

The Power of the Graphics Template Language

Jeff Cartier, SAS Institute Inc., Cary, NC

ABSTRACT

In SAS 9.2, the ODS Graphics Template Language becomes production software. You will see more SAS procedures employing GTL to create graphical output and refinements to procedures that already used GTL in SAS 9.1. Programmers now have the opportunity to modify procedure output and use GTL to produce graphs of their own design. This presentation provides a programmer's overview of the basic concepts and language features of GTL.

- Basic ODS template concepts: compilation and runtime actions
- Gridded, Overlay, and Lattice layouts: choosing the right one
- Basic GTL syntax: combining layouts, plots, axes, legends, titles, etc.
- Plot types: parameterized vs. computed, rules of overlaying
- Making templates flexible: conditional logic, expressions, using dynamic and macro variables.
- Producing output: executing templates with the Data step, ODS GRAPHICS statement.

INTRODUCTION

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, TAGSET and TABLE. Starting in SAS 9.1, a new template type was added. A STATGRAPH template describes the structure and appearance of a graph to be produced – akin to how a TABLE template describes the organization and content of a table.

COMPILATION AND RUNTIME ACTIONS

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

```
proc template;
  define statgraph mygraphs.scatter;
    layout gridded;
      entrytitle "Scatter Plot of Height vs. Weight";
      scatterplot x=height y=weight;
    endlayout;
  end;
run;
```

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 is created and stored in the MYGRAPHS item store (physically in SASUSER.TEMPLAT, by default). It should be noted that STATGRAPH syntax requires that SAS variable names be assigned to the required X= and Y= options for the SCATTERPLOT statement, but no checking for the existence of these variables is done at compile time (notice that no reference to an input data source appears in the program).

RUNTIME

To produce a graph, a STATGRAPH template must be bound to an ODS data object at runtime. This can be done in two ways:

- 1) Any SAS procedure that supports ODS graphical templates will build appropriate ODS data object(s) based on the procedure options the user specifies. These data objects are then bound to one or more predefined graphics templates. Graphical output is automatically integrated into the ODS destination. In general, a user-defined template such as SCATTER with references to arbitrary variables cannot be accessed from a procedure.
- 2) The DATA step supports statements and options that build and bind a data object to a template.

```
data _null_;
  set sashelp.class;
  file print ods=(template="mygraphs.scatter");
  put _ods_;
run;
```

In the Data step case, 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 that the template defines new computed columns.

Once all the observations have been read, the data object is passed to the template SCATTER. At this point the actual rendering of the graph begins and the image file for the graph is created.

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 we will see later, STATGRAPH templates can be made more flexible by introducing dynamics or macro variables.

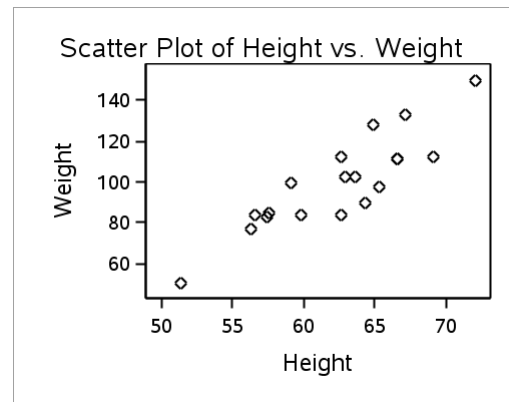
GRAPHICAL LAYOUTS

One of most powerful aspects of the GTL is the syntax build around hierarchical statement blocks called *layouts*. A layout is a container that arranges its contents as *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 rows or columns or into a single cell with its contents superimposed. All STATGRAPH templates definitions begin with a LAYOUT block.

LAYOUT GRIDDED

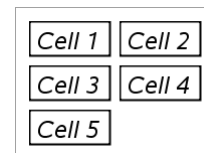
A gridded layout is the simplest organization of cells. By default, the output of each contained statement is placed in its own cell. The cells are arranged into one column and each is center aligned. The example below shows a gridded layout with one column and two rows – one for ENTRYTITLE and one for the SCATTERPLOT.

```
layout gridded;
  entrytitle "Scatter Plot of Height vs. Weight";
  scatterplot x=height y=weight;
endlayout;
```



