Paper 262-31

A Programmer's Introduction to the Graphics Template Language

Jeff Cartier, SAS Institute Inc., Cary, NC

ABSTRACT

In SAS 9.2, the ODS Graphics Template Language (GTL) becomes production software. This powerful language is used by many SAS/STAT® and SAS/ETS® procedures (including the new SAS/GRAPH® procedures SGPLOT and SGSCATTER) to produce graphical output. The Graphics Template Language can also be used in conjunction with special DATA step features to produce graphs independently. This presentation helps you understand the basics of the Graph Template Language and create graphs with the DATA step.

Topics include:

- Basic concepts: template definition and storage, compilation, and run-time actions
- Graphical Layouts: features of Gridded, Overlay, and Lattice layouts
- Common tasks: customizing axes and adding legends and titles `
- Templates: making flexible templates by using dynamic and macro variables, conditional logic, and expressions
- Output: controlling image name, size, type, quality, and scaling
- Integration with ODS styles

INTRODUCTION

By now you are probably familiar with the ODS graphics that are automatically produced by many SAS statistical procedures in SAS®9 (see SUGI 31 paper 192-31 by Robert Rodriguez). These procedures use compiled programs written in the Graphics Template Language (GTL) to produce their graphs. Such procedures have been designed so you do not need to know anything about GTL programming details to get this graphical output.

GTL can be used by SAS application developers and other programmers to create sophisticated analytical graphics independent of the statistical procedures. This presentation helps you understand this new technology and shows the code for several types of graphs you can produce. The focus is on the organization and potential of this new language.

TEMPLATE COMPILATION AND RUNTIME ACTIONS

The DEFINE statement of PROC TEMPLATE allows you create specialized SAS files, called templates, that are used for controlling the appearance of ODS output. The template types you are most familiar with are STYLE, TABLE, and TAGSET. Starting in SAS 9.1, a new experimental template type was added. A STATGRAPH template describes the structure and appearance of a graph to be produced--similar to how a TABLE template describes the organization and content of a table. In SAS 9.2, the language that makes up the STATGRAPH template definition is production.

All templates are stored compiled programs. Here is the source program that produces a very simple STATGRAPH template named SCATTER:

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

NOTE: STATGRAPH 'Mygraphs.Scatter' has been saved to: SASUSER.TEMPLAT
```

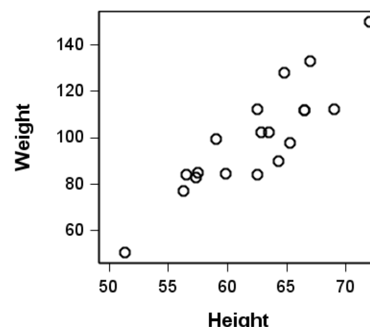
COMPILATION

When the above 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 MYGRAPHS item store (physically in SASUSER.TEMPLAT, by default). No graph is produced. It should be noted that STATGRAPH syntax requires that any required arguments be specified (X= and Y= for the SCATTERPLOT statement), but no checking for the existence of these variables is done at compile time (also notice that no reference to an input data set appears in the template).

RUNTIME ACTIONS

To produce a graph, a STATGRAPH template must be bound to an ODS data object at runtime and directed to an ODS destination. The simplest way to do this is with the new SAS 9.2 SAS/GRAPH procedure called SGRENDER, which was created to simplify the execution of user-defined templates such as SCATTER:

```
ods listing style=listing;

proc sgrender data=sashelp.class
              template="mygraphs.scatter";
run;
```



An ODS data object is constructed by comparing the template references to column names with variables that exist in the current data set. Here, there is a match for HEIGHT and WEIGHT so they are added to the data object and other variables are ignored. It is possible for a template to define new computed columns based on existing columns.

Once all the observations have been read, the data object and template definition are passed to a graph renderer that produces an image file for the graph which is then automatically integrated into the ODS destination. Rendering is totally independent of the legacy SAS/GRAPH GRSEG generation. In this example, a GIF image is created in the LISTING destination. The visual properties of the graph are determined by the ODS style in effect.

You should note that the SCATTER template is a very restrictive definition in that it can only create a plot of variables named HEIGHT and WEIGHT. As you will see later, STATGRAPH templates can be made more flexible by introducing dynamics or macro variables that allow variables and other information to be supplied at runtime.

GRAPHICAL LAYOUTS

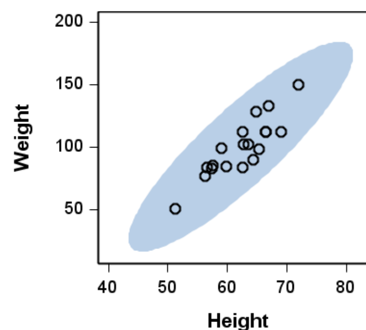
One of most powerful features of the GTL is the syntax built around hierarchical statement blocks called *layouts*. A layout is a container that arranges its contents in *cells*. A cell may contain a plot, a title, a legend, or even another layout. The layout arranges the cells in a predefined manner--into a single cell with its contents superimposed or into rows or columns of cells. All STATGRAPH template definitions begin with a LAYOUT statement block.

Single-cell Layouts that Support Superimposition		Multi-cell Layouts in Rows and Columns of Cells	
OVERLAY	2D plots, legends, text	GRIDDED	Basic grid of plots and text; all cells independent
OVERLAY3D	3D plots, text	LATTICE	Externalized axes, headers, sidebars
OVERLAYEQUATED	2D plots, legends, text, axes have equal sized units	DATALATTICE	Data-driven number of cells; 1-2 classifiers
PROTOTYPE	Simplified OVERLAY used in DATAPANEL and DATALATTICE	DATAPANEL	Data-driven number of cells; *n* classifiers

THE OVERLAY LAYOUT

The OVERLAY layout enables plots to be "stacked" in the order you declare them - with the first plot on the bottom of the stack. This layout manages the integration of plots based on different variables into single set of shared axes.

