Chapter 1
Overview

Graph Template Language (GTL)

GTL and the Output Delivery System (ODS)

The SAS/GRAPH *Graph Template Language* (GTL) is an extension to the Output Delivery System (ODS) that enables you to create sophisticated analytical graphics that are not available from traditional SAS/GRAPH procedure statements. For example, using the GTL you can generate Model-Fit plots, Distribution Plots, Comparative plots, Prediction Plots, and more.

The graphics produced by the GTL are generated by template definitions that control the graph format and appearance and specify the variable roles to represent in the graph display. The graphs can then be rendered by associating the templates with a data source.

- The GTL templates are defined with PROC TEMPLATE. The GTL includes conditional statements that can be used to determine what graph features are rendered,

layout statements that specify the arrangement of graph features, plot statements that request specific plot types (such as histograms and scatter plots), and text and legend statements that specify titles, footnotes, legends, and other text-based graph elements.

- The GTL templates are rendered using the SGRENDER procedure, which specifies a data source that contains appropriate data values and the template to use for rendering the graph.

- You can also modify predefined GTL templates that the SAS System delivers for use on the SAS statistical procedures. For information about modifying existing templates, refer to *SAS/STAT User's Guide*.

This manual provides a complete reference to the Graphics Template Language. For detailed usage information, consult the *SAS/GRAPH: Graph Template Language User's Guide*.

A Quick Example

The data set SASHELP.CLASS is delivered with the SAS System. It includes data columns named HEIGHT and WEIGHT, which store height and weight measures for a small sample of subjects. The Graphics Template Language can be used to generate a histogram that shows the distribution of weight recorded in that data set:
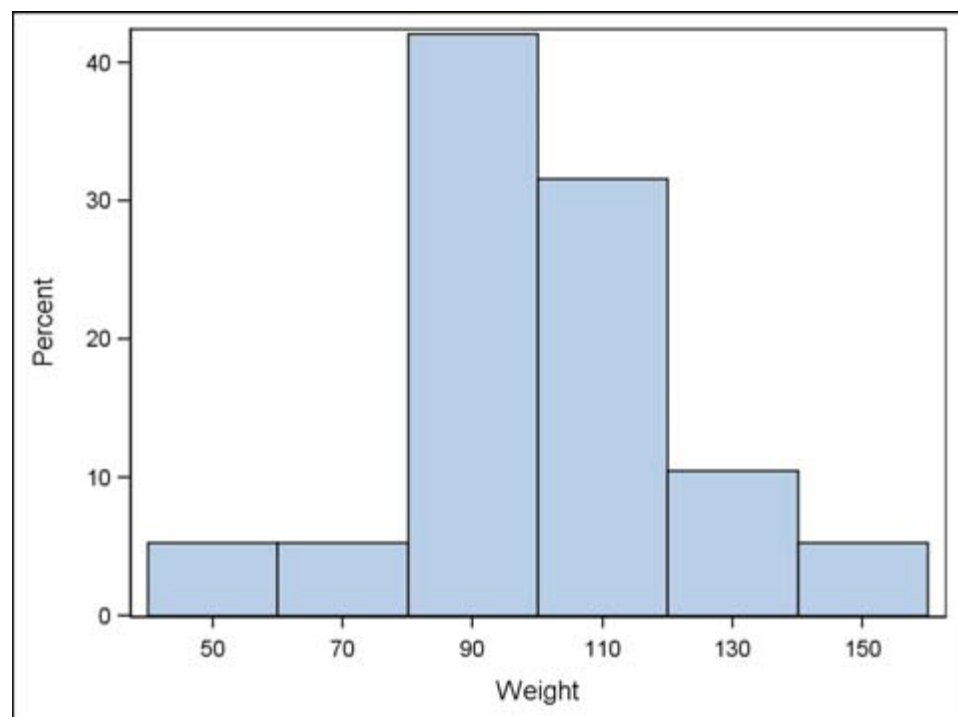
The following SAS program produces the graph:

```
proc template;
  define statgraph histogram;
    begingraph;
      layout overlay;
        histogram weight;
      endlayout;
    endgraph;
  end;
run;
```

```
ods graphics / width=450px;
ods listing;
proc sgrender data=sashelp.class
  template=histogram;
run;
```

- The DEFINE STATGRAPH statement on PROC TEMPLATE opens a definition block for defining a graphics template named HISTOGRAM. The HISTOGRAM template is stored in the template folder (also called the "template store," by default located in SASUSER.TEMPLAT).