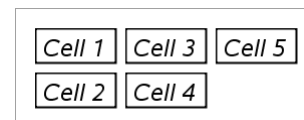
This example uses just ENTRYFOOTNOTE statements to populate the cells. Here we set COLUMNS=2, which causes the layout to wrap horizontally (start a new row) whenever two columns are filled from left to right. This is called ROWMAJOR ordering and is the default.

```
layout gridded / columns=2;
  entryfootnote "Cell 1" / border=true;
  entryfootnote "Cell 2" / border=true;
  entryfootnote "Cell 3" / border=true;
  entryfootnote "Cell 4" / border=true;
  entryfootnote "Cell 5" / border=true;
endlayout;
```



You can also request that cells be filled from top to bottom. Here we set ROWS=2 and ORDER= COLUMNMAJOR to force vertical wrapping whenever two rows are filled.

```
layout gridded / rows=2 order=columnmajor;
  entryfootnote "Cell 1" / border=true;
  entryfootnote "Cell 2" / border=true;
  entryfootnote "Cell 3" / border=true;
  entryfootnote "Cell 4" / border=true;
  entryfootnote "Cell 5" / border=true;
endlayout;
```

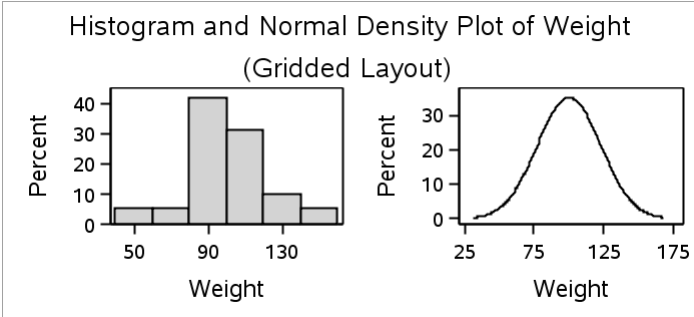


This example shows how layouts can be nested. The outermost layout has 3 cells arranged in 1 column. Note that the nested gridded layout is counted a single cell.

```

layout gridded;
  entrytitle "Histogram and Normal Density Plot of Weight";
  entrytitle "(Gridded Layout)";
  layout gridded / columns=2;
    histogram weight / xaxisopts=(label="Weight");
    densityplot weight / xaxisopts=(label="Weight");
  endlayout;
endlayout;

```



Each plot has options for controlling its own axis properties. The reason for labeling the X-axis of each plot will be discussed later.

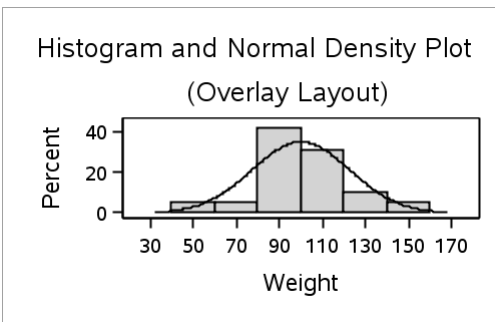
LAYOUT OVERLAY

The overlay layout creates a single-cell plot. It is used primarily to combine (superimpose) plots. Plots are "stacked" in the order you declare them, with the first on the bottom. All plots in the overlay "lose" their individual axes. Instead a consolidated sets of axes "owned" by the layout is created. Each overlaid plot's data ranges contribute to the overall scaling of the axes. Compare the axes from the output of the gridded layout above with that of the overlay layout below.

```

layout overlay / xaxisopts=(label="Weight");
  entrytitle "Histogram and Normal Density Plot";
  entrytitle "(Overlay Layout)";
  histogram weight;
  densityplot weight;
endlayout;

```



It important to remember that when coding overlays, many options that would normally appear on individual plot statements must be relocated to the overlay statement. In this case, any XAXISOPTS= option on a plot statement is ignored. Only an XAXISOPTS= option on the LAYOUT OVERLAY statement will be used. Such options are clearly noted in the documentation.

On the subject of axes, all 2D plots support 4 axes:

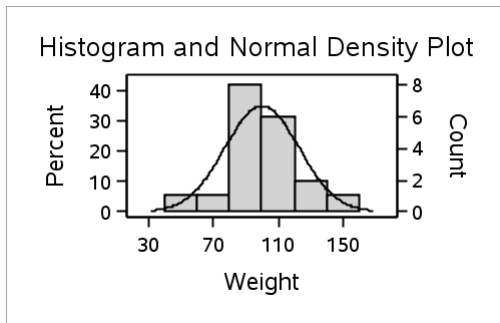
- X (lower horizontal)
- X2 (upper horizontal)
- Y (left vertical)
- Y2 (right vertical)

3D plots support X Y and Z and axes.