```
proc template;
  define statgraph mygraphs.scatteroverlay;
   layout overlay;
     ellipse x=height y=weight/ alpha=.01 type=predicted;
     scatterplot x=height y=weight;
   endlayout;
  end;
run;

proc sgrender data=sashelp.class
              template="mygraphs.scatteroverlay";
run;
```

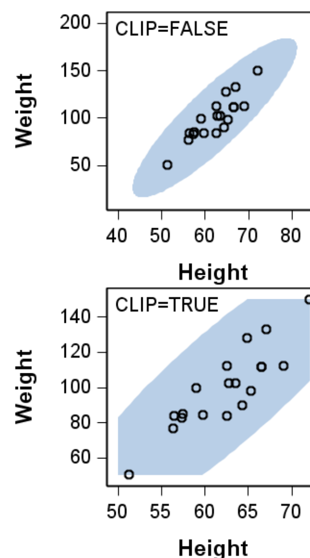


In this case, the X and Y data ranges for prediction ellipse are larger than the data ranges for the scatter plot and the layout automatically adjusts the range for each axis.

THE GRIDDED LAYOUT

The GRIDDED layout creates a single- or multi-cell graph. It is primarily used to present multiple plots in a grid or to create a small table of text and statistics (inset) you want to embed in a graph. The cells are completely independent of one another. In this example, two OVERLAY layouts are nested in the GRIDDED layout to create a graph with two cells placed in one column.

```
proc template;
  define statgraph mygraphs.gridded;
    layout gridded / columns=1;
      layout overlay;
        ellipse x=height y=weight / clip=false
           alpha=.01 type=predicted;
        scatterplot x=height y=weight;
        entry "CLIP=FALSE" / autoalign=auto;
      endlayout;
      layout overlay;
        ellipse x=height y=weight / clip=true
          alpha=.01 type=predicted;
        scatterplot x=height y=weight;
        entry "CLIP=TRUE" autoalign=auto;
      endlayout
    endlayout;
  end;
run;

proc sgrender data=sashelp.class
        template="mygraphs.gridded";
run;
```

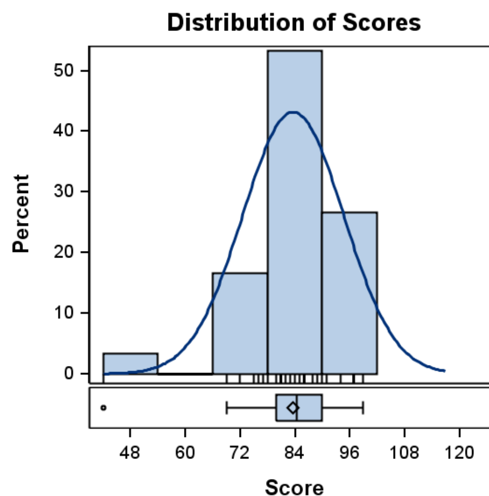


An ENTRY statement overlays text in each plot. The AUTOALIGN=AUTO option allows the software to judge where to place the text so as to avoid collision with the data being plotted. From this example, you can see that CLIP=TRUE option requests that the ellipse boundaries be ignored when scaling the axes.

THE LATTICE LAYOUT

The LATTICE layout is a multi-cell layout with special features for combining data ranges of plots in columns or rows and externalizing axis information so it is not repeated in each cell. This layout and the OVERLAY are the most important and frequently used layouts.

