Paper Number DV-09

Introduction to the Graph Template Language

Sanjay Matange, SAS Institute, Cary, NC

ABSTRACT

In SAS 9.2, the SAS/GRAPH® Graph Template Language (GTL) goes production. This system is used by many SAS analytical procedures to create the automatic graphical output within the Output Delivery System (ODS). Now, you can access this same system to create your own custom graphs or to customize the graphs created by the SAS analytical procedures. This presentation helps you understand the basics of GTL, and how you can leverage its features to customize your graphs.

INTRODUCTION

The ODS Graphics system is an extension to ODS and is available with SAS 9.2. With this system, you can get modern analytical graphics produced automatically by many SAS procedures using the statement ODS GRAPHICS ON;. As part of the ODS Graphics system, SAS/GRAPH includes GTL syntax for defining sophisticated analytical graphs using the TEMPLATE procedure. A new family of statistical graphics procedures is included with SAS/GRAPH to create commonly used analytical graphs. Also included is the ODS Graphics Editor, an interactive tool for editing and customizing ODS graphics output.

OVERVIEW OF THE ODS GRAPHICS SYSTEM

With the ODS Graphics system, getting the statistical graphs you need is now as easy as 1 2 3. The system can be used at different levels, depending on the needs of the user.

1. Users wanting the standard graphs expected from the procedures simply need to turn on the ODS Graphics system to automatically get these graphs. This is done by submitting the statement ODS GRAPHICS ON;. Using GTL, SAS procedure writers have created the templates needed for the graphs for each procedure. Users can use different ODS styles to get the visual appearance they want for their output. For more information on this topic, see the paper “Getting Started with ODS Statistical Graphics in SAS 9.2.”

2. Users can customize the ODS graphics output from the procedures by using the ODS Graphics Editor, an interactive, GUI-based tool without any knowledge of GTL. The ODS Graphics Editor provides an intuitive GUI to change the style, size, or DPI of the graph. You can edit, add, or delete titles and footnotes, reposition the legends, and customize the visual properties of the plots and axes. Furthermore, you can add free-form annotation to the graph to emphasize the relevant aspects of the results. You can copy the graph to the system clipboard to include it in other documents, or you can save the graph for future use. For more information on this topic, see the paper “ODS Graphics Editor.”

3. Users creating custom graphs to complement the ODS graphs produced by the procedures can use the new SAS/GRAPH Statistical Graphics (SG) procedures. These procedures leverage the power of GTL behind the scenes, and support a simple and concise syntax for creating commonly used graphs for analytical tasks. For more information on this topic, see the paper “Effective Graphics Made Simple Using SAS/GRAPH SG Procedures.” These commonly used graphs are:

    a. the SGPLOT procedure for creating single-cell composite graphs
    b. the SGPANEL procedure for creating paneled graphs driven by multiple class variables
    c. the SGSCATTER procedure for creating gridded scatter plots and scatter plot matrices

4. Users building custom graphs that are beyond the capability of the SG procedures can use the GTL syntax directly. Users who want to make persistent changes to the graphs produced by the analytical procedures (mentioned in 1) can edit the templates for these graphs using GTL. Once the original template is modified, subsequent execution of the procedure will use the modified template.

So, users that constitute this group, or users that are curious what GTL is all about, this presentation is for you!

GETTING STARTED

With SAS 9.2, the TEMPLATE procedure has been extended to define a new type of template called statgraph. The statgraph template is defined using GTL syntax. Creating a graph using GTL is a two-step process:

1. First, a statgraph template is created using the TEMPLATE procedure and GTL syntax.
2. Second, this template is associated with a data set using the SGRENDER procedure.

Figure 1b shows a program that creates a graph using the TEMPLATE and SGRENDER procedures. Part A of the code defines the **`statgraph`** template. When submitted, the syntax is compiled into a statgraph template called **`graph.density`**. The template defines only the structure of the graph. To produce the graph, the template must be associated with the appropriate data set as shown in part B. Now, the system has the information needed to create the graph shown in figure 1a.

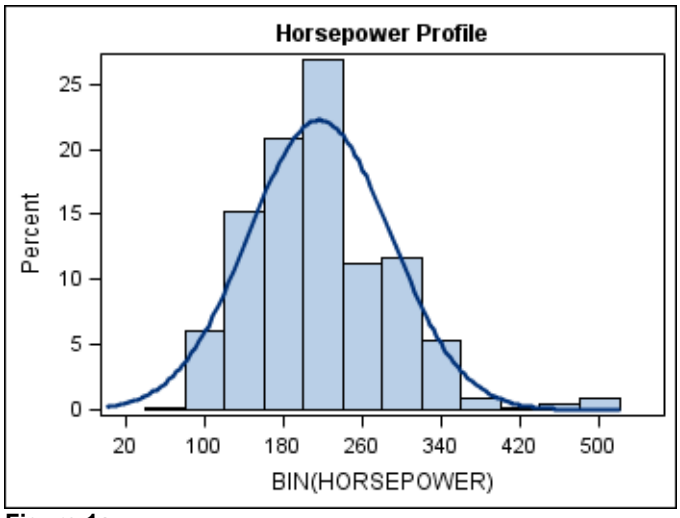

Figure 1a

```
A   proc template;
      define statgraph graph.density;
        begingraph;
          entrytitle "Horsepower Profile";
          layout overlay;
            histogram horsepower;
            densityplot horsepower;
          endlayout;
        endgraph;
      end;
    run;

B   ods graphics on / width=3.5n;
    proc sgrender data=sashelp.cars
            template=graph.density;
     run;
```

Figure 1b

Look at the parts of this simple GTL template, as shown in figure 2.