Often you would like have either a duplicated or totally independent Y-axis on your graph (both a Y and Y2 axes). Every 2D plot supports XAXIS= and YAXIS= options that control the axis to which the plot's data are mapped.

In this example, two histograms are overlaid with different computed statistics.

```
layout overlay / xaxisopts=(label="Weight");
  entrytitle "Histogram and Normal Density Plot";
  histogram weight / scale=count yaxis=y2;
  histogram weight;
  densityplot weight;
endlayout;
```



The YAXIS=Y2 option forces the Y data for the first histogram to be mapped to the Y2 (right vertical) axis

The second Histogram and Densityplot are mapped to the Y axis, by default.

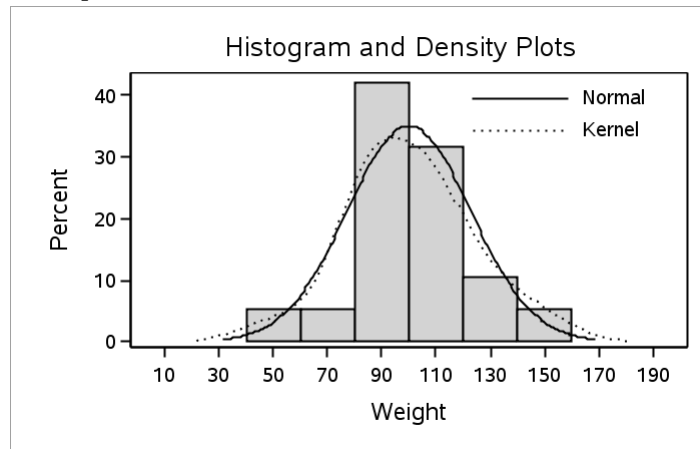
Corresponding to each axis, there are XAXISOPTS=, X2AXISOPTS=, YAXISOPTS=, Y2AXISOPTS=, and ZAXISOPTS for separate axis fine tuning.

When creating plots with grouped data or plots with multiple fit lines, you often need a legend. In the example below we show both normal and kernel density curves overlaid on the histogram and a legend that identifies each curve. For each curve, we assign

- o a unique name (NAME= option)
- o different visual appearance (LINEATTRS= option)
- o a descriptive label to appear in a legend (LEGENDLABEL= option)

The DISCRETELEGEND statement requires the names of one or more plots that will contribute to the legend. By default, the legend will appear outside and below the X-Axis. Here we place it inside the axis area and position it in the upper right corner. The ACROSS=1 option forces the legend entries to appear in one column.

```
layout overlay / xaxisopts=(label="Weight");
  entrytitle "Histogram and Density Plots";
  histogram weight;
  densityplot weight / name="norm"
    legendlabel="Normal" lineattrs=(pattern=solid);
  densityplot weight / name="kern" kernel()
    legendlabel="Kernel" lineattrs=(pattern=dash);
  discretelegend "norm" "kern" / across=1 location=inside halign=right valign=top;
endlayout;
```



Here we chose to visually differentiate the curves by line pattern only. We could have used line color or line thickness or any combination of these line attributes:

```
lineattrs=(pattern=dash color=gray
  thickness=2)
```

The legend representation automatically matches whatever properties we chose for the contributors.

LAYOUT LATTICE

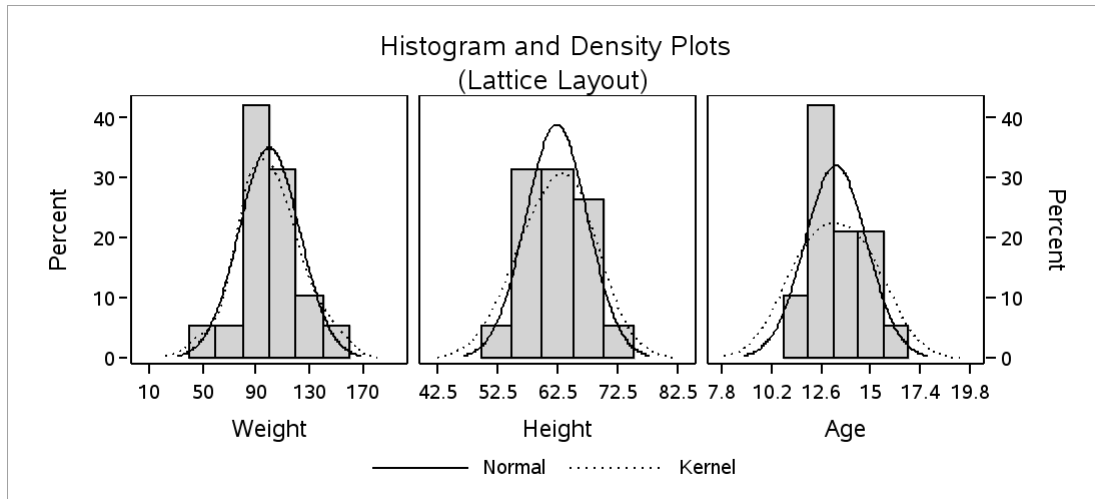
The lattice layout is a multicell layout that combines features of gridded and overlay layouts and offers reserved areas for additional formatting:

- o 4 sidebars (top, bottom, left, and right) that span all rows and columns
- o individual row and column headings
- o individual cell headings
- o axis scaling on a per cell, per row, per column, all rows, or all columns basis

```

layout lattice / columns=3 columngutter=3 vaxisrange=union;
  layout overlay / xaxisopts=(label="Weight");
    histogram weight;
    densityplot weight;
    densityplot weight / kernel() lineattrs=(pattern=dash);
  endlayout;
  layout overlay / xaxisopts=(label="Height");
    histogram height;
    densityplot height;
    densityplot height / kernel() lineattrs=(pattern=dash);
  endlayout;
  layout overlay / xaxisopts=(label="Age");
    histogram age;
    densityplot age / name="n" legendlabel="Normal";
    densityplot age / name="k" legendlabel="Kernel" kernel() lineattrs=(pattern=dash);
  endlayout;
  sidebar / align=top;
    layout gridded;
      entrytitle "Histogram and Density Plots";
      entrytitle "(Lattice Layout)";
    endlayout;
  endsidebar;
  sidebar / align=bottom;
    discretelegend "n" "k" / down=1;
  endsidebar;
  rowaxes;
    axiscomp / label="Percent";
  endrowaxes;
  row2axes;
    axiscomp / label="Percent";
  endrow2axes;
endlayout;

```



This is a relatively simple lattice with 3 columns and 1 row, each cell containing an overlaid histogram. The `VAXISRANGE=UNION` option indicates that the vertical axes of all cells in a row will have the same scale. The `COLUMNGUTTER=3` option adds some spacing between the plots. The titles appear in the top sidebar and the legend in the bottom sidebar. The `ROWAXES` and `ROW2AXES` blocks control the left and right axes.

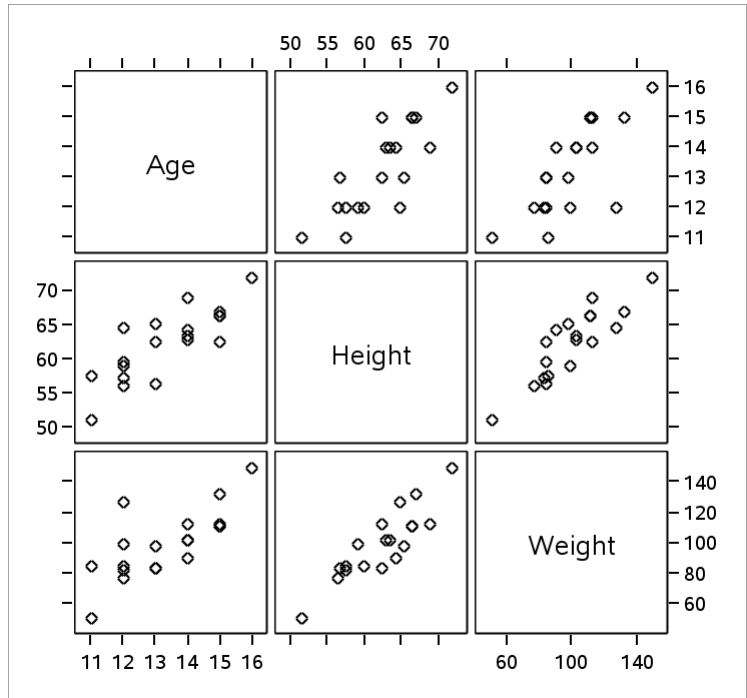
A scatterplot matrix is a widely recognized representation for viewing all possible pairings of a set of variables (or one set of variables with another set of variables).

The matrix shown here is for the variables Age, Height and Weight (NxN).

The matrix economizes on space by sharing axis scales both vertically and horizontally.