```
proc template;
  define statgraph mygraphs.distribution;
    layout lattice / columns=1 rows=2
                     rowweight=(.9 .1)
                     columndatarange=union
                     rowgutter=2px;
      columnaxes;
          latticeaxis / label='Score' ;
      endcolumnaxes;
      layout overlay / yaxisopts=(offsetmin=.03);
         entrytitle 'Distribution of Scores';
         histogram score / scale=percent;
         densityplot score / normal( ) ;
         fringeplot score;
      endlayout;
      boxplot y=score / orient=horizontal;
    endlayout;
  end;
run;
```



This lattice has one column and two rows. The row cells contain

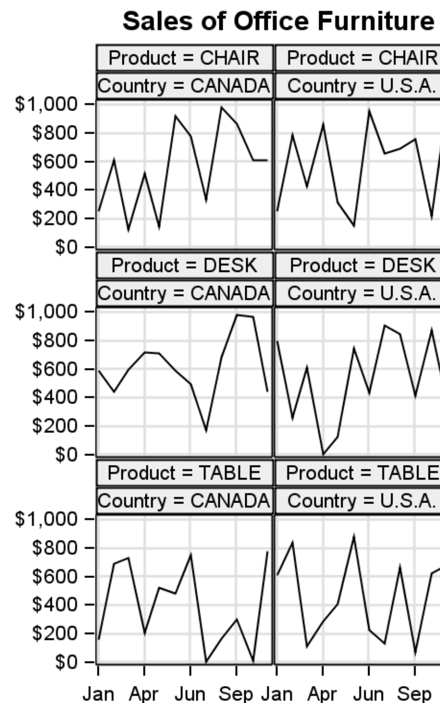
- an overlay consisting of a histogram, normal distribution curve, and "fringe plot" showing the location of individual observations under the histogram bins.
- a boxplot showing the median (line), mean (diamond marker), interquartile range, and outliers.

The overlay is apportioned 90% of the available height and the boxplot 10%. The data ranges the X axes of the two cells are merged and LATTICEAXIS statement externalizes a single X axis and sets its label.

THE DATAPANEL LAYOUT

The DATAPANEL layout is a data-driven layout. It creates a grid of plots based on one or more classification variables and a graphical prototype. A separate instance of the prototype cell is created for each crossing of the classifiers. The data for each prototype is a subset of the all the data based on the current classification level(s).

```
proc template;
 define statgraph mygraphs.datapanel;
  layout gridded;
    entrytitle 'Sales of Office Furniture';
    layout datapanel classvars=(product country)/
           rows=3 order=rowmajor
           rowaxisopts=(griddisplay=on label=' ')
           columnaxisopts=(griddisplay=on label=' ');

     layout prototype;
       seriesplot x=date y=actual;
     endlayout;

   endlayout;
  endlayout;
 end;
run;
```



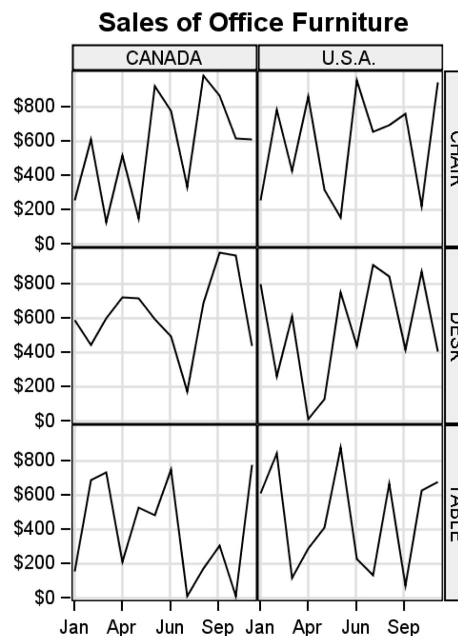
The number of unique values of the classification variables PRODUCT and COUNTRY determine the number of cells in the grid.

The cells are filled based on order in which the classifiers are declared, a grid dimension, and a wrapping order.

THE DATALATTICE LAYOUT

The DATALATTICE layout is similar to DATAPANEL but it requires a row classifier, a column classifier, or both.

