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;
```

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 1 displays a scree plot from the FACTOR procedure. Displaying and modifying its template is simple. Figure 2 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 5.

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 / label;

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

Figure 1 Default Scree Plot

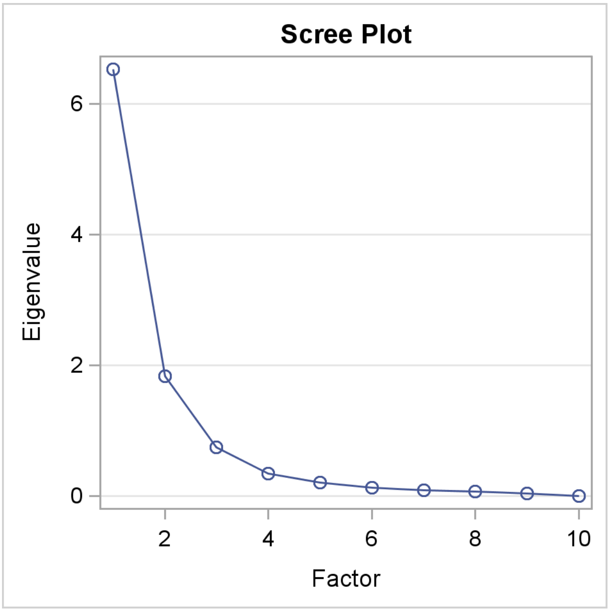
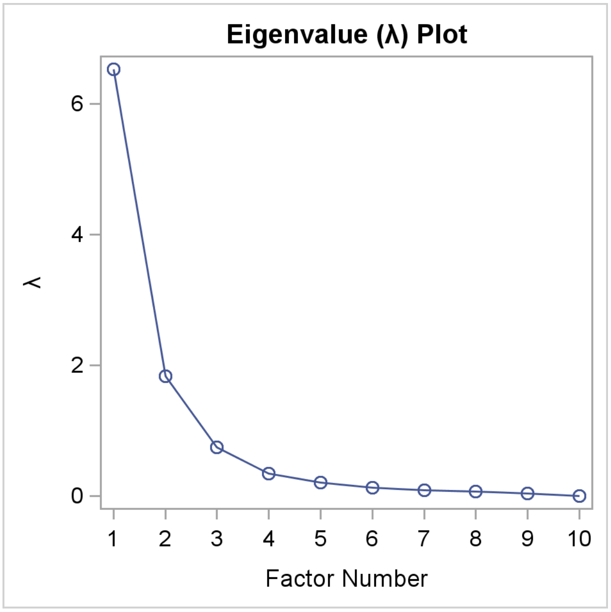


Figure 2 Scree Plot with Modified Title and Axis Labels



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
Label Path: 'The KDE Procedure'.'Bivariate Analysis'.'Height and
Weight'.'KDE.Bivar1.Height_Weight'

Name:        ScatterPlot
Label:       Scatter Plot
Template:    Stat.KDE.Graphics.ScatterPlot
Path:        KDE.Bivar1.Height_Weight.ScatterPlot
Label Path: 'The KDE Procedure'.'Bivariate Analysis'.'Height and Weight'.'Scatter Plot'
```

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 Libraries and the ODS 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=scatter(obs=3);
run;
```

The results are displayed in Figure 3.

Figure 3 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. 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 / label;
```

CHANGING TITLES AND AXIS LABELS

The following step runs PROC FACTOR and produces the eigenvalue (or scree) plot displayed in Figure 1:

```
   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
   Label Path: 'The Factor Procedure'.'Initial Factor Solution'.'Scree and
   Variance Plots'.'Scree Plot'
```

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;
```

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 2:

```
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
Label Path: 'The Glimmix Procedure'.'Box Plots'.'Residuals by Sex'
```

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;
```