The SCATTERPLOTMATRIX statement can construct matrices with NxN or MxN cells. For the NxN case, the diagonal can be populated with univariate plots such as histograms and density curves.

```
layout gridded;
  scatterplotmatrix age height weight;
endlayout;
```



This kind of matrix can also be made with the lattice layout. This particular lattice would require about 40 statements.

PARAMETERIZED VS. COMPUTED PLOTS

Many plots in the GTL come in two versions:

Barchart	Boxplot	Histogram	BivariateHistogram	Surfaceplot	Contourplot
BarchartParm	BoxplotParm	HistogramParm	BivariateHistogramParm	SurfaceplotParm	ContourplotParm

PARAMETERIZED PLOTS

Parameterized plots require that all of your input data be pre-summarized. Computed plots accept raw data and perform calculations on it to get the resulting graph.

For example, compare the syntax for Barchart and BarchartParm (only a portion of the syntax is shown).

<pre>barchart x= column < y= numeric-column > / stat= statistic confidence= level errorbar= UPPER LOWER BOTH;</pre>	<p>The X <i>column</i> contains raw data values. All distinct values become midpoints. If the Y <i>column</i> is specified, it is summarized within each midpoint and a statistic like SUM or MEAN is computed. Otherwise, the midpoints represent frequencies or percents. If requested, error bars will be computed with a specified confidence level.</p>
<pre>barchartparm x=column y= numeric-column / errorupper= numeric-column errorlower=numeric-column ;</pre>	<p>The X <i>column</i> contains one observation per midpoint. The Y <i>column</i> contains a response value for the midpoints. There is no internal binning of the data or computation for response values. To show error bars, columns containing the values must be provided.</p>

To use parameterized plots, you typically first create an output data from some procedure or the data step and then pass the summarized data to the template.

In the next example, we use PROC SUMMARY to compute summary statistics for a BARCHARTPARGM.

```

proc template;
  define statgraph mygraphs.barparm;
    layout overlay;
    entrytitle "Bar Chart of Mean Weight";
    entrytitle "with Upper and Lower CLM";
    barchartparm x=sex y=weight_mean /
      errorupper=uclm errorlower=lclm;
  endlayout;
end;
run;

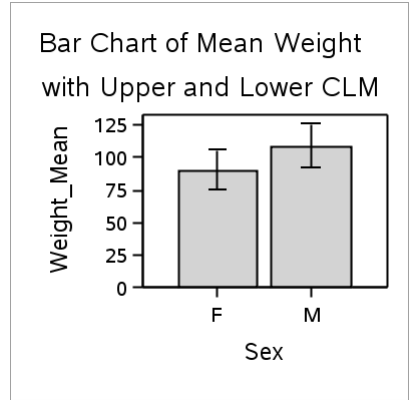
proc summary data=sashelp.class nway;
  class sex;
  var weight;
  output out=class(drop=_type_ _freq_)
    mean=Weight_Mean lclm=LCLM uclm=UCLM;
run;

ods rtf style=journal;

/* so we can see the input data */
proc print data= class noobs;
run;
data _null_;
  set class;
  file print ods=(template="mygraphs.barparm");
  put _ods_;
run;
ods rtf close;

```

Sex	Weight_Mean	LCLM	UCLM
F	90.111	75.2113	105.011
M	108.950	92.6920	125.208



Because computations are performed independent of the template, you could include different statistics or use your own algorithms to compute the statistics.

COMPUTED PLOTS

An important feature of GTL is the Statistical Graphics Engine (SGE). One of SGE's roles is to automatically generate computed columns for you. Plots such as BARChart, HISTOGRAM, DENSITYPlot, ELLIPSE and BOXPlot use this functionality.

```

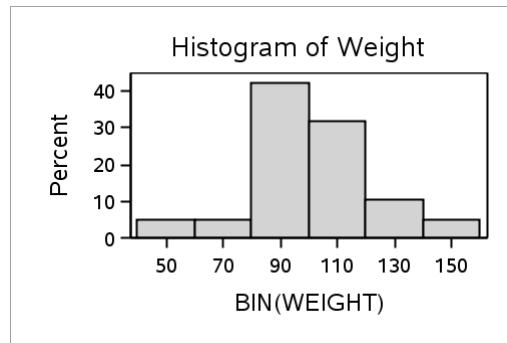
proc template;
  define statgraph mygraphs.histogram;
    layout overlay;
    entrytitle "Histogram of Weight";
    histogram weight;
  endlayout;
end;
run;

/* so we can see the output object */
ods output histobj=histdata;

ods rtf style=journal;
data _null_;
  set sashelp.class;
  file print ods=(
    object=histobj
    template="mygraphs.histogram"
  );
  put _ods_;
run;

proc print data=histdata label;
run;
ods rtf close;

```



Obs	BIN(WEIGHT)	Percent
1	50.000	5.263
2	70.000	5.263
3	90.000	42.105
4	110.000	31.579
5	130.000	10.526
6	150.000	5.263

The ODS OUTPUT statement and PROC PRINT were added to illustrate the nature of the data object created in this case.