```
proc template;
 define statgraph mygraphs.datalattice;
  layout gridded;
   entrytitle 'Sales of Office Furniture';
   layout datalattice
           columnvar=country rowvar=product/
             headerlabeldisplay=value
             rowaxisopts=(griddisplay=on label='')
             columnaxisopts=(griddisplay=on label='');
     layout prototype;
       seriesplot x=date y=actual;
     endlayout;
   endlayout;
  endlayout;
 end;
run;
```



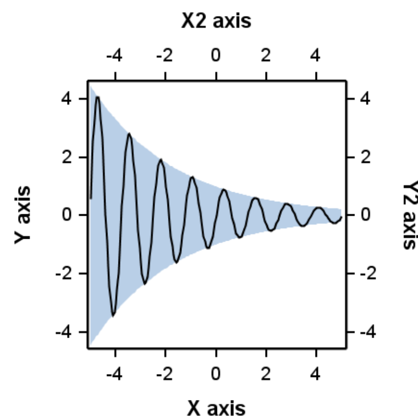
The number of unique values of the row classifier PRODUCT determines the number of rows in the grid. The number of unique values of the column classifier COUNTRY determines the number of columns in the grid.

This layout allows the labels for the classification levels to appear outside or inside the grid.

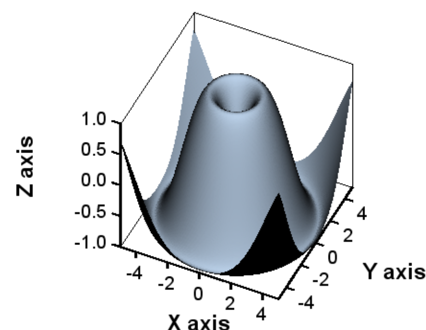
AXES

2D plots have four independent axes that can be used. By default, the X2 and Y2 axes are duplicates of X and Y axes and are not displayed unless requested. For 3D plots, there are the standard X, Y, and Z axes.

```
proc template;
 define statgraph mygraphs.axes2d;

  layout overlay /
       xaxisopts= (label='X axis')
       yaxisopts= (label='Y axis')
       x2axisopts=(label='X2 axis')
       y2axisopts=(label='Y2 axis');
    bandplot x=x limitupper=upper limitlower=lower /
                  display=(fill);
    seriesplot x=x y=y;
  endlayout;
 end;
run;
```

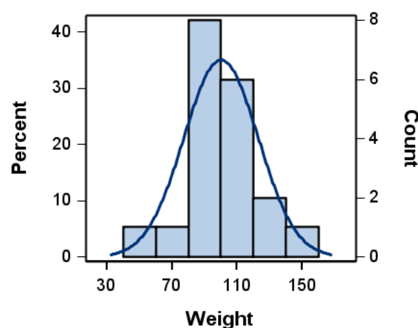



```
proc template;
  define statgraph mygraphs.axes3d;

    layout overlay3d /
         xaxisopts=(label="X axis")
         yaxisopts=(label="Y axis")
         zaxisopts=(label="Z axis")
         rotate=25  zoom=.6  tilt=45 cube=false;
      surfaceplotparm  x=x y=y z=z;
    endlayout;
  end;
run;
```



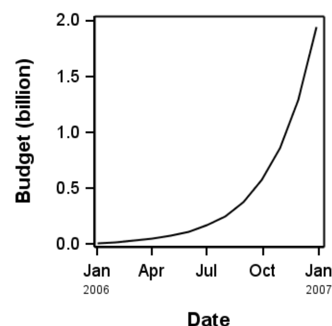
If any data are explicitly mapped to the X2 or Y2 axes, these axes automatically displayed. In the example below, the first histogram's frequency counts are mapped to the Y2 axis and the percentages are mapped to the Y axis.

```
proc template;
  define statgraph mygraphs.y2axis;
   layout overlay;
     histogram weight / scale=count yaxis=y2;
     histogram weight / scale=percent yaxis=y;
     densityplot weight / normal();
   endlayout;
 end;
run;
```



Axes can be of different types: LINEAR, DISCRETE, TIME, and LOG. Each type supports many different options.

```
proc template;
  define statgraph mygraphs.timeaxis;
   layout overlay /
     xaxisopts=(type=time
                timeopts=(tickvalueformat=monyy7.
                         splittickvalue=true interval=quarter))
     yaxisopts=(type=linear
           linearopts=(tickvalueformat=(extractscale=true)));
     seriesplot x=date y=budget;
   endlayout;
 end;
run;
```

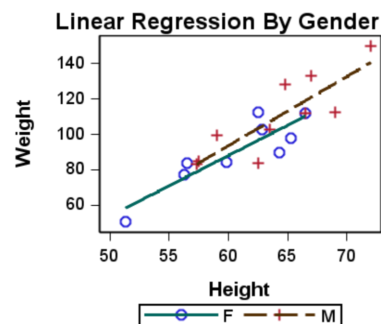


Here, the X axis values have the year portion split from the month portion to conserve tick labeling space. The Y axis values are very large. A factor of 10^9 (billion) was extracted and placed in the axis label automatically.

LEGENDS

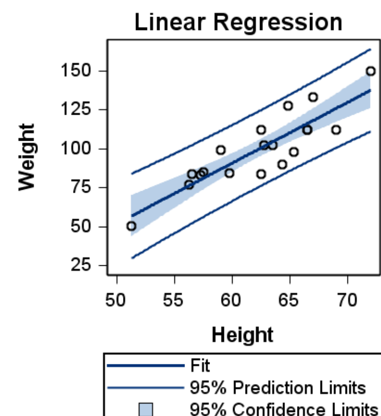
Many plot statements support a GROUP= option that partitions the data into unique values, performs separate analysis, if necessary, and automatically assigns distinct visual properties to each group value. The visual properties of group values are defined by the style in effect. In this example, a legend is added to associate group values with markers and line patterns from the scatter plot and linear regression lines. Note that a name is assigned to each plot. These names are used as arguments for the DISCRETELEGEND statement. The MERGE=TRUE option causes the marker and line entries to be combined.