- The template definition for HISTOGRAM specifies two GTL statements within a BEGINGRAPH/ENDGRAPH block: LAYOUT OVERLAY and HISTOGRAM.

- The LAYOUT OVERLAY statement is one of the most fundamental layout statements. It can overlay the results of one or more plot statements, each of which shares the same plot area, axes, and legends. The layout in this example specifies only a single element: a HISTOGRAM with bars showing the distribution of observations of the data column named WEIGHT.

- The ENDLAYOUT statement ends the layout block, the ENDGRAPH statement ends the graph definition, and the END statement ends the template definition.

- The ODS GRAPHICS statement uses the WIDTH= option to set a width for the output graph. Because the HEIGHT= option is not specified, GTL manages the graph's aspect ratio and set an appropriate height.

- The ODS LISTING statement opens the LISTING destination for the output. The LISTING destination is open by default. However, explicitly specifying it ensures that it is active for this graph in case another ODS destination has been set in the current SAS session.

- The DATA= option on PROC SGRENDER specifies SASHELP.CLASS as the data source for the graph. TEMPLATE= specifies HISTOGRAM as the template definition to use for rendering the graph.

Template Compilation

A GTL template describes the structure and appearance of a graph to be produced, similar to the way a TABLE template describes the organization and content of a table.

All templates are stored, compiled programs. The following source program produces a simple GTL template named SCATTER:

```
proc template;
  define statgraph scatter;
    begingraph;
      layout overlay;
        scatterplot x=height y=weight;
      endlayout;
    endgraph;
 end;
run;
```

When this code is submitted, the statement keywords and options are parsed, just as with any other procedure. If no syntax error is detected, an output template named SCATTER is created and stored in the default template folder SASUSER.TEMPLAT. No graph is produced. Note the following:

- Any required arguments in the template must be specified. In this example, X= and Y= in the SCATTERPLOT statement must specify variables for the analysis, but no checking for the existence of these variables is done at compile time. (Unlike other SAS procedures, PROC TEMPLATE does not perform a compile and then run sequence, which includes variable validation.)

- No reference to an input data set appears in the template.

Run Time Actions

To produce a graph, a GTL template must be bound to a data source using the SGRENDER procedure. The following example uses SGRENDER to bind the SCATTER template to the SAS data set SASHELP.CLASS, which is delivered with the SAS system:

```
ods listing;
proc sgrender data=sashelp.class
              template=scatter;
run;
```
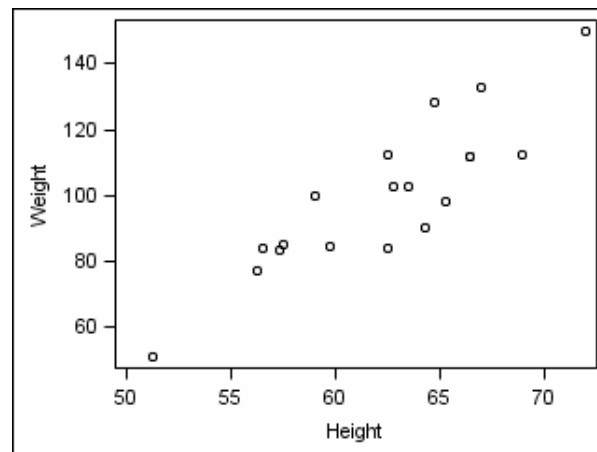

Generally, an ODS data object is constructed by comparing the template references to column names with variables that exist in the current data set. In the current example, SASHELP.CLASS contains variables named HEIGHT and WEIGHT. Because these variable names match the variables that are named on template SCATTER, variables HEIGHT and WEIGHT are added to the data object, while other variables in SASHELP.CLASS are ignored. (It is possible for a template to define new computed columns based on existing columns.)

After all the observations have been read, the data object and template definition are passed to a graph renderer, which produces an image file for the graph. The image file is then automatically integrated into the ODS destination. In this example, a PNG image is created in the LISTING destination. The visual properties of the graph are determined by the ODS style that is in effect.