The name and label of the computed column will change based on what options you specify with the plot statement. To control the axis labeling you may want to use XAXISOPTS= or YAXISOPTS= in your template:

```
layout overlay / xaxisopts=(label="Weight") ;
  entrytitle "Histogram of Weight";
  histogram weight;
endlayout;
```

MAKING TEMPLATES FLEXIBLE

There are several features in GTL that can make template definitions less restrictive on input data and more general in nature.

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 EVAL you can use data step functions, arithmetic operators, and other special functions supported by the SGE. 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.

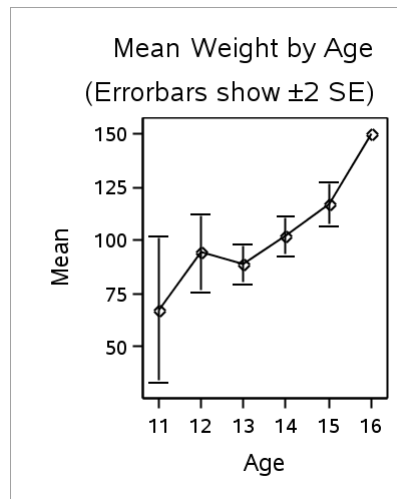
The example below shows how the ± symbol was included in the second title line using its hexadecimal Unicode value. Also, new data columns were computed for the upper and lower error bars of the scatter plot. The X column for the series plot was sorted to assure the correct joining of the data points (in this particular example, the sort wasn't needed, but in general being able to sort via template code is helpful).

```
proc template;
define statgraph Mygraphs.expression;
  layout overlay;
  entrytitle "Mean Weight by Age";
  entrytitle "(Errorbars show " {unicode "00B1"x}
    "2 SE)";
  scatterplot x=age y=mean /
    yerrorlower=eval(mean - 2*stderr)
    yerrorupper=eval(mean + 2*stderr);
  seriesplot x=eval(sort(age)) y=mean;
endlayout;
end;
run;
```

Age	Mean	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

```
proc summary data=sashelp.class nway;
  class age;
  var weight;
  output out=class(drop=_type_ _freq_)
    mean=Mean stderr=stderr;
run;
ods rtf style=journal;
proc print data=class noobs;
run;

data _null_;
  set class;
  file print ods=(template="mygraphs.expression");
  put _ods_;
run;
ods rtf close;
```



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.

DYNAMIC statement	defines dynamic(s)	Value supplied via DYNAMIC= suboption of ODS= option of FILE PRINT.
MVAR statement	defines macro variable(s)	Value supplied via %LET or CALL SYMPUT()
NMVAR statement	defines macro variable(s) that resolves to a number	Value supplied via %LET or CALL SYMPUT()

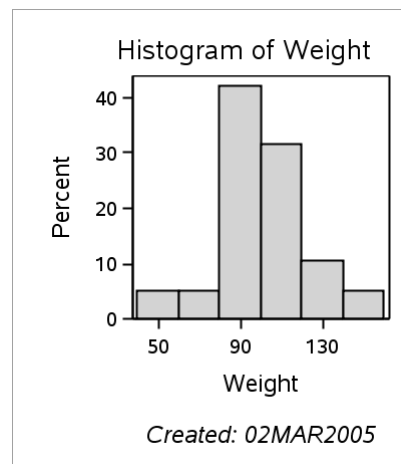
If you use any of these statements, they must appear after the DEFINE STATGRAPH statement and before the outermost LAYOUT block.

The next example creates a generic template for a histogram. No data dependent information is hard coded in the template.

```
proc template;
  define statgraph mygraphs.dynhist;
    mvar SYSDATE9 VARLABEL;
    dynamic VAR WIDTH HEIGHT;
    layout overlay / width=WIDTH height=HEIGHT
      xaxisopts=(label=VARLABEL);
      entrytitle "Histogram of " VARLABEL;
      histogram VAR;
      entryfootnote "Created: " SYSDATE9;
    endlayout;
  end;
run;

data _null_;
  set sashelp.class;
  if _n_=1 then
    call symput("VARLABEL",trim(vlabel(weight)));

  file print ods=(template="mygraphs.dynhist"
    dynamic=(VAR="weight" WIDTH=200 HEIGHT=230)
  );
  put _ods_;
run;
```



Macro variables or dynamics can resolve to data columns or option values. You cannot use them to build pieces of the grammar (e.g. you can't define a dynamic WIDTH that resolves to WIDTH=200).