- The template definition starts with the **`proc template`** statement.
- The **`define`** statement (C) specifies the type as **`statgraph`** and provides the name of the template.
- The template definition is provided inside the **`begingraph/endgraph`** block indicated by the outer brace (E). All GTL statements have to be inside this block.
- The body of this template contains an **`entrytitle`** statement (D) that defines the title of the graph, and a **`layout overlay`** block (F) that contains a **`histogram`** and a **`densityplot`**.

```
C   proc template;
      define statgraph graph.density;
        begingraph;

D     entrytitle "Horsepower Profile";

E     layout overlay;
F       histogram horsepower;
        densityplot horsepower;
      endlayout;

        endgraph;
      end;
    run;
```

Figure 2

The template in figure 2 defines a graph with a title, and one layered composite plot built using a histogram and a density plot of the variable HORSEPOWER. PROC SGRENDER is run as shown in part B of figure 1b. The name of the template is provided, and a data set that has the variable HORSEPOWER is used to produce the graph. This template will not work if the data set does not contain a variable named HORSEPOWER. Later in this presentation, you will learn how to design templates that have the flexibility to work with different variables.

LANGUAGE COMPONENTS

GTL supports a structured syntax that provides a building-block approach to designing your graphs. The syntax elements of GTL fall into four main categories. These are:

1. plot statements. such as HISTOGRAM or DENSITYPLOT in the template above
2. layout statements, such as LAYOUT OVERLAY in the template above
3. accessory statements for titles, footnotes, legends, and text entries, such as ENTRYTITLE in the template above
4. conditionals, expressions, functions, and more

1. Plot Statements

GTL provides a rich set of plot statements for designing many types of analytical graphs. Generally, these fall into these categories:

- basic plots, such as SCATTERPLOT, SERIESPLOT, STEPPLOT, BANDPLOT, NEEDLEPLOT, and BARCHART
- fits and confidence plots, such as LOESSPLOT, REGRESSIONPLOT, PBSPLINEPLOT, and ELLIPSEPLOT
- distribution plots, such as HISTOGRAM, BOXPLOT, DENSITYPLOT (normal and Kernel)
- parameterized plots, such as BARCHARTPARM, ELLIPSEPARM, LINEPARM, and REFERENCELINE

2. Layout Statements

GTL provides a set of layout statements to create sophisticated composite, paneled, or comparative graphs. Some layout statements can be nested. The layouts are:

- LAYOUT OVERLAY: the most basic layout container used to create layered composite plots
- LAYOUT LATTICE: used to create gridded comparative graphs with uniform data ranges
- LAYOUT DATAPANEL and DATALATTICE: used to create gridded panels by classification variables
- LAYOUT PROTOTYPE: used to create a rubber-stamp for repeated usage in a DATAPANEL
- LAYOUT GRIDDED: used to create grids of data statistics or plots
- LAYOUT OVERLAYEQUATED: used for special plots where the X and Y axis have the same scale
- LAYOUT OVERLAY3D: used for special 3D plots only

3. Accessory Statements

GTL provides a rich variety of accessory statements for placing titles, footnotes, legends, and text insets to fully describe the data in the graph. These are:

- Textual elements, such as ENTRYTITLE and ENTRYFOOTNOTE. These statements are global to the graph.
- Entries and legends that can be in the layered composite or in a cell of a gridded layout.

4. Conditionals, Expressions, Functions, and More

GTL provides additional syntax for the following:

- conditionals: IF, ENDIF, and ELSEIF statements
- expressions can be used, such as Y=EVAL(EXPRESSION);
- functions: SAS, DATA step, and ODS graphics functions can be evaluated using EVAL ( )

These are the building blocks for defining your graphs. They are like the ingredients for a recipe, and you can mix and match them creatively to build the graphs you need.

SINGLE-CELL GRAPH

The single-cell composite graph is the most basic and frequently used plot type. It is important to understand how to build this plot type using multiple plot types within a layout overlay. The graph in figure 1a was created in this way using the program in figure 1b. It is also important to understand the behavior of the plots within a layout overlay, and to get a feel for the various options available on the plot statements and the layout statements. This plot type can be easily built using the SGPLOT procedure. However, this information will be useful when more complex plots are built using the other layouts.

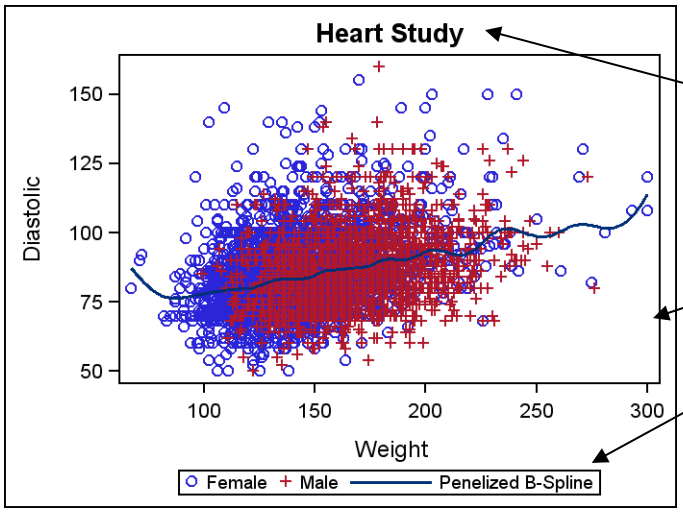

Figure 3a

```
proc template;
  define statgraph graph.fit1;
    begingraph;
      entrytitle "Heart Study";
      layout overlay;
        scatterplot x=weight y=diastolic /
              group=sex name="scatter";
        pbsplineplot  x=weight y=diastolic /
             name="fit"
             legendlabel="Penelized B-Spline";
        discretelegend "scatter" "fit";
      endlayout;
    endgraph;
  end;
run;
```

A

B

Figure 3b

Figure 3b shows the template for the fit plot that includes a title and one layered composite plot. First, a title for the plot is specified using the **`entrytitle`** statement (A). Then, a layered composite plot is created by placing the following plots in a **`layout overlay`** container (B):

- a scatter plot with **`x=weight`** and **`y=diastolic`**
- the optional argument **`group=sex`** is set
- a Penalized B-Spline with **`x=weight`** and **`y=diastolic`**
- a legend with a list of names to be included

The SGRENDER procedure is used to run the template with the sashelp.heart data set. The results are shown in figure 3a. The legend displays items from both the scatter plot and the fit plot. To do this, the code has to tell the legend what items to display. This is done by first providing a name for each of the plots you want to include in the legend by setting the optional argument NAME= for both SCATTERPLOT and PBSPLINEPLOT. Then, the list of names is provided to the DISCRETELEGEND statement.