Note: Template SCATTER is a restrictive definition: it can create a plot only with variables named HEIGHT and WEIGHT. A GTL template can be made more flexible by introducing dynamics or macro variables that supply variables and other information at run time. For more information, see "Flexible Templates" on page 12.

Basic Anatomy of an ODS Graph

The GTL is flexible and able to produce many different types of graphs with varying layout features. The following figure shows the basic anatomy of an ODS graph:
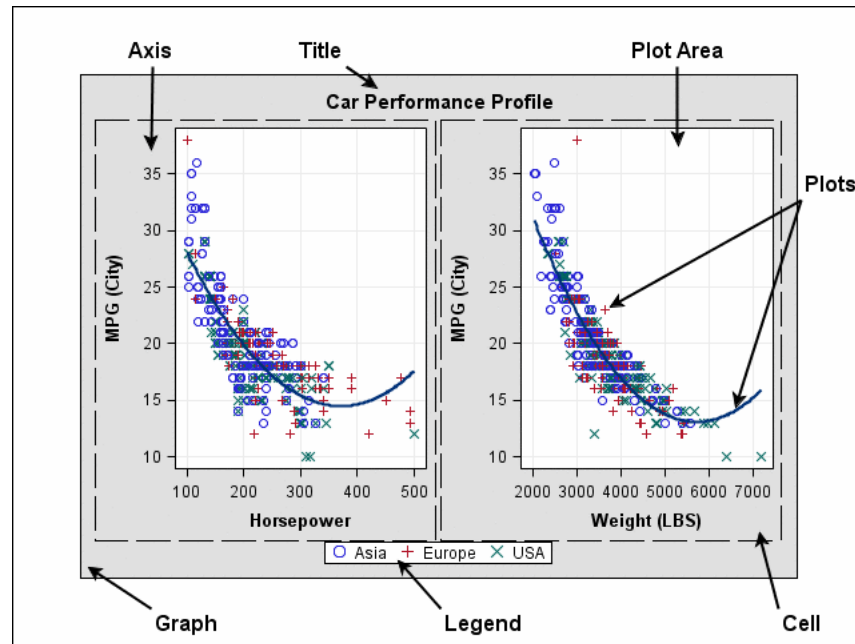
Graph
> the output produced from all of the statements that are nested in a BEGINGRAPH statement block. The graph comprises all of the graphics elements in the template definition.

Title Area
> area for one or more titles. This area is always displayed above all cells in the graph.

Footnote Area
> area for one or more footnotes. This area is always displayed below all cells in the graph.

Cell
> refers collectively to the area containing the plot areas. In this diagram, there are two cells, each of which contains two axes for the plot area. A cell can also contain descriptive text and legends. Graphs are often described as single-cell or multi-cell.

Plot Area
> the display area for plot-statement results. This area is bounded by the axes (when present) and can also contain data labels and other text that annotates the graph.

Axis
> refers collectively to the axis line, the major and minor tick marks, the major tick values, and the axis label.

Plots
> refers collectively to all plot statements that can be overlaid in the plot area. This includes graphical items such as fit lines, scatter plots, reference lines, and many others.

Legend
> refers collectively to one or more legend entries, each made up of a graphical value and a text label. The legend can also have a title and border. Legends can also display a color ramp corresponding to a continuous response range.
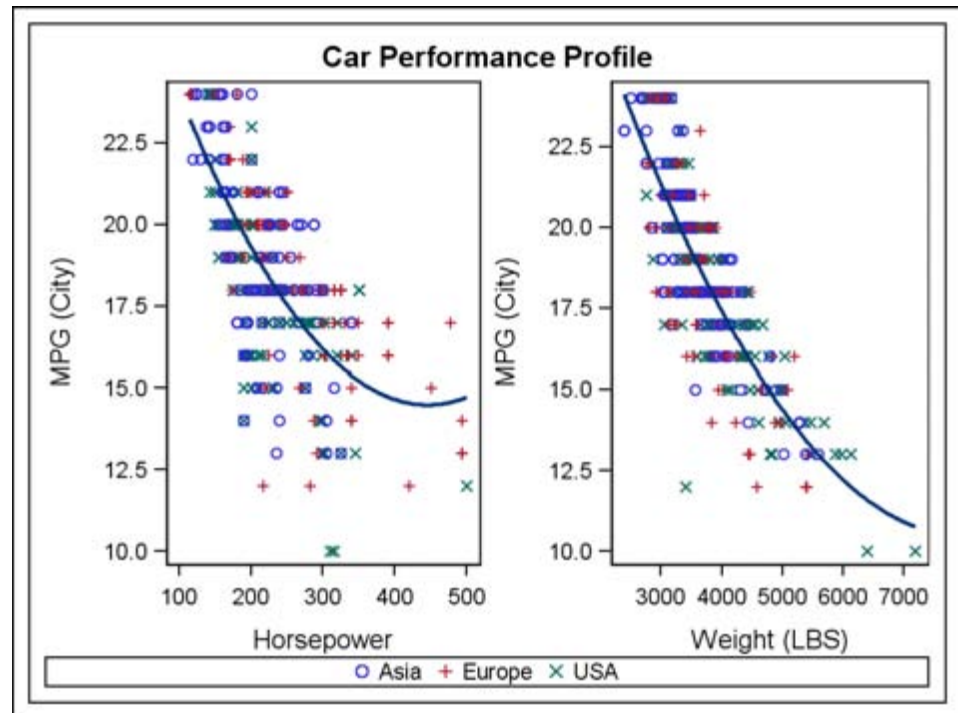
Graphical Layouts