Figure 4 Default Box Plots

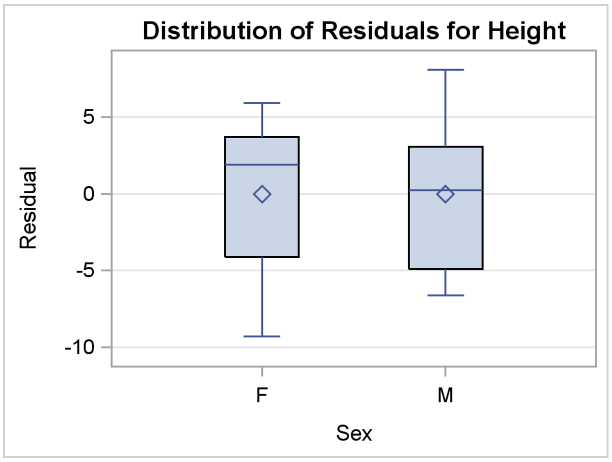


Figure 5 Examining Dynamic Variables

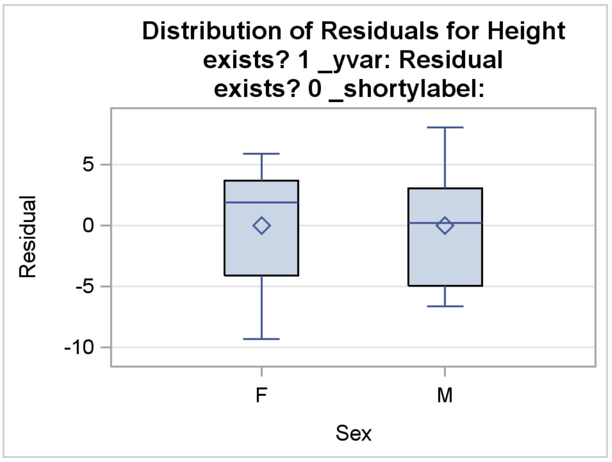
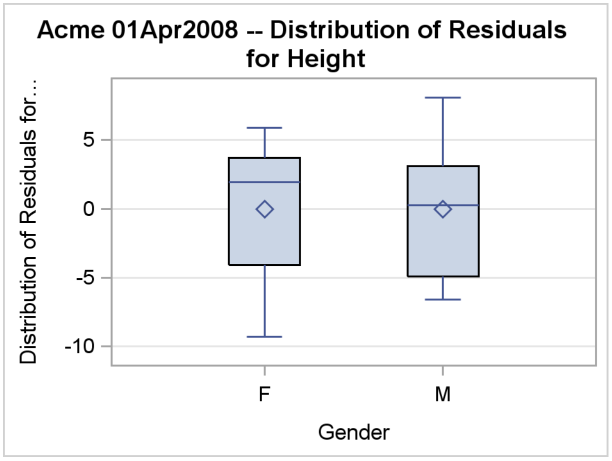


Figure 6 Modified Box Plots



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 7. 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 7 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]

[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.

Your options for modifying templates include the following:

- Explicitly specify character strings for the title or axis labels to override what is currently there.
- Add text to augment what is currently there.
- Use the dynamic variables that are provided in creative ways.

The following example illustrates these options:

```
%let DateTag = Acme 01Apr2008;

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

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

The new graph is displayed in Figure 6.

In this example, the title has a new component, 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:

```
entrytitle "&datetag -- " _TITLE;
```

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 Y axis label is modified to match the original title, using the dynamic variable _Title in another way. The X axis label is set to a constant character string “Gender”. The X axis change is ad hoc, so changes such as this are usually made temporarily. The other changes could be permanent.

The following step restores the default template:

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

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 8. 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 9 with axes passing through the center of the data. The following steps make a highly modified scatter plot:

```
proc means data=sashelp.class noprint;
   var height weight;
   output out=means(where=(_stat_ = 'MEAN'));
run;