By default, the legend is drawn outside the plot area, at the bottom, as shown in figure 3a. The system reserves space for it. Options are available to place the legend in one of the eight compass positions. The user can choose to place the legend inside the plot area using optional arguments, which is shown later.

Adjusting Common Attributes

While the various plot types have special features and required and optional arguments, all plots have a certain set of common arguments. The scatter plot can be used to show many of these. In the template syntax in figure 4b, the following is adjusted:

- set the transparency of the scatter marker by setting **`datatransparency=0.9`** (A)
- reduce the marker size by setting **`size=5`** in the **`markerattrs`** bundle (A)
- set the **`group=`** option for the **`pbsplineplot`** to get fit by **`sex`** (B)
- move the legend inside the plot area by setting the option **`location=inside`** (C)
- position the legend in the upper-right corner by setting the **`halign=`** and **`valign=`** options (C)
- change the legend to show the fit lines only in one column by setting **`across=1`** (C)

The results of these adjustments are shown in figure 4a. Setting the transparency for the markers allows us to better visualize areas of high observation counts.

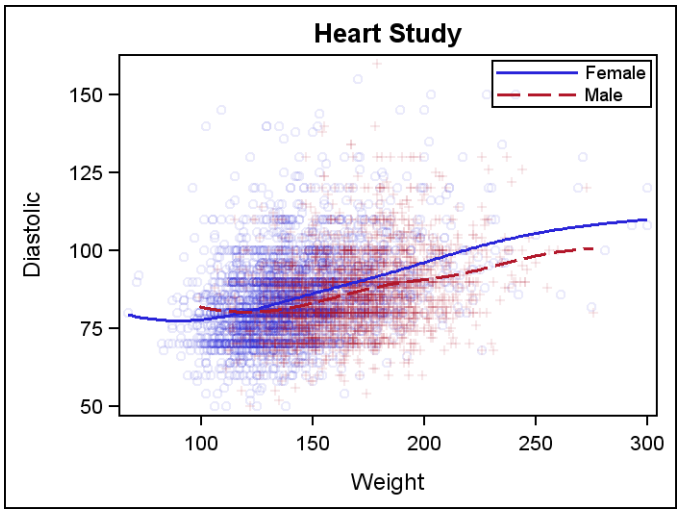

Figure 4a

```
begingraph;
  entrytitle "Heart Study";
  layout overlay;
A   scatterplot x=weight y=diastolic /
                group=sex name="scatter"
                datatransparency=0.9
                markerattrs=(size=5);
B   pbsplineplot  x=weight y=diastolic /
                group=sex clm="pbs"
                alpha=0.05 name="fit";
C   discretelegend "fit" / location=inside
                halign=right valign=top
                across=1;
  endlayout;
endgraph;
```

Figure 4b

Finally, a band plot was added to the graph to show the confidence limits of the mean BY group. The band plot needs to be associated with the clm data computed by the PB_SPLINEPLOT statement. This is done as follows:

- Provide a name for the **`clm="pbs"`** option of the **`pbsplineplot`** statement. This is the model. (D)
- Add a **`modelband`** statement to the composite and associate it with the model **`"pbs"`**. (E)
- Add a text **`entry`** to show the number of observations. (F)
- Add footnote using a Unicode character to display the value of alpha. (G)

The plots inside of the layout overlay container are drawn in the order they are specified. It is important to order the plot statements correctly, especially when plots that use fill areas are included. It is best to draw these plots first, so that they will not hide other plots in the composite. In the case shown in figure 5b, the bands have been assigned some transparency, so even if they were placed later in the order, they would not fully occlude the plots under them.

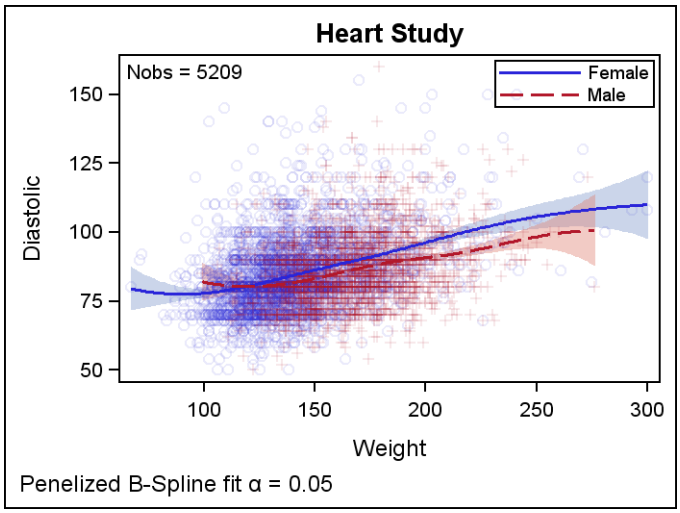

Figure 5a

```
  begingraph;
    entrytitle "Heart Study";
    layout overlay;
E     modelband "pbs" / datatransparency=0.6;
D     scatterplot x=weight y=diastolic /
                  group=sex  datatransparency=0.9;
      pbsplineplot  x=weight y=diastolic / name="fit"
                  group=sex clm="pbs" alpha=0.05;
      discretelegend "fit" / location=inside
                  halign=right valign=top across=1;
F     entry halign=left "Nobs = 5209" / valign=top;
    endlayout;
G   entryfootnote halign=left "Penelized B-Spline fit
    "
                  {UNICODE ALPHA} " = 0.05";
  endgraph;
```