One of most powerful features of the GTL is the syntax built around hierarchical statement blocks called "layouts." The outermost layout block determines

- The overall organization of the graph—whether it uses a single-cell or a multi-cell display.

- What statements are allowed in the block. Generally, layout blocks can contain plots, lines of text, a legend, or even another layout.

- How the contained statements interact.

Table 1.1 *Outermost Layouts in GTL*

Layout	Description
OVERLAY	General purpose layout for displaying 2-D plots in a single-cell.
OVERLAY3D	Layout for displaying 3-D plots in a single-cell.
OVERLAYEQUATED	Specialized OVERLAY with equated axes.
GRIDDED	Basic grid of plots. All cells are independent.
LATTICE	Advanced multi-cell layout. Axes can be shared across columns or rows and be external to grid. Many grid labeling and alignment features.
DATALATTICE	Generates a classification panel from the values of 1 or 2 classifiers.
DATAPANEL	Generates a classification panel from the values of n classifiers.

For example, the following graph is a two-cell graph produced using the LAYOUT LATTICE statement as the outermost template in the layout.

The LAYOUT LATTICE statement is typically used to create a multi-cell layout of plots that are aligned across columns and rows. In the following template, which produced the graph, plot statements are specified within nested LAYOUT OVERLAY statements. Thus, the LATTICE automatically aligns the plot areas and tick display areas in the plots. The LATTICE layout is a good layout to choose when you want to compare the results of related plots.

```
proc template;
  define statgraph lattice;
  begingraph;
    entrytitle "Car Performance Profile";
    layout lattice / border=true pad=10 opaque=true
                     rows=1 columns=2 columngutter=3;
      layout overlay;
        scatterplot x=horsepower y=mpg_city /
          group=origin name="cars";
        regressionPlot x=horsepower y=mpg_city / degree=2;
      endlayout;

      layout overlay;
        scatterplot x=weight y=mpg_city / group=origin;
        regressionPlot x=weight y=mpg_city / degree=2;
      endlayout;

      sidebar;
        discretelegend "cars";
      endsidebar;
    endlayout;
  endgraph;
  end;
run;
```

For detailed information about each layout, see the chapter for that layout type.

Plots

The plots in the GTL are classified in different ways, depending on the context of the discussion.

Within layout blocks, plots are often classified according to graphical dimension: whether they are projected in two or three visual dimensions. Thus, plots in the GTL are often referred to as 2-D or 3-D plots, based on their graphical dimensions, not their data dimensions.

Relative to their input data, plots are classified according to the statements that calculate summary statistics from raw input data, and those that use calculated statistics as input parameters on the plot statement. Thus, many GTL plot statements have two versions: BARCHART and BARCHARTPARM, HISTOGRAM and HISTOGRAMPARM, and so forth. The main distinction between such plots is the nature of the input data that they accept:

- The "non-parm" version (for example, BARCHART) computes its values from raw, unsummarized data. For example, a BARCHART computes the summary values it needs for the bars in the chart. Such plots are often referred to as "computed plots."