```
proc template;
 define statgraph mygraphs.scatterfit;
   layout overlay;
     entrytitle 'Linear Regression By Gender';
     scatterplot x=height y=weight / group=sex name='scat';
     regressionplot x=height y=weight/ group=sex name='reg';
     discretelegend 'scat' 'reg' / merge=true border=true;
   endlayout;
 end;
run;
```



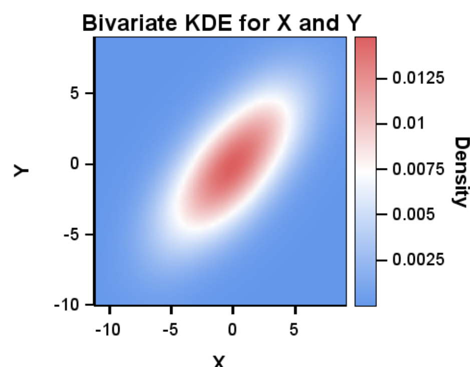
Sometimes you would like a legend to identify parts of the graph when there is no grouping. The plot's LEGENDLABEL= option specifies the legend entry text for each legend contributor. The legend representations automatically match the visual properties of the contributors.

```
proc template;
 define statgraph mygraphs.scatterfit2;
   layout overlay;
     entrytitle 'Linear Regression';
     modelband 'pred' / name='band1' display=(outline)
                        legendlabel='95% Prediction Limits';
     modelband 'conf' / name='band2' display=(fill)
                        legendlabel='95% Confidence Limits';
     scatterplot x=height y=weight;
     regressionplot x=height y=weight / name='line'
         alpha=.05 cli='pred' clm='conf' legendlabel='Fit';
     discretelegend  'line' 'band1' 'band2' /
                          across=1 border=true;
  endlayout;
 end;
run;
```



Some plot statements allow you to define a continuous response variable that is mapped to a color range. To show the colors associated with response values, you use a CONTINUOUSLEGEND statement. Note that the COLORMODEL= option of the contour specifies which style element to use for the color ramp. There are several two- and three-color ramps in any style from which to choose.

```
proc template;
 define statgraph mygraphs.contour;
   layout overlay;
    entrytitle 'Bivariate KDE for X and Y';
    contourplotparm x=x y=y z=density / name='cont'
                       filldisplay=gradient
                       colormodel=ThreeColorLowHigh;
    continuouslegend 'cont' / pad=(left=8)
                              title='Density';
   endlayout;
 end;
run;
```



MAKING TEMPLATES FLEXIBLE

There are several features in GTL that can make template definitions less restrictive on input data and more general in nature. This allows a single compiled template to produce many output variations.

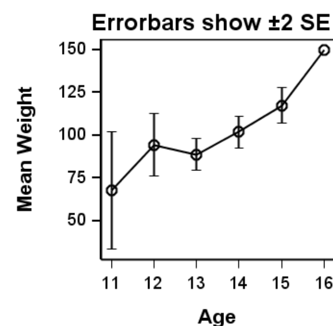
EXPRESSIONS AND FUNCTIONS

In GTL you can define constants or data columns with expressions. All expressions must be enclosed in an EVAL function. Within the expression you can use DATA step functions, arithmetic operators, and other special functions supported by the GTL. Text statements such as ENTRY and ENTRYTITLE support rich text and have special text commands such as {sup }, {sub }, and {unicode } to enable subscripting, superscripting, and Unicode characters.

The example below shows how the ± symbol was included in the title line using its hexadecimal Unicode value. Also, new data columns were computed for the upper and lower error bars of the scatter plot based on the input column StdErr.

```
 proc template;
 define statgraph mygraphs.expression;
  layout overlay;
    entrytitle 'Errorbars show ' {unicode '00B1'x} '2 SE';
    scatterplot x=age y=meanweight /
            yerrorlower=eval(meanweight - 2*stderr)
            yerrorupper=eval(meanweight + 2*stderr);
    seriesplot x=age y=meanweight;
  endlayout;
 end;
run;

proc summary data=sashelp.class nway;
   class age;
   var weight;
   output out=weights(drop=_type_ _freq_)
          mean=MeanWeight  stderr=StdErr;
run;

