multiple graph elements, tables of statistics, legends, and so on. Complicated graphs require complicated programs. The aphorism "the devil is in the details" is indeed true for graph templates. Some have incredible levels of detail so that they can be used in a number of different situations and handle many options. Before ODS Graphics, you had to write your own programs to produce graphs. You might have needed hundreds or even thousands of lines of code to make a complicated graph, particularly if you were using the annotate facility. Now, this work is done by SAS procedure writers who use the GTL.

The GTL is a powerful language with many statements and options, and often many different ways to accomplish the same thing. Different procedure writers sometimes found different ways to do the same thing. Templates are constructed in many different ways. Some are constructed with SAS macros, so sometimes options appear because they are needed

in other uses of the macro. The template that you see often bears very little resemblance to the template that the developer wrote. The template you see has been compiled by PROC TEMPLATE, digested, and then output in a different format. The template that you see might be very large, complex, and verbose. The template that the developer created is usually more parsimonious and structured. When you look at a template, do not expect that you will be able to find some justification for every statement and every option if you only search hard enough. Nor should you expect to find documentation for every option. Some options are deliberately undocumented because they might change in future releases.

Some templates are complex. Others are very complex. Some are much more complex than they need to be for your particular application. However, you do not have to understand most of that complexity. All you have to do is isolate the parts that you want to change, and change those parts while ignoring the surrounding complexity.

Templates are *not* intended to be like the polished sample code that SAS generally provides. Templates were developed with the goal of producing outstanding graphs, not outstanding templates. Do not expect to see perfect beauty, elegance, consistency, or the most parsimonious use of the language. Do not expect everything to be obvious. Instead, expect to see a powerful program that produces outstanding graphics. Expect the template to be complicated, but much simpler than your old graphics programs.

Appendix II: Displaying Simple Templates

Sometimes when you are working on a template, it is helpful to look at and search other templates to see how the statements and options are used elsewhere. The following program creates a file *templates.sas* with all of the SAS/STAT graphics templates, and displays them in the SAS log in sorted order with the smallest (and hence usually simplest) templates first:

```
proc template;
   source / where=(type='Statgraph') file="templates.sas";
run;

data x;
   n = _n_;
   infile "templates.sas" lrecl=204 pad;
   input line $200.;
   l = lowcase(line);
   if (index(l, 'define') and index(l, 'statgraph')) or index(l, 'endgraph');
run;

data z(keep=line dif);
   set x;
   n1 = n;
   if index(l, 'define') then do;
      i = _n_ + 1;
      set x(keep=n) point=i;
      dif = n - n1;
      line = scan(line, 3, ' ;');
      if lowcase(line) =: 'stat.' then output;
   end;
run;

proc sort;
   by dif;
run;

data _null_;
   set z;
   if _n_ = 1 then call execute('proc template;');
   call execute('source ' || trim(line) || ';');
run;
```


I would like to thank Jeff Cartier, Bob Derr, Anne Jones, Bari Lawhorn, and Bob Rodriguez for helpful comments on earlier drafts of this paper. I would also like to thank Gerardo Hurtado for testing the **`ModTmplt`** macro.