- The "parm" version (for example, BARCHARTPARM) does not summarize or compute values from the input data but instead simply renders the input data it is given. Thus, the input data must be pre-summarized, perhaps by a SAS procedure. The "parm" version of plots, often referred to as "parameterized plots," produce the same result as the non-parm version. However, they don't perform the calculations or data summarizations needed to achieve the result.

Chapter 12, "Key Concepts for Using Plots," on page 111 discusses general concepts that apply across plot types. For detailed information about a particular plot, see the chapter for that plot.

Axes

The GTL uses various criteria to determine the displayed axis features for a graph. Generally, axis features are based on the layout type, the order of plot statements in the layout and the options specified on those statements, the use of "primary" and "secondary" axes on the plots (when secondary axes are supported), the plot type, the column(s) of data that contribute to defining the axis range, and the data formats for the contributing data columns.

Depending on the layout type, 2-D plots can have up to four independent axes that can be displayed: X, Y, X2, and Y2. The X and Y axes are considered the primary axes, and the X2 and Y2 axes are considered the secondary axes. By default, the X2 and Y2 axes are not displayed. When requested, the secondary axes can be displayed as copies of the primary axes, or data can be mapped separately to them. The following figure identifies the X, Y, X2, and Y2 axes.
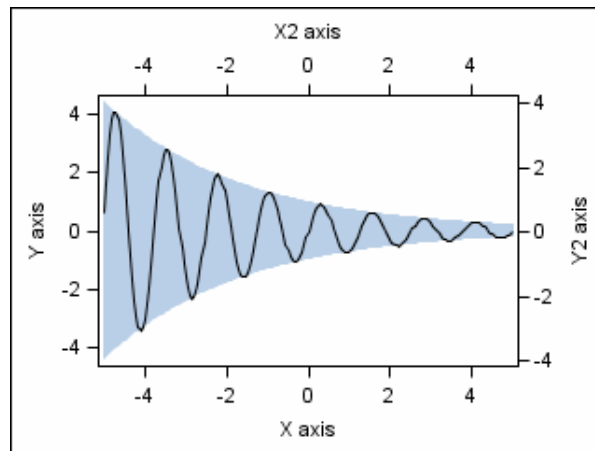
All 3-D plots display the standard X, Y, and Z axes.
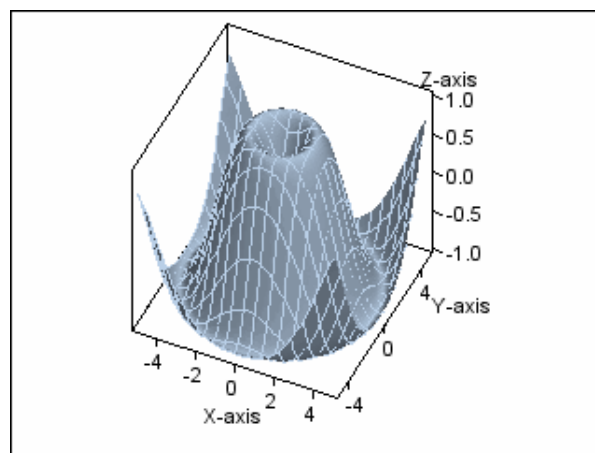
For more information about axis features in GTL, see Chapter 41, "Axis Features in Layouts," on page 391.

Legends

Many plot statements support a GROUP= option that partitions the data into unique values, performs separate analysis, if necessary, and automatically assigns distinct visual properties to each group value. The visual properties of group values are defined by the style in effect.