ods rtf style=listing;
proc print data=weights noobs;
run;
proc sgrender data=weights template='mygraphs.expression';
  label meanweight='Mean Weight';
run;
ods rtf close;
```

Age	MeanWeight	StdErr
11	67.750	17.2500
12	94.400	9.1807
13	88.667	4.6667
14	101.875	4.6069
15	117.375	5.2097



This example also illustrates a very common application theme: use a procedure's output data set as input to a template.

DYNAMICS AND MACRO VARIABLES

An extremely useful technique for generalizing templates is to define dynamics and/or macro variables that resolve when the template is executed.

Template Statement	Purpose	Value supplied by
DYNAMIC	defines dynamic(s)	1) DYNAMIC= suboption of ODS= option of FILE PRINT 2) 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( )

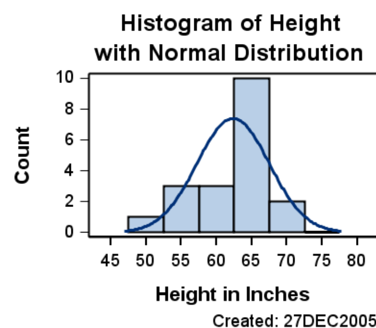
Template dynamics/ macro variables are normally used to supply required arguments and option values--you can't substitute keywords or statements.

If you use any of these statements, they must appear after the DEFINE STATGRAPH statement and before the outermost LAYOUT block.

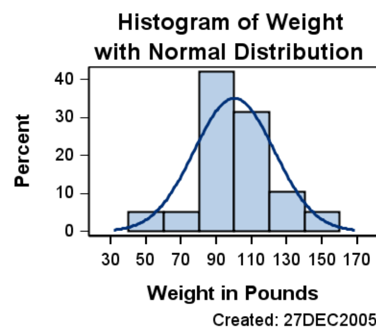
The next example shows a single template that can create a histogram for any variable. It defines both macro variables and dynamics for runtime substitution. No data dependent information is hard coded in the template. You can execute templates with special ODS features of the DATA step.

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

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

data _null_;
  set sashelp.class;
  if _n_=1 then do;
     call symput('bins','');
     call symput('scale','percent');
  end;
  file print ods=(template='mygraphs.dynamics'
     dynamic=(var='Weight' varlabel='Weight in Pounds') );
  put _ods_;
run;
```

PROC SGRENDER output:



DATA step output:



Notice that, in general, you should avoid open code macro variables references (with ampersands) unless you want resolution at template compile time. For example, if you had used &SYSDATE9 in the template definition, this reference would have resolved at template compilation, not when the template executed. You can think of the SYSDATE9 reference in the template as `eval(symget`('SYSDATE9')).

Another very interesting language feature is that options will "drop out" if you do not supply values for them at run time. For example, in the above template the only piece of information that is truly required is the name of the column (VAR) for the histogram statement. If you were to execute the template with

```
proc sgrender data=sashelp.class template='mygraphs.dynamics';
  dynamic var='Weight';
run;
```

you would find that this is the template code† that would be executed:

```
layout overlay / xaxisopts=( );
   entrytitle 'Histogram of ' Weight ;
   entrytitle 'with Normal Distribution';
   histogram Weight /  ;
   densityplot Weight / normal( );
   entryfootnote halign=right 'Created: ' 27DEC2005 /
      textattrs=GraphValueText;
endlayout;
```

CONDITIONAL LOGIC

You can make a template take a different code path (execute alternate statements) using conditional logic. The evaluation of a logical expression must generate one or more complete statements (not portions of statements). All conditional logic uses one of these constructs:

<table>
<tr>
<td>if (condition)
  statement(s);
endif;</td>
<td>if (condition)
  statement(s);
else
  statement(s);
endif;</td>
<td>if (condition1)
  statement(s);
else if (condition2)
  statement(s);
else
  statement(s);
endif;</td>
</tr>
</table>

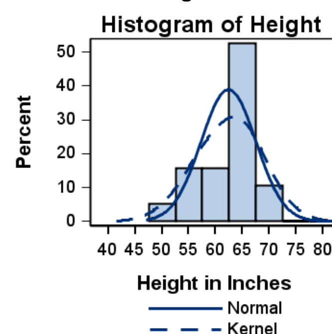
C*ondition* must be enclosed in parentheses. *Condition* may be any standard SAS expression involving arithmetic, logical, comparison, Boolean, or concatenation operators as well as SAS DATA step functions. The expression resolves to a single scalar value which is true or false.

† GTL code is not sent directly to the renderer. At runtime, the graphics template communicates with a supplied TAGSET template. It translates GTL into markup language, such as XML, that a renderer acts upon. It is the tagset logic that is conditionally "dropping out" options when no option value is supplied.