Figure 5b

General Concepts

GTL uses a layout- and element-centric architecture, instead of a chart-centric one. Each plot element is responsible only for rendering its own graphical elements within the region assigned to it by the layout manager. All plots drawn in a layout overlay share a common plot area and a common set of axes. Different plots placed in one layout overlay container can have different variables assigned to the X and Y roles. It is the responsibility of the container to collect all of the data for each axis from each plot, and then draw the axis appropriately. The container reports back to each plot on how to render its data in the merged data space of the axis.

Figure 6b shows a template definition that uses three variables: Europe, Asia, and Actual. The variable Europe has three countries, and Asia has two. In the graph template, the first bar chart plots Actual by Europe, and the second bar chart plots Actual by Asia.

- **`barchart x=europe y=actual`** : variable Europe has values France, Germany, and Spain (A)
- **`barchart x=asia y=actual`** : variable Asia has values Japan and China (B)
- set X and Y axis labels and **`cycleattrs=true`**

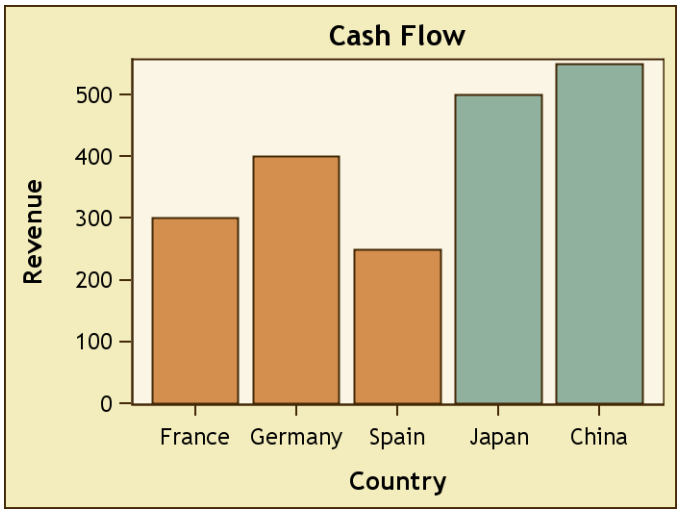

Figure 6a

Obs	Europe	Asia	Actual
1	France		300
2	Germany		400
3	Spain		250
4		Japan	500
5		China	550

```
begingraph;
  entrytitle "Cash Flow";
C layout overlay / xaxisopts=(label="Country")
                   yaxisopts(label="Revenue")
A                  cycleattrs=true;
B   barchart x=europe y=actual;
    barchart x=asia y=actual;
  endlayout;
endgraph;
```

Figure 6b

The category X axis of the bar chart collects all of the values from both bar charts, and creates a union of all the values to be included on the X axis. Then, each bar chart is instructed where to draw its observations. Similarly, the interval Y axis creates a union of the range of the data to be represented on it, and draws the axis accordingly. Each bar chart is informed of the mapping it needs to use to draw the bars correctly.

The axes and the common plot area are controlled by the layout overlay container, and the options to customize them are made available on the LAYOUT OVERLAY statement. In this case, the X axis label is set to **`"Country"`** and the Y axis label to **`"Revenue"`** using the XAXISOPTS and YAXISOPTS option bundles on the LAYOUT OVERLAY statement.

Primary Plot

In the example in figures 6a and 6b, both of the plot statements in the layout overlay container are bar charts, where the X axis is always categorical. In this case, the axis simply creates a union of the set of the axis values from each statement. However, because the two plots have different variables assigned to X, the system has to decide which label should be used. There can be other cases where the axis roles are a mixture of different data types, such as categorical, interval, date, and so on. To resolve this difference, the system uses the concept of a "Primary Plot," and the attributes of the primary plot are used where necessary. The primary plot is determined as follows:

- the plot for which the option PRIMARY=TRUE
- if multiple plots as set as primary, the last plot is considered primary
- some plot types cannot be set as primary, such as lineparm and referenceline
- if no plots are set as primary, the first eligible plot is considered primary

In the template in figure 6b, none of the bar charts are set as primary, so the first bar chart is considered as the primary plot. The axis label for the X axis is derived from the first bar chart and is **"Europe"**. For this case, it makes sense to change the axis label to **"Country"** as shown in the template syntax.

Data Types
The various plot types support different data types, such as interval, discrete, and date. Plots having different types of data can be mixed in a layout overlay where allowed, and the primary plot decides what the axis looks like. If the axis type is discrete (because the primary plot is a bar chart), then you can add plots with interval data to the layout. In this case, the data is considered discrete. However, if the axis type is interval, then plots that use discrete data cannot be added to the layout. In this case, these plots are rejected.