Legends are not automatically displayed for plots with group values. Rather, an appropriate legend statement must be added to the template to generate the desired legend. In the following example, a legend is added to display markers and line patterns that show the association between the group values from a scatter plot and corresponding linear regression lines. The example shows the mechanism that GTL uses to associate a legend with its corresponding plot(s): a name is assigned to each plot that must be represented in the legend, and these names are then used as arguments for the legend statement (in this case, DISCRETELEGEND).

```
proc template;
 define statgraph scatterfit;
  begingraph;
```

```
        entrytitle "Linear Regression By Gender";
         layout overlay;
            scatterplot x=height y=weight / group=sex name="scat";
            regressionplot x=height y=weight/ group=sex name="reg";
            discretelegend "scat" "reg" / border=true;
         endlayout;
        endgraph;
       end;
      run;
```
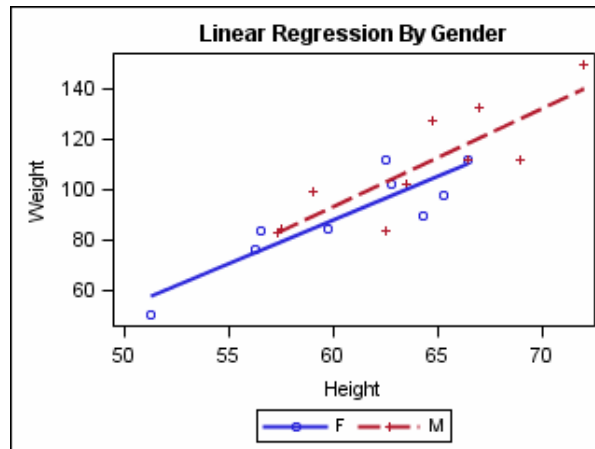

For more information about managing legends in GTL, see *SAS/GRAPH: Graph Template Language User's Guide*.

Flexible Templates

Several features in the GTL can make template definitions less restrictive on input data and more general in nature. These features enable a single compiled template to produce many output variations.

Expressions and Functions

In the GTL, expressions can be used to compute constants and data columns. The expressions must be enclosed in an EVAL construct. Within the expression you can use DATA step functions, arithmetic operators, and other special functions supported by the GTL.

Expressions are also useful in text statements like ENTRY and ENTRYTITLE. Both of these statements support rich text and have special text commands such as {SUP}, {SUB}, and {UNICODE}, which enable subscripting, superscripting, and Unicode characters.

The following template shows how the ± symbol is included in the title line using its hexadecimal Unicode value. Also, new data columns are computed for the upper and lower error bars of the scatter plot, based on the input columns MEANWEIGHT and STDERR.

```
proc template;
 define statgraph expression;
  begingraph;
  entrytitle "Errorbars show " {unicode "00B1"x} "2 SE";
   layout overlay;
     scatterplot x=age y=meanweight /
```

```
            yerrorlower=eval(meanweight - 2*stderr)
            yerrorupper=eval(meanweight + 2*stderr);
        seriesplot x=age y=meanweight;
      endlayout;
    endgraph;
  end;
run;
```

For more information about using expressions, see Chapter 54, "Expressions," on page 541. For more information about using functions, see Chapter 55, "Functions," on page 545.

Dynamics and Macro Variables

An extremely useful technique for generalizing templates is to define dynamics, macro variables, or both. The dynamics and macro variables resolve when the template is executed. The following PROC TEMPLATE statements can be used in a DEFINE STATGRAPH block:

Template Statement	Purpose	Value supplied by...
DYNAMIC	defines dynamic(s)	either of the following: • DYNAMIC= suboption of ODS= option of FILE PRINT • DYNAMIC statement of PROC SGRENDER
MVAR	defines macro variable(s)	%LET or CALL SYMPUT( )
NMVAR	defines macro variable(s) that resolves to a number(s)	%LET or CALL SYMPUT( )
NOTES	provides information about the graph definition	user-supplied text

The following example defines a template named DYNAMICS that can create a histogram and density plot for any variable. It defines both macro variables and dynamics for run time substitution. No data-dependent information is hard coded in the template.

Note: You can initialize macro variables with %LET statements and dynamics with SGRENDER's DYNAMIC statement.

```
proc template;
 define statgraph dynamics;
   mvar SYSDATE9 SCALE;
   nmvar BINS;
   dynamic VAR VARLABEL;
   begingraph;
     entrytitle "Histogram of " VAR;
     entrytitle "with Normal Distribution";
     layout overlay / xaxisopts=(label=VARLABEL);
       histogram VAR /  scale=SCALE  nbins=BINS;
       densityplot VAR / normal();
     endlayout;
```
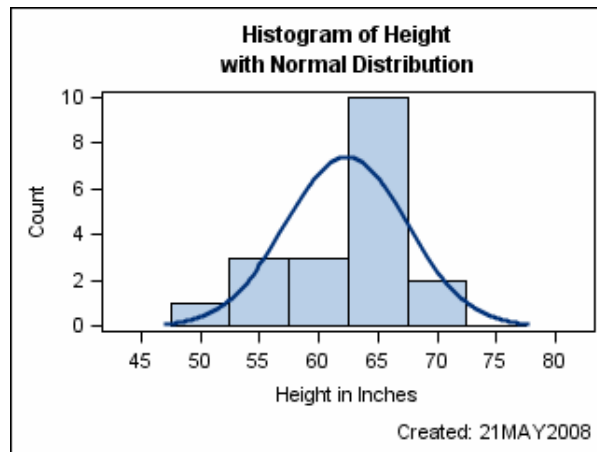
```
          entryfootnote halign=right "Created: " SYSDATE9 /
             textattrs=GraphValueText;
      endgraph;
    end;
  run;

  %let bins=6;
  %let scale=count;
  proc sgrender data=sashelp.class
                  template=dynamics;
    dynamic var="Height" varlabel="Height in Inches";
  run;
```