Notice that, in general, you should avoid open code macro variables references (with ampersand) 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 characteristic of macro variables and dynamics 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

```
data _null_;
  set sashelp.class;
  file print ods=(template="mygraphs.dynhist" dynamic=(VAR="weight"));
  put _ods_;
run;
```

you would find that this is the code* that would be generated:

```
layout overlay / xaxisopts=( );
  entrytitle "Histogram of ";
  histogram weight;
  entryfootnote "Created: " 02MAR2005;
endlayout;
```

* STATGRAPH template code is not directly rendered as such. At runtime the graphics template communicates with another template of type TAGSET. A tagset generates markup language, such as XML, that a renderer consumes. It is the tagset logic that is conditionally "dropping out" options when no option value is supplied.

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:

```
if ( condition )
  statement(s);
endif;
```

```
if ( condition )
  statement(s);
else
  statement(s);
endif;
```

```
if ( condition )
  statement(s);
else if ( condition )
  statement(s);
else
  statement(s);
endif;
```

Notice that *condition* 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.

```
proc template;
  define statgraph mygraphs.conditional;
    mvar VARLABEL;
    dynamic VAR BINS CURVE;
    layout overlay / xaxisopts=(label=VARLABEL);
    entrytitle "Histogram of " VARLABEL;
    histogram VAR / nbins=BINS;

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

    endif;

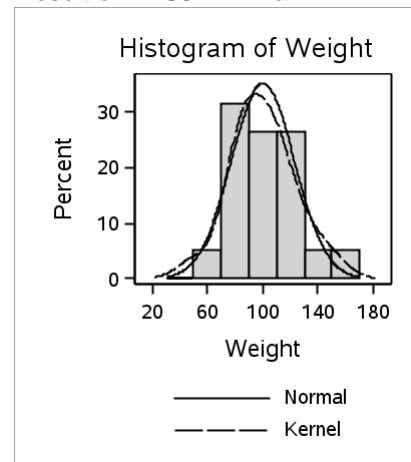
    if (upcase(CURVE) in ("ALL" "NORMAL"))
      densityplot VAR / normal() name="n"
                  legendlabel="Normal";

    endif;

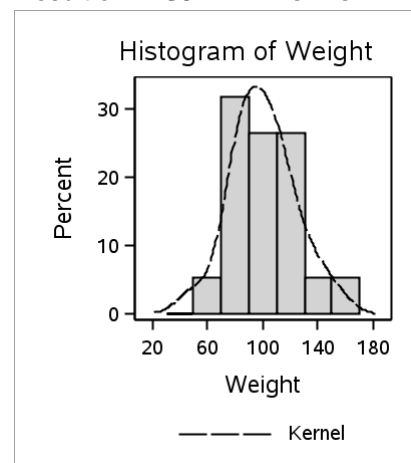
    discretelegend "n" "k";
  endlayout;
end;
run;

data _null_;
  set sashelp.class;
  if _n_=1 then
    call symput("varlabel",trim(vlabel(weight)));
  file print ods=(template="mygraphs.conditional"
                 dynamic=(VAR="weight" BINS=7
                          CURVE="all")
                 );
  put _ods_;
run;
```

Result of CURVE="all"



Result of CURVE="kernel"



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

OTHER CONSIDERATIONS

The final output you produce from graphics templates is controlled by factors independent of the template and dynamic values you may pass to it.

ODS GRAPHICS STATEMENT

This statement is used to modify the environment in which graphics templates are executed.

You use the ODS GRAPHICS statement to control

- whether ods graphics is enabled or not
- type and name of the image created
- the size of the image
- whether mouse-over tool tip information is generated for the HTML destination.

```
ods graphics on < / IMAGEFMT= STATIC | GIF | PNG | JPEG> IMAGENAME="name"  
                HEIGHT= size WIDTH= size  
                STATICMAP=TRUE | FALSE >;  
procedures or data steps
```

```
ods graphics off;
```

For example,

```
ods graphics on / imagefmt=gif imagename='scatter' height=300 width=350;  
data _null_;  
  set sashelp.class;  
  file print ods=(template='mygraphs.scatter');  
  put _ods_;  
run;  
ods graphics off;
```

This way of setting the image size will work only if the template does not contain HEIGHT= and WIDTH= options on the outermost layout – that is, image size declared in the template has precedence.