Axes Sets
The layout overlay supports two sets of axes as shown in figure 7. These are:

- the X and Y axes at the bottom and left of the plot
- the X2 and Y2 axes at the top and right of the plot

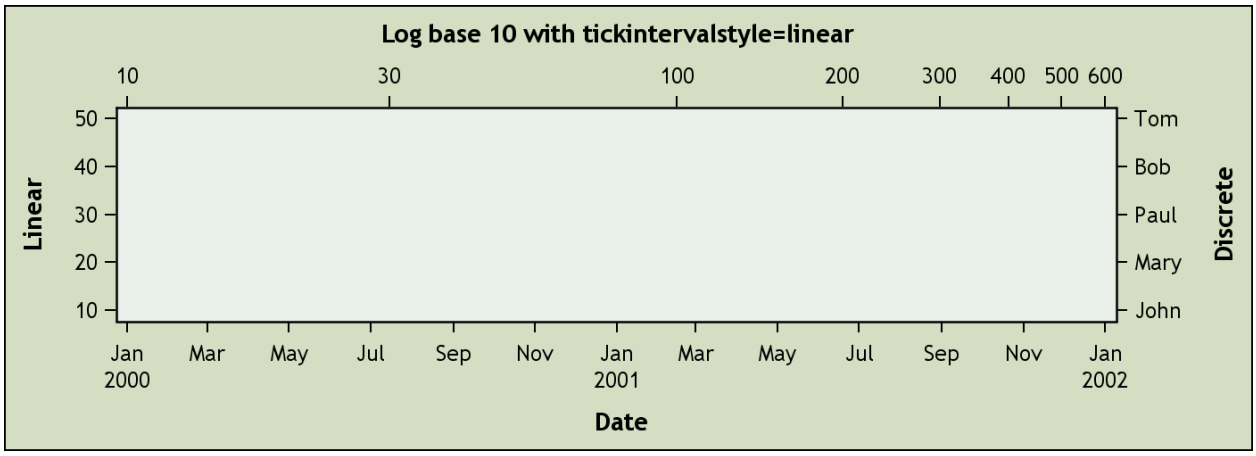

Figure 7

By default, the data for any plot in the layout overlay is plotted on the X and Y axes. However, any plot can request that its data be plotted to X2 or Y2 axis by setting the XAXIS=X2 or YAXIS=Y2 options. The display settings for the various axes can be controlled in the LAYOUT OVERLAY statement with the XAXISOPTS, YAXISOPTS, X2AXISOPTS, and Y2AXISOPTS options.

Axes Types
The system supports many types of axes for the different data types and use cases as shown in figure 7.

- Linear: This is the default for interval data as shown on the left Y axis.
- Date/Time: This is used by default for a date or time format as shown for the lower X axis.
- Discrete: This is used for character or numeric discrete data as shown on the right Y2 axis.
- Log: This us used for linear data with axis TYPE=LOG as shown on the upper X2 axis.

MULTI-CELL GRAPHS

So far, single-cell plot composites using the layout overlay have been discussed, which is the most commonly used layout in GTL. All plots in a layout overlay are drawn in a common plot region in the order they are specified. How to build a composite plot using multiple plot, entry, and layout statements has been explained.

Now we move on to discuss layouts that partition the graph region into smaller cells to build a paneled graph. In this section, the following layouts are described:

- LAYOUT LATTICE: creates a regular grid of *MxN* independent cells with uniform scaling
- LAYOUT DATALATTICE: creates a regular grid of cells based on row and column class variables
- LAYOUT DATAPANEL: creates a regular grid of cells based on multiple class variables
- LAYOUT GRIDDED: creates a regular grid of *MxN* independent cells for tables
- LAYOUT PROTOTYPE: a special layout used by LAYOUT DATAPANEL and DATALATTICE to define the cell

Each cell of the grid can be populated by one of the layered composite plots previously created using layout overlay.

LAYOUT LATTICE

This is one of the most flexible layout types in GTL. It is heavily used in many graphs created by analytical procedures. A common use case for this graph is shown in figure 8a and the template is shown in figure 8b.

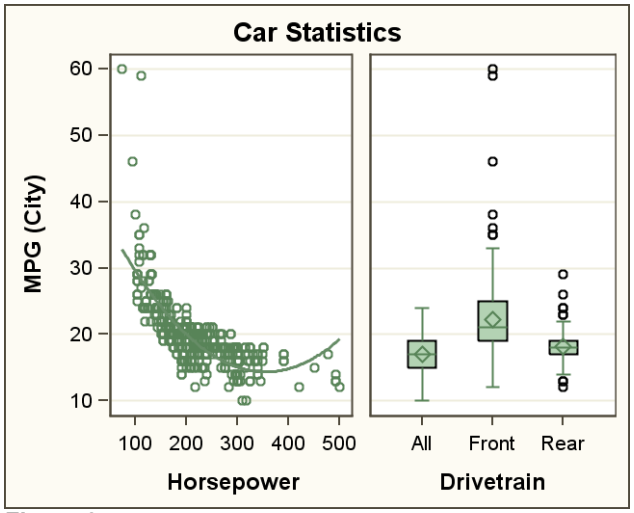

Figure 8a

```
begingraph;
  entrytitle halign=center 'Car Performance Profile';
  layout lattice / columngutter=10 columns=2
                   rowdatarange=union;
    layout overlay;
      scatterplot x=horsepower y=mpg_city;
      regressionplot x=horsepower y=mpg_city /
                     degree=2;
    endlayout;
    layout overlay / xaxisopts=(label="Drivetrain");
      boxplot x=drivetrain y=mpg_city;
    endlayout;
    rowaxes;
      rowaxis / griddisplay=ON label='MPG (City)';
    endrowaxes;
  endlayout;
endgraph;
```

Figure 8b

The various elements of the template that create this plot are:

- a composite plot of a **`scatterplot`** and a **`regressionplot`** in a **`layout overlay`** (A)
- a **`boxplot`** inside a **`layout overlay`** where the X axis label is customized (B)
- place plots (A) and (B) in a **`layout lattice`**, with **`columns=2`** (C)
- set external axes for the row by defining **`rowaxis`** inside the **`rowaxes`** block (D)
- the **`layout lattice`** specifies a uniform data range for the row and sets the gutter (E)

Other features of this layout include:

- Each cell can have a cell header and its own set of X, Y, X2, and Y2 axes.
- Sidebar regions and row and column headers are supported.
- Each cell contains individual plots fully defined by the user.