For more information about using dynamics and macro variables, see Chapter 53,
"Dynamics and Macro Variables," on page 539.

Conditional Logic

Using conditional logic, you can create templates that have multiple visual results or output
representations, depending on existing conditions. The evaluation of a logical expression
must generate one or more complete statements (not portions of statements). All conditional
logic uses one of the following constructs:

<pre>if (condition)
  statement(s);
endif;</pre>
	<pre>if (condition)
	   statement(s);
	else
	   statement(s);
	endif;</pre>

On the IF statement, *condition* must be enclosed in parentheses. The *condition* can be any
standard SAS expression involving arithmetic, logical operators, comparison operators,
Boolean operators, or concatenation operators. The expression can also use SAS DATA
step functions. The expression resolves to a single numeric value, which is *true* or *false*.

In the following example, a histogram is conditionally overlaid with a normal distribution
curve, a Kernel Density Estimate distribution curve, both, or neither:

```
  proc template;
    define statgraph conditional;
      dynamic VAR VARLABEL BINS CURVE;
```

```
    begingraph;
     entrytitle "Histogram of " VAR;
     layout overlay / xaxisopts=(label=VARLABEL);
      histogram VAR /  nbins=BINS;

      if (upcase(CURVE) in ("ALL"  "KERNEL"))
         densityplot VAR / kernel() name="k"
                              legendlabel="Kernel"
                              lineattrs=(pattern=dash);
      endif;

      if (upcase(CURVE) in ("ALL" "NORMAL"))
         densityplot VAR / normal() name="n"
                              legendlabel="Normal";
      endif;

      discretelegend "n" "k";
     endlayout;
    endgraph;
   end;
  run;
```

Note that the legend syntax does not have to be made conditional. At run time, each plot name in the legend is checked. If the plot does not exist, its name is removed from the legend name list. If no names appear in the DISCRETELEGEND statement, the legend "drops out" and the histogram is resized to fill the remaining space.

For more information about using conditional logic, see Chapter 56, "Conditional Logic," on page 549.

Output

When using the GTL, you focus primarily on defining template definitions that produce specific graphs and generate a particular output layout. Ultimately, you must also tailor the graphical environment to get the exact output that you desire. The ODS GRAPHICS statement is available for tailoring the graphical environment, and ODS styles enable you to manage the output appearance.

ODS GRAPHICS Statement

The ODS GRAPHICS statement is used to modify the environment in which graphics templates are executed. The ODS GRAPHICS statement is used to control

- whether ODS graphics is enabled

- the type and name of the image created

- the size of the image

- whether features such as scaling and anti-aliasing are used.

The following ODS GRAPHICS statement uses the HEIGHT= and WIDTH= options to set an aspect ratio for the output image.

```
ods graphics on / height=175px width=200px;
 proc sgrender data=sashelp.class
            template=scatter;
```

```
 run;
ods graphics off;
```

For more information about using the ODS GRAPHICS statement in GTL, see
SAS/GRAPH: Graph Template Language User's Guide. For a more complete discussion
of the ODS GRAPHICS statement, see *SAS Output Delivery System: User's Guide*.

ODS Styles

When any graphics template is executed, there is always an ODS style in effect that governs
the appearance of the output. The following ODS statement sends graphics output to the
RTF output destination using the LISTING style:

```
ods rtf style=listing;

ods graphics on / height=175px width=200px border=off;
 proc sgrender data=sashelp.class
               template=scatter;
   run;
ods graphics off;

ods rtf close;
```

Support for ODS styles is highly integrated into GTL syntax. By default, the graphical
appearance features of most plot and text statements are mapped to corresponding style
elements and associated attributes. Because of this, your output tables and graphs always
have a reasonable overall appearance. Moreover, output for a given ODS destination has
a consistent look (for example, table colors and graph colors don't clash).