ODS STYLES

When any graphics or table template is executed there is always an ODS style in effect that governs the appearance of the output. All the output examples in this paper were generated with a style named JOURNAL and the RTF output destination:

```
ods rtf style=journal;  
  
ods graphics on / height=300 width=350;  
data _null_;  
  set sashelp.class;  
  file print ods=(template='mygraphs.scatter');  
  put _ods_;  
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 whenever you change the active style. Shown below are a few of the mappings for GTL statements to corresponding style elements.

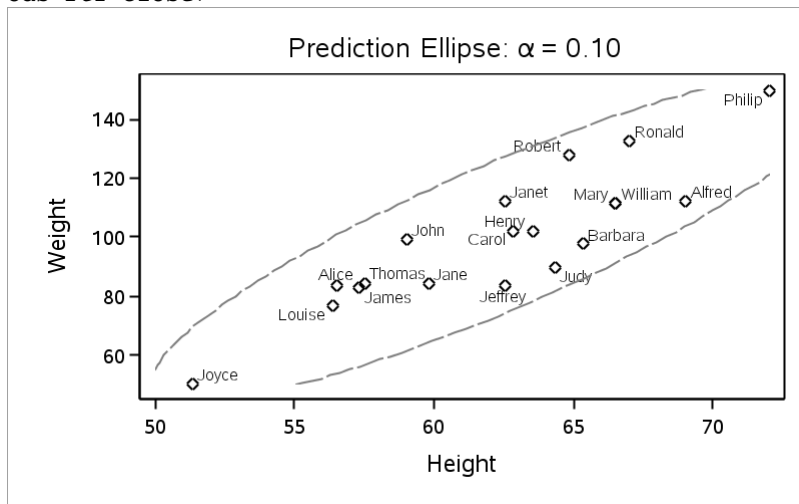
GTL statement	ODS style elements
entrytitle	GraphTitleText
entryfootnote	GraphFootnoteText
boxplot, boxplotparm	GraphBoxWhisker, GraphBoxFarOutlier, GraphBoxConnect, GraphBoxMean, GraphBoxMedian

The documentation includes the default style mapping for all appearance options.

This 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.

```
proc template;
  define statgraph mygraphs.plotstyle;
    dynamic XVAR YVAR CONF IDVAR;
    layout overlay;
    entrytitle "Prediction Ellipse: " {unicode alpha} " = " eval(put(1-CONF,4.2));
    scatterplot x=XVAR y=YVAR /datalabel=IDVAR datalabelattrs=GraphDataText;
    ellipse x=XVAR y=YVAR / clip=true type=predicted confidence=CONF
           outlineattrs=GraphPredictionLines;
  endlayout;
end;
run;

ods rtf style=journal;
ods graphics on / height=250 width=400;
data _null_;
  set sashelp.class;
  file print ods=(template="mygraphs.plotstyle"
                 dynamic=(XVAR="height" YVAR="weight" CONF=.9 IDVAR="name" ));
  put _ods_;
run;
ods graphics off;
ods rtf close;
```



GraphDataText is a style element that defines a set of font attributes including Color, Family, Size, Weight, and Style.

GraphPredictionLines is a style element that defines a set of line attributes including Pattern, Thickness, and Color.

The alternative to setting appearance features via style references is to “hardcode” them:

```
datalabelattrs=(color=black
                family='Arial'
                size=6pt)

outlineattrs=(color=black
              pattern=dash)
```

CONCLUSION

With each release of SAS, more procedures are capable of producing graphical output with template technology. Simple procedure options let you request which graphs are produced. You really need to know nothing about GTL to produce graphs. However, there are two common situations where an understanding of GTL is necessary.

- A procedure produces graphs but you would like to have a slightly different graph generated each time the procedure runs. Explanations and examples of how to do this for the SAS 9.1 release can be found at <http://support.sas.com/rnd/base/topics/statgraph/>. This information will be updated as necessary for the SAS 9.2 production release.
- You would like to produce a graph of your own design. All examples in this paper illustrate this. It should be noted that the GTL code in this paper is for the SAS 9.2 release, and differs somewhat from SAS 9.1 coding. If you have access to SAS 9.1 and would like to experiment with the SAS 9.1 version of the template code presented here, send an email to Jeff.Cartier@sas.com

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.