Figure 9a shows a stock plot that has two plots (A) (B), stacked in one column in a layout lattice container (C), with a common X axis (D). Each cell contains a layered composite plot built using multiple plot and legend statements in a layout overlay, as shown in the template in figure 9b

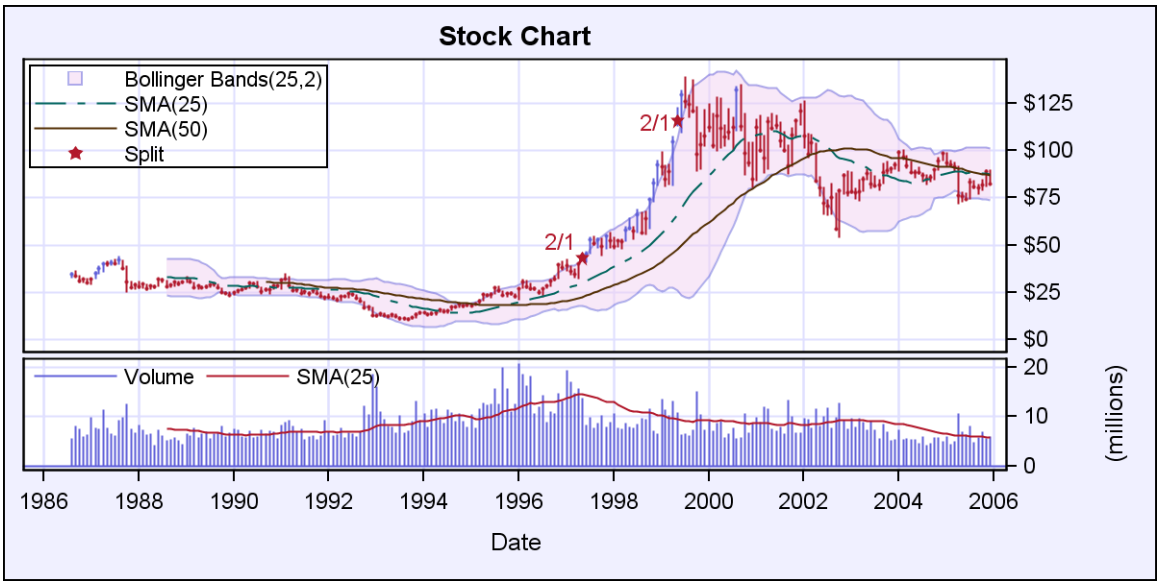

Figure 9a

```
begingraph;
  entrytitle "Stock Chart";
  layout lattice / columns=1 columndatarange=union rowweights=(0.7 0.3);
    columnaxes;
      columnaxis / offsetmin=0.02 griddisplay=on;
    endcolumnaxes;                                                          D
    layout overlay /  yaxisopts=(griddisplay=on label=" " display=(line) displaysecondary=all)
                      cycleattrs=true;
      bandplot x=date limitupper=bolupper limitlower=bollower / datatransparency=0.5
                      name="boll" outlineattrs=(pattern=solid) fillattrs=graphconfidence
                      display=(fill outline) legendlabel="Bollinger Bands(25,2)";
      vectorplot xorigin=date y=low yorigin=high x=date / arrowheads=false group=cindex
                      index=cindex lineattrs=(pattern=solid thickness=1px) shaftprotected=true;
      scatterplot x=date y=close / group=cindex index=cindex markerattrs=(symbol=plus size=2);
      scatterplot x=date y=close / freq=freq markerattrs=(symbol=starfilled size=7)
                      datalabel=split name="split" legendlabel="Split";
      seriesplot x=date y=avg25 / lineattrs=(thickness=1px) legendlabel="SMA(25)" name="d25";
      seriesplot x=date y=avg50 / lineattrs=(pattern=solid thickness=1px) legendlabel="SMA(50)"
                      name="d50";
      discretelegend "boll" "d25" "d50" "split" / across=1 border=on valign=top halign=left location=inside;
    endlayout;                                                              A
    layout overlay / cycleattrs=true xaxisopts=(griddisplay=on)
                      yaxisopts=(griddisplay=on display=(line) displaysecondary=all);
      needleplot x=date y=volume /  name="vol" legendlabel="Volume"
                      lineattrs=(pattern=solid thickness=1px);
      seriesplot  x=date y=vavg / name="vavg" legendlabel="SMA(25)"
                      lineattrs=(pattern=solid thickness=1px);
      discretelegend "vol" "vavg" / location=inside border=off halign=left valign=top;
    endlayout;                                                              B
  endlayout;                                                                C
endgraph;
```

Figure 9b

LAYOUT DATALATTICE and LAYOUT DATAPANEL

Graphs that have gridded panels of multiple cells driven by a set of class variables are a powerful part of information delivery tools. The data can be categorized by class variables such as REGION, PRODUCT, DEPARTMENT, and so on.

LAYOUT DATALATTICE

The LAYOUT DATALATTICE creates classification panels driven by one row and one column variable. The variable is treated as discrete. Figure 11ab shows a common example of a data lattice built using the template in figure 10a. The classification data lattice can have both row and column variables, or just a column variable or a row variable.