data _null_;
   set means;
   call symputx('xref', height);
   call symputx('yref', weight);
run;
```

Figure 8 Transposed Scatter Plot with Labels

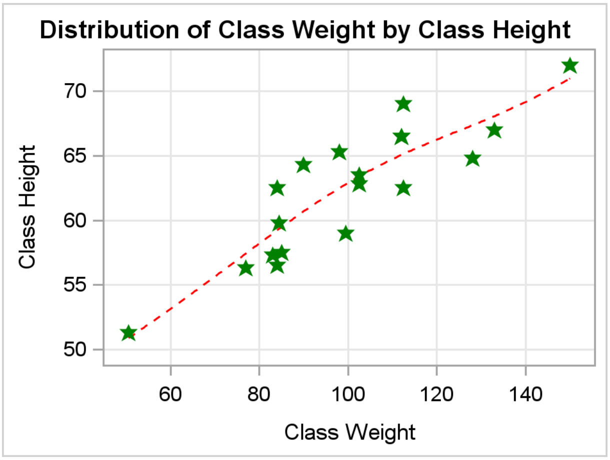
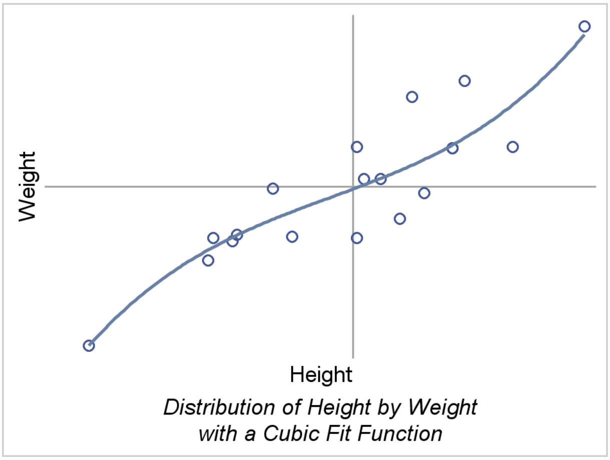


Figure 9 Scatterplot with Numerous Modifications



```
proc template;
   define statgraph Stat.KDE.Graphics.ScatterPlot;
      dynamic _VAR1NAME _VAR1LABEL _VAR2NAME _VAR2LABEL;
      mvar xref yref;
      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=yref;
            referenceline x=xref;
            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 MEANS procedure computes the mean height and mean weight. The DATA step stores each mean in a macro variable. 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 9.

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 10) is used in many procedures.

Figure 10 Default Residual Histogram

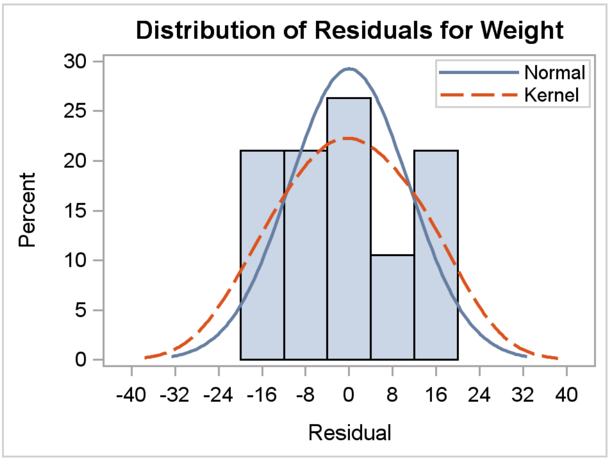
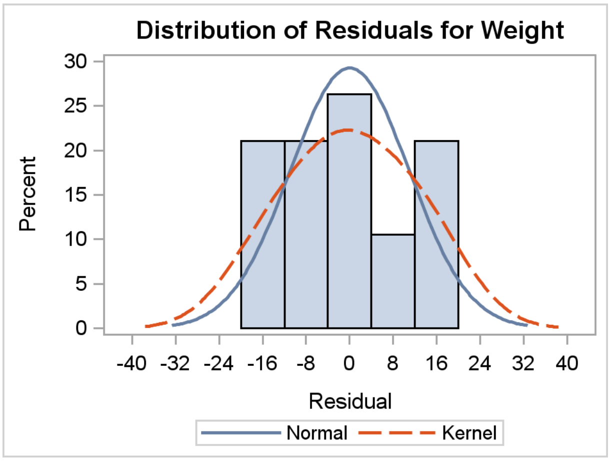