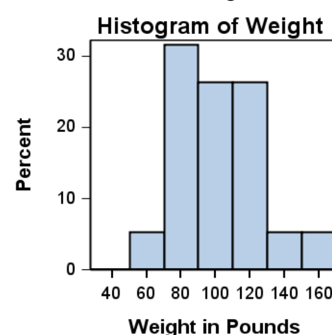
In this example, we extend our histogram template to allow a normal distribution curve, a Kernel Density Estimate distribution curve, both, or neither to be conditionally overlaid on the histogram.

```
proc template;
  define statgraph mygraphs.conditional;
    dynamic VAR VARLABEL BINS CURVE;
    layout overlay / xaxisopts=(label=VARLABEL);
     entrytitle 'Histogram of ' VAR;
     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;
  end;
run;

proc sgrender data=sashelp.class
                   template='mygraphs.conditional';
  dynamic var='Height' varlabel='Height in Inches'
curve='all';
run;

proc sgrender data=sashelp.class
              template='mygraphs.conditional';
  dynamic var='Weight' varlabel='Weight in Pounds';
run;
```

Result of setting CURVE='all'



Result of not setting CURVE:



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

Most of the supplied templates for statistical procedures use conditional logical and dynamics to allow a single template to produce many variations of a standard statistical graph.

CONTROLLING OUTPUT

So far, we have focused on template definitions to get specific graphs and have shown how to produce output. Ultimately, you will also need to tailor the graphical environment to get the exact output you desire.

ODS GRAPHICS STATEMENT

This statement is used to modify the environment in which graphics templates are executed. It is somewhat like the GOPTIONS statement in intent, but it has no effect on legacy SAS/GRAPH output. Similarly, the GOPTIONS statement does not affect GTL-rendered graphs.

You use the ODS GRAPHICS statement to control

- whether ODS graphics is enabled or not
- the type and name of the image created
- the size of the image
- whether features such as scaling and anti-aliasing are used.

```
ODS GRAPHICS ON < / RESET
                     IMAGEFMT= STATIC | GIF | PNG | JPEG | other-types
                     IMAGENAME= 'path-and-name'
                     HEIGHT= size   WIDTH= size /* default: HEIGHT=480px WIDTH=640px */
                     SCALE= ON | OFF
                     BORDER= ON | OFF
                     ANTIALIASING = ON | OFF
                     IMAGEMAP = ON | OFF    /* produces tooltips for HTML destination only */
                     more-options
                 >;
    procedures or data steps

ODS GRAPHICS OFF;
```

For example, RESET can be used to prevent the image name from incrementing (scatter1, scatter2, etc.) every time the graph is produced. If only one option, HEIGHT= or WIDTH=, is specified, then the aspect ratio that is defined by the template or the default aspect ratio (3/4) is maintained. The SCALE=OFF option prevents the fonts, markers, and other graphical elements from scaling up or down in size based on image size.

```
ods graphics on / reset imagename='c:\odsgraphs\scatter' imagefmt=png
                  height=175px width=200px;
  proc sgrender data=sashelp.class template='mygraphs.scatter';
  run;
ods graphics off;
```

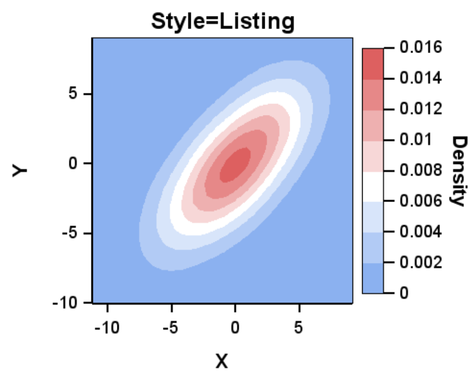
ODS STYLES

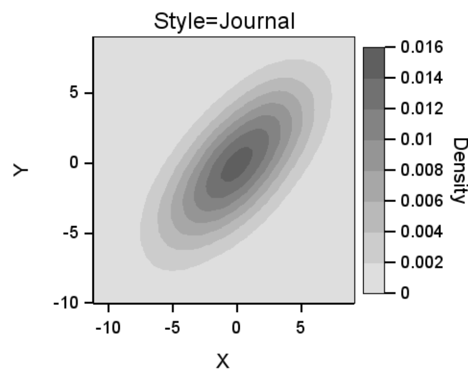
When any graphics template is executed, there is always an ODS style in effect that governs the appearance of the output. All the graphical output in this paper was generated with the RTF output destination and the LISTING style:

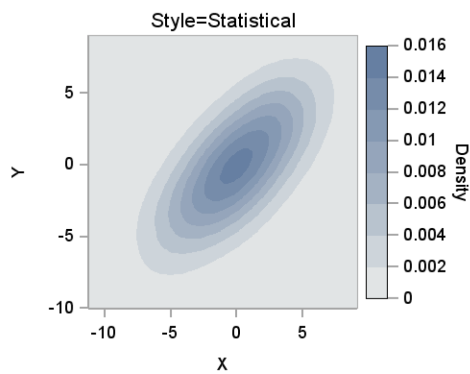
```
ods rtf style=listing;

ods graphics on / height=175px width=200px border=off;
proc sgrender data=sashelp.class template='mygraphs.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, you will always get very reasonable overall appearance of tables and graphs for any supplied ODS style.

This next example shows how graphical appearance options can be set with references to style elements. Nearly all supplied STATGRAPH templates use this technique because it permits changes in graph appearance by style modification instead of graphical template modification.







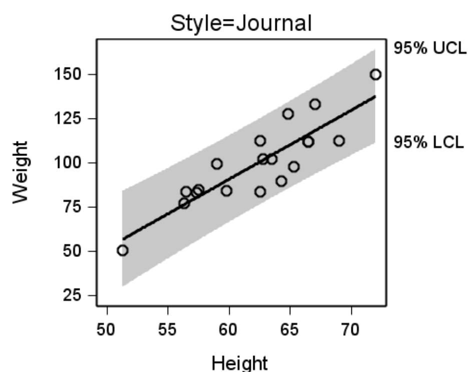
GTL:

```
contourplotparm x=x y=y z=density /
  filldisplay=fill nhint=9
  colormodel=ThreeColorLowHigh ;
```

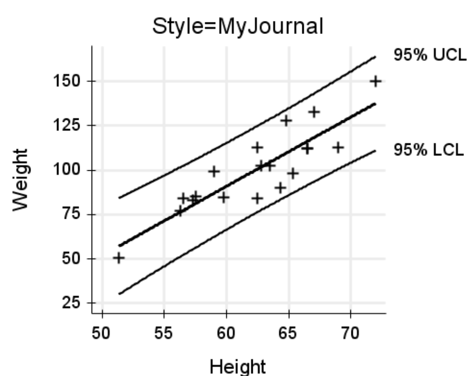
Style template:

```
style ThreeColorLowHigh /
  endcolor = GraphColors('gramp3cend')
  neutralcolor = GraphColors('gramp3cneutral')
startcolor = GraphColors('gramp3cstart');
```

Styles minimally contain information about fonts, colors, lines and markers for various parts of the graph. This example shows that other appearance features can be set from the style. The supplied JOURNAL style uses filled non-outlined bands, a frame around the axes, outside axis ticks, and a circle for the default marker.




```
proc template;
  define statgraph mygraphs.styledemo;
   dynamic TEXT;
   layout overlay;
     entrytitle TEXT;
     modelband 'p' / curvelabelupper='95% UCL'
                     curvelabellower='95% LCL';
     scatterplot x=height y=weight;
     regressionplot x=height y=weight /
                     cli='p' alpha=.05;
   endlayout;
  end;
run;
```




```
proc template;
   define style Styles.MyJournal;
    parent = styles.Journal;
     style GraphBand from GraphGraphBand /
          displayopts = "outline" ;
     style GraphWalls from GraphWalls /
          frameborder = off ;
     style GraphAxisLines from GraphAxisLines /
          tickdisplay = "across" ;
     style GraphGridLines from GraphGridLines /
          displayopts = "on" ;
     style GraphDataDefault from GraphDataDefault /
            markersymbol='plus';
  end;
run;
```

Of course, the appearance changes defined by the MYJOURNAL style below could have been made with plot statement options in GTL, but if appearance consistency is an issue across graphs, it makes more sense to use custom style approach.

CONCLUSION

From the beginning, GTL was designed to

- allow basic plots to be combined in many different ways.
- have built-in analytical and general computational capabilities.
- be sufficiently high-level in nature, so only minimal information needs to be supplied.
- be sufficiently low-level in nature, so small details can be specified if desired.
- incorporate many automatic features such as label-collision avoidance algorithms, polices for axis tick thinning, auto-placement of text and legend, and different levels of scaling behavior.
- use state-of-the-art rendering technology with a wide choice of output possibilities.
- integrate with all ODS styles and destinations.
- be extensible. New plot types and layouts are already planned for future releases.

As you can see from the examples presented here, the notion of layouts is the cornerstone of GTL. They provide general purpose containers for simple and complex gridding of plots, overlays, and more advanced trellis-like arrangements.

GTL has basic 2D plot types for bandplot, barchart, boxplot, blockplot, contour, densityplot, ellipse, fringeplot, histogram, loessplot, modelband, needleplot, regressionplot, splineplot, scatterplot, scatterplotmatrix, seriesplot, stepplot, and vectorplot. There are 3D plots for bivariate histograms and for general surfaces. There are statements for reference lines, drop lines, point and slope lines, various legends and text. This paper has illustrated only about one-third of the available GTL statements and very few statement options.

As a programmer/application developer, you can use GTL create sophisticated analytical graphs.

REFERENCES

Rodriguez, R. 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*, San Francisco, CA.

CONTACT INFORMATION

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

Jeff Cartier
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27511
Phone: (919) 677-8000

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

Other brand and product names are trademarks of their respective companies.