The following figures show how a graph's appearance can be changed by using references
to style elements to set the graph's appearance options. This technique permits changes in
graph appearance by style modification instead of graphical template modification. The
graphs in the figures are generated with the following GTL statement:

```
contourplotparm x=x y=y z=density /
  contourtype=fill nhint=9
  colormodel=ThreeColorRamp ;
```

The following style template shows the definition for the ThreeColorRamp style element:

```
style ThreeColorRamp /
  endcolor     = GraphColors("gramp3cend")
  neutralcolor = GraphColors("gramp3cneutral")
  startcolor   = GraphColors("gramp3cstart");
```
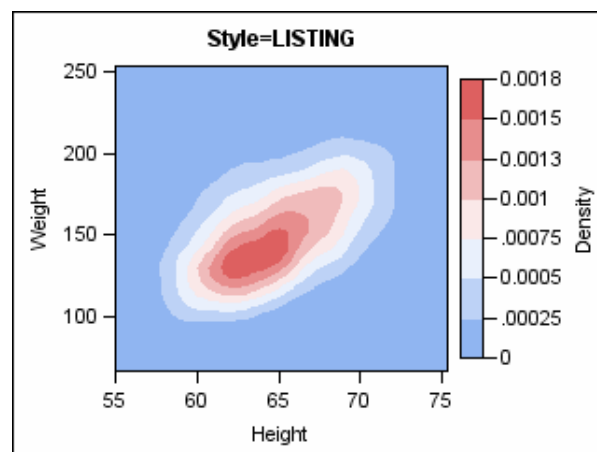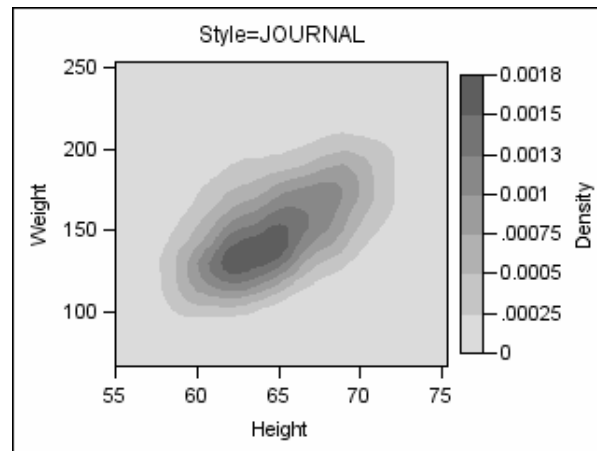
For more information about the use of ODS styles in GTL, see *SAS/GRAPH: Graph Template Language User's Guide*. For a more complete discussion of ODS styles, see *SAS Output Delivery System: User's Guide*. The *SAS/STAT User's Guide* also has a detailed discussion for using styles with ODS Graphics.

About the Examples in this Documentation

The programs in this documentation often provide all of the code that you need to generate the graphs that are shown in the figures. We encourage you to copy and paste the code into your SAS session and generate the graphs for yourself. The graphs that you generate in the LISTING destination are rendered in their default 640 pixel by 480 pixel size. The exception is the examples that show you how to change the graph size.

The graphical output in this documentation does not show graphs in their default size because of the limitations of the production system used. The maximum graph width that can be included in this document is 495 pixels. Hence, all graphs are scaled down to fit.

When graphs that are produced with ODS graphics are reduced in size, several automatic processes take place to optimize the appearance of the output. Among the differences between default size graphs and smaller graphs are that the smaller graphs have scaled down font sizes. Also, their numeric axes might display a reduced number of ticks and tick values. Thus, the graphs that you generate from the example programs will not always look identical to the graphs that are shown in the figures. However, both graphs will accurately represent the data.

When producing your graphical output, you can scale the graph size and also modify font attributes if needed. The *SAS/GRAPH: Graph Template Language User's Guide* provides chapters that explain how to set fonts, DPI, anti-aliasing, and other features that contribute to producing professional-looking graphics of any size in any output format.