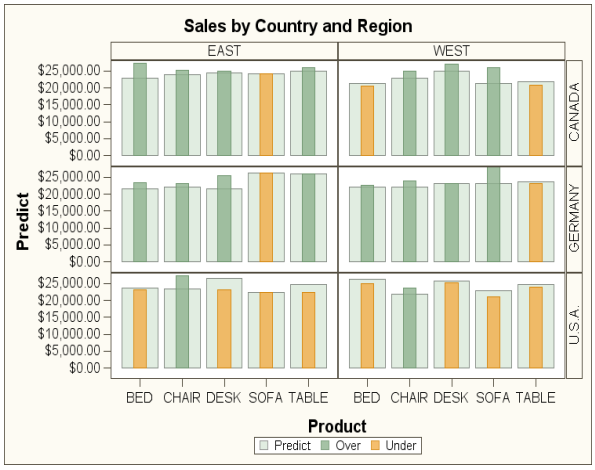

Figure 10a

```
begingraph;
  entryTitle "Sales by Country and Region";
  layout datalattice rowvar=country columnvar=region /
       headerlabeldisplay=value;
    layout prototype;
      barchart x=product y=predictsum / stat=sum name="a"
            datatransparency=0.6 legendlabel="Predict";
      barchart x=product y=actualsum / stat=sum name="b"
          group=status barwidth=0.3 datatransparency=0.2;
    endlayout;
    sidebar;
      discretelegend "a" "b" / across=3;
    endsidebar;
  endlayout;
endgraph;
```

A, B, C

Figure 10b

The elements of syntax to build this classification data lattice are:

- the **`layout prototype`** defines the layered composite plot to be populated in each cell (A)
- define the **`layout datalattice`**, specifying the column and row class variables (B)
- the legend is placed in the **`sidebar`**, which is at the bottom by default (C)

LAYOUT DATAPANEL

The LAYOUT DATAPANEL creates panels driven by multiple class variables. The variables are treated as discrete. Figure 11a shows an example of a data panel built using the template in figure 11b.

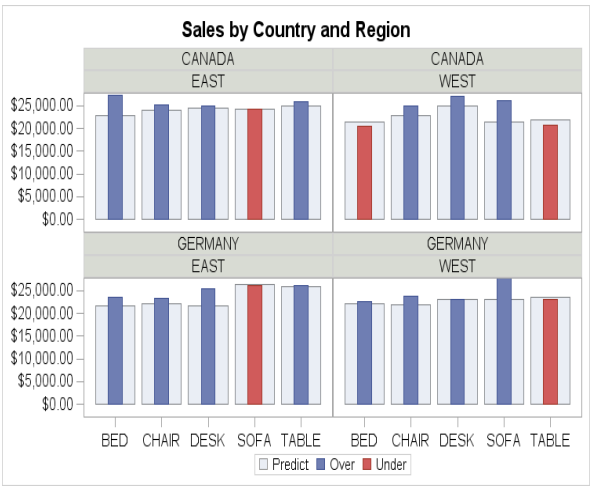

Figure 11b

```
begingraph;
  entryTitle "Sales by Country and Region";
  layout datapanel classvars=(country region) /
      columns=2  headerlabeldisplay=value
        rowaxisopts=(display=(line ticks tickvalues))
        columnaxisopts=(display=(line ticks tickvalues));
    layout prototype;
      barchart x=product y=predictsum / stat=sum
          datatransparency=0.6 name="a";
      barchart x=product y=actualsum /
          group=status stat=sum barwidth=0.3 name="b";
    endlayout;
    sidebar;
      discretelegend "a" "b" / across=3;
    endsidebar;
  endlayout;     endgraph;
```

A, B, C

Figure 11a

The data panel draws a cell for each crossing of the class variables that have data. The values of the class variables are placed in a cell header in each cell. Construction of the syntax for the **`layout prototype`** block (A), **`sidebar`** (C), and **`layout datapanel`** block (B) is similar to LAYOUT DATALATTICE.

LAYOUT GRIDDED

LAYOUT GRIDDED is a general-purpose, regular grid layout manager for grouping components. Its primary usage is dividing the graph region for other nested layouts. This layout is also used for creating statistic tables for embedding in layered composites. It can be used to arrange multiple legends in a sidebar.

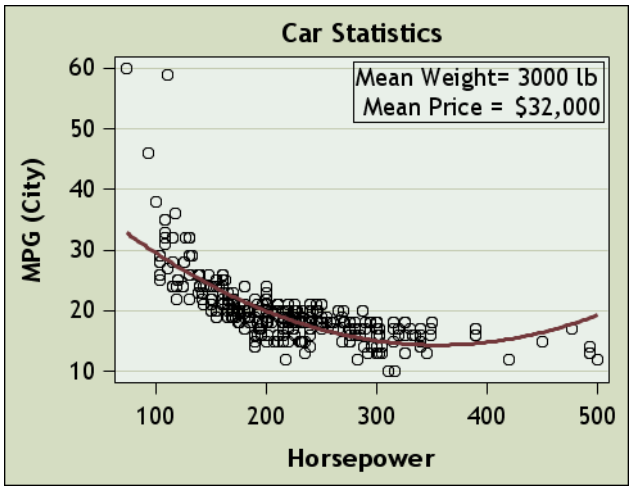

Figure 12a

```
begingraph;
  entryTitle "Car Statistics";
  layout overlay / yaxisopts=(griddisplay=on);
    scatterplot x=horsepower y=mpg_city;
    regressionplot x=horsepower y=mpg_city / degree=2;

    layout gridded / columns=2 border=true
                     halign=right  valign=top;
      entry "Mean Weight="; entry "3000 lb";
      entry "Mean Price ="; entry "$32,000";
    endlayout;

  endlayout;
endgraph;
```

A — layout overlay … endlayout;
B — layout gridded … endlayout;

Figure 12b

The graph in figure 13a is built using the template in figure 13b. The key aspects are:

- this layered graph has a **`scatterplot`** and a **`regressionplot`** inside a **`layout overlay`** (A)
- a table of statistics is built using four **`entry`** statements within a **`layout gridded`** (B)
- two columns and a border are specified, and the grid is placed in the upper-right corner (B)

In all of the examples shown, values for the required and optional arguments were hardcoded to specific variable names. Also, options such as COLUMNS= were hardcoded. While this is convenient for a specific graph, this definition of the template can be too rigid. The templates would be more useful if the values for the arguments could be changed at run time.

Dynamics, Mvar, and Nmvar

The system supports dynamic variables and macro variables. These variables can be defined in the template, and can be set when PROC SGRENDER is executed.