Figure 11 Residual Histogram with Repositioned Legend



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
   Label Path: 'The GLM Procedure'.'Analysis of Variance'.'Weight'.'Diagnostic
   Plots'.'Residual Histogram'
```

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 12. 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 12 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 11.

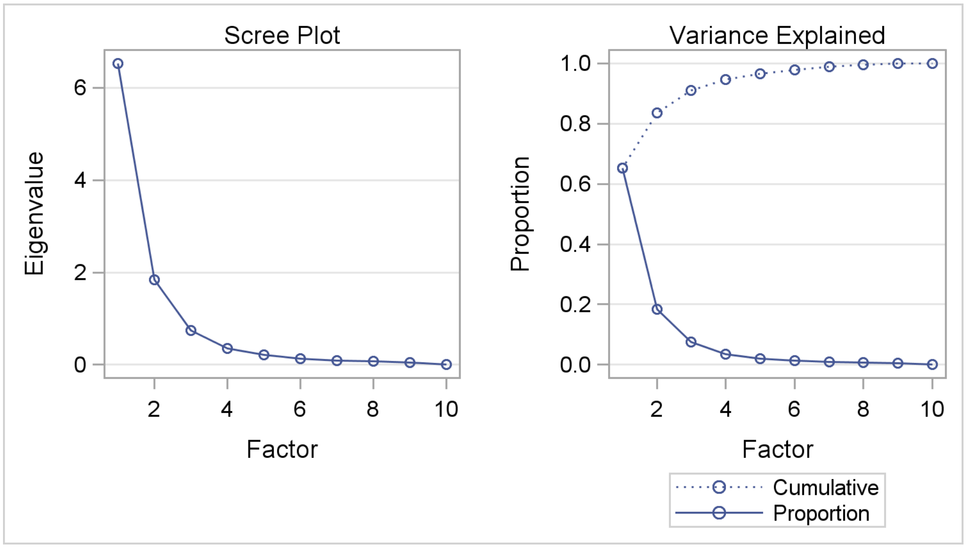
UNDERSTANDING THE LATTICE LAYOUT AND PANELS

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 1 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;
```

The graph is displayed in Figure 13.

Figure 13 Graph with Two Panels



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. 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 "APPENDIX I: TEMPLATE COMPLEXITY" on page 24.

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 fourth title to “Kaplan-Meier Plot”.

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! */
```

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.

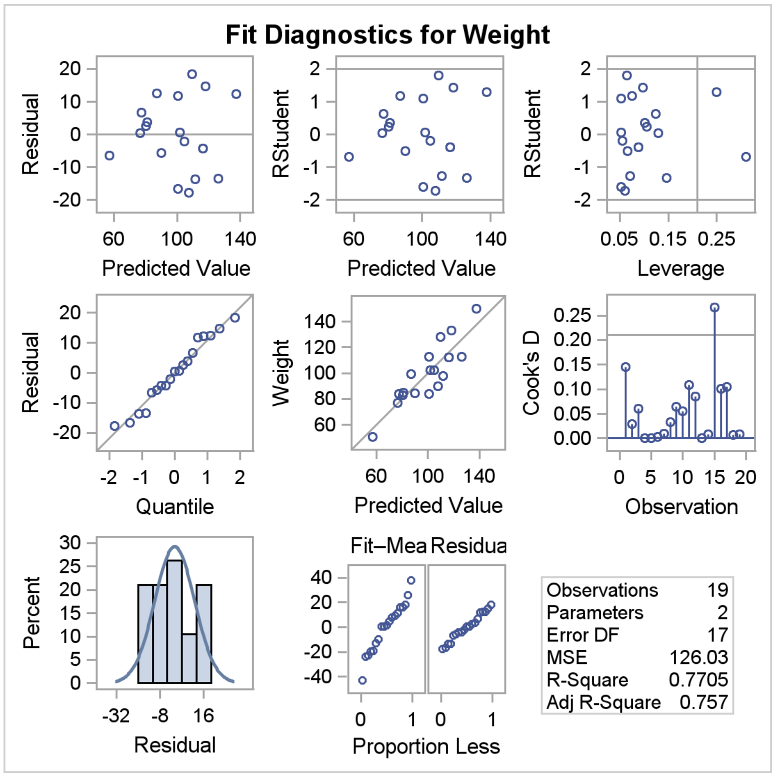
WORKING WITH TEMPLATES FOR PANELED DISPLAYS

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 14.

Figure 14 Diagnostics Panel for the REG Procedure