In the example in figure 13, two dynamics are defined: **`_var1`** and **`_var2`**. In the **`scatterplot`** statement, instead of using hard coded variable names such as height and weight, you can use these dynamic variables. Also, you can set other options to macro or numeric macro variables, which can be set at run time. Dynamics can be set using the DYNAMIC= argument on the SGRENDER procedure.

```
begingraph;
  dynamic _var1 _var2;
  mvar _scale;
  nmvar _size;
  layout overlay;
    scatterplot x=_var1 y=_var2 /
                markerattrs=(size=_size);
  endlayout;
endgraph;

proc sgrender data=sashelp.class template=…
  dynamic _var1="height" _var2="weight";
run;
```

Figure

Conditionals and Function Evaluation

The system supports conditional logic in the template syntax, such as IF, ELSEIF, and ENDIF. GTL supports function evaluation using the EVAL () syntax. Many GTL options allow expressions to be specified.

Rich Text and Unicode Fonts

The ENTRYTITLE, ENTRYFOOTNOTE, and ENTRY statements support rich text and Unicode fonts. Many commonly used symbols have defined names (such as ALPHA). See the ENTRYFOOTNOTE in figure 5a and figure 5b for an example. See the GTL Reference Documentation for the full list of defined names.

INTEGRATION WITH ODS STYLES

ODS graphics output is tightly integrated with the ODS Styles. All graphical features of GTL derive their default visual attributes from specific style elements. A table of these associations can be found in the documentation for GTL. Some common associations are listed in Table 1. For full list, please see the GTL reference doc.

GTL Graphical Element	ODS Style Element
Title	GraphTitleText
Footnote	GraphFootnoteText
Axis Labels and Legend Title	GraphLabelText
Axis Tick Values	GraphValueText
Marker Data Labels and Curve Labels	GraphDatatext
Graph region outside the plot area	GraphBackground
Graph region within the axes bounds	GraphWalls
Ungrouped Scatter, Series, Step, Needles, Histograms, Bars	GraphDataDefault
Grouped Scatter, Series, Step, Needles, Bars	GraphData1 – GraphData12
Density plots	GraphFit
Box Plots	GraphBox

Table 1

Overriding visual attributes

By default, the visual attributes of various graphical elements of the graph are derived from specific style elements of the active ODS Style. The ODS Styles and usage of these attributes have been carefully selected to ensure maximum discriminability between various graphical elements in the graph.

However, users may override the visual attributes used for the graphical elements. This can be done at the element or attribute levels. For attribute settings, style references or hard-coded values may be used.

- **Element level:** Override a bundle of attributes of a plot by assigning the attributes option for the plot. For example, when group variable is not in effect, the series plot line derives its visual attributes from the GRAPHDATADEFAULT style element. If two series plots are used simultaneously, both will be drawn with the same attributes (from GRAPHDATADEFAULT). To differentiate between the two, the user can assign one plot to use the GRAPHDATA1 element and the other plot the GRAPHDATA2 element as follows:

  ```
  seriesplot x=date y=predict / lineattrs=graphdata1;
  seriesplot x=date y=actual / lineattrs=graphdata2;
  ```

  Now each plot will derive their attributes (color, pattern & line thickness) from different elements.

- **Attribute Level:** Individual attributes of the line may be set by setting the "attribute" level value instead of, or in addition to, the element level. So, in the example shown below, users can set the line thickness to 3 pixels, in addition to using the values from GraphData1, or just set the color value, while retaining other default settings:

  ```
  seriesplot x=date y=predict / lineattrs=graphdata1(thickness=3px);
  seriesplot x=date y=actual / lineattrs=(color=red)
  ```

- **Style References v/s Hard-Coded values:** In the examples above, style references and hard coded values have been used. Usage of style references is prudent and the recommended

policy. If the active style for the graph is changed, usage of style references will continue to produce good results. However, hard coded values can be used, as was done for `color=red`. In this case, the color of the line will always be red, regardless of the style of the graph. This could produce unpredictable results.

Note: When multiple plots are used (without groups), the user may request different visual attributes for each by setting the `cycleattrs` option on the `layout overlay` statement. This will cause each plot to obtain its visual properties from the GraphData1 – GraphData12 elements.

CONCLUSION

The Graph Template Language (GTL) is a powerful new syntax for the definition of analytical graphics, from the simple scatter plot to complex classification panels. Many analytical procedures and the new SG procedures leverage this syntax to produce their graphics. SAS users needing this level of control for their graphs can use this syntax directly.

REFERENCES

Rodriguez, Robert. "Getting Started with ODS Statistical Graphics in SAS 9.2." Paper presented at SAS Global Forum 2008.

Heath, Dan. "Effective Graphics Made Simple Using SAS/GRAPH SG Procedures." Paper presented at SAS Global Forum 2008.

Matange, Sanjay. "ODS Graphics Editor." Paper presented at SAS Global Forum 2008.

RECOMMENDED READING

Heath, Dan. 2007. "New SAS/GRAPH Procedures for Creating Statistical Graphics in Data Analysis." *Proceedings of the SAS Global Forum*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/events/sasglobalforum/previous/index.html.

Cartier, Jeff. 2006. "A Programmer's Introduction to the Graphics Template Language." *Proceedings of the Thirty-first Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/events/sasglobalforum/previous/index.html.

Rodriguez, R. N., and T. E. Balan. 2006. "Creating Statistical Graphics in SAS 9.2: What Every Statistical User Should Know." *Proceedings of the Thirty-first Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/events/sasglobalforum/previous/index.html.

CONTACT INFORMATION

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

Sanjay Matange
SAS Institute
SAS Campus Dr
Cary, NC 27512
(919) 531-6753
Fax (919) 677-4444
Sanjay.Matange@sas.com