```
Observations       19
Parameters          2
Error DF           17
MSE            126.03
R-Square       0.7705
Adj R-Square    0.757
```

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 14.

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

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. See the documentation section "Some Common Style Elements" (Chapter 21, *SAS/STAT User's Guide*). The new style uses only color to distinguish lines and groups. The macro is documented in the macro header. The following example illustrates the default use of this macro:

```
%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 is displayed in Figure 15. The females are depicted with a solid blue line and blue circles, and the males are depicted with a solid red line and red circles. If this style modification had not been performed, the second group would have been depicted with a dashed red line and red pluses. The `ModStyle` macro has many other uses, some of which are illustrated in the documentation section "Styles" (Chapter 21, *SAS/STAT User's Guide*).

Figure 15 A Color-Based Line Style

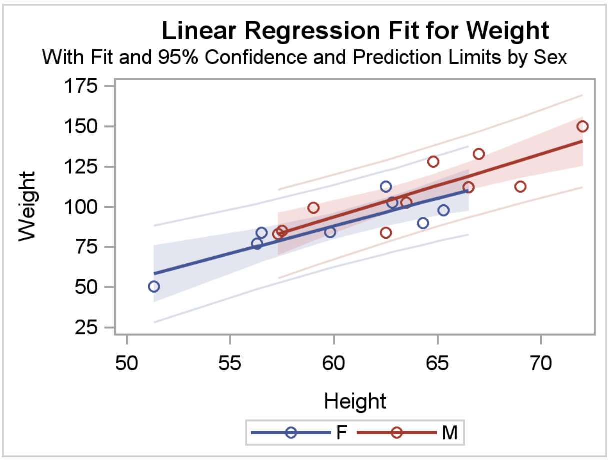


Figure 16 Ad Hoc Style Modification to GraphData1

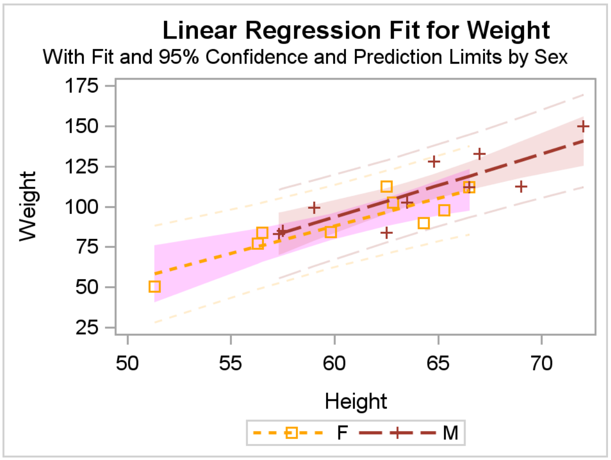


Figure 16 shows results from style changes that you can make by directly modifying a style template. First, use PROC TEMPLATE to see the style that you want to change:

```
   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 very small part of the results are as follows:

```
   class GraphColors
      . . .
      'gcdata1' = cx2A25D9
      . . .
      'gdata1' = cx7C95CA;
   . . .
   class GraphData1 /
      markersymbol = "circle"
      linestyle = 1
      contrastcolor = GraphColors('gcdata1')
      color = GraphColors('gdata1');
   . . .
```

The style elements displayed in the preceding results are not specified directly in the STATISTICAL style, so they come from the parent DEFAULT style. 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 17. (Alternatively, you can convert decimal to hex by using PROC PRINT and the statement: `format red green blue hex2.;`)

Figure 17 The 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 all aspects of the `GraphData1` style element:

```
proc template;
   define style Styles.MyStyle;
      parent = Styles.statistical;
      class GraphData1 /
         markersymbol = "square"
         linestyle = 2
         contrastcolor = orange
         color = magenta;
    end;
run;
```

The following step uses the new style with the LISTING destination:

```
ods listing style=MyStyle;

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

The results are displayed in Figure 16. 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.

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

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 the *SAS/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*.

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;
```