THE POWER TO KNOW.

SAS/GRAPH® 9.2
Graph Template Language
User's Guide
Second Edition

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2009. *SAS/GRAPH® 9.2: Graph Template Language User's Guide, Second Edition*. Cary, NC: SAS Institute Inc.

SAS/GRAPH® 9.2: Graph Template Language User's Guide, Second Edition

ISBN 978-1-60764-387-6

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, September 2009

1st printing, September 2009

Contents

Chapter 1
ODS Graphics

Overview of ODS Graphics

ODS Graphics is a system for creating graphics that address the following requirements:

- the need for a flexible syntax to create complex graphs
- the need to create high quality graphical output.

Modern analytical graphs are an integral part of an analysis or a study. ODS Graphics gives SAS analytical procedures the ability to create complex analytical graphs that deliver the analysis results with clarity and without clutter. By enabling ODS Graphics, SAS users get the relevant graphs automatically as part of the analysis process. Additionally, they have easy to use tools that can create related graphs for preview of the data or for creating graphs from the results of multiple analyses.

ODS Graphics are driven by the Graph Template Language (GTL) syntax, which provides the power and flexibility to create many complex graphs. Whereas you can use GTL to create your own graphics, its power and flexibility comes with some complexity. For that reason, this document discusses the ways in which the SAS System leverages the power of GTL to create graphics using other tools and systems. You might find that these other tools meet all of your needs.

Automatic Graphics from SAS Analytical Procedures

With SAS 9.2, users can generate ODS graphs automatically from analytical procedures, which can produce the graphs along with the tabular data. These graphs can be produced automatically by turning on ODS Graphics using the following statement:

```
ods graphics on </ options>;
```

After ODS Graphics is switched on, the graphs defined as part of any procedure's output are written to the active ODS destinations. Users can control the specific graphs produced by using the PLOTS= options on the procedure statement or by using the ODS SELECT and ODS EXCLUDE statements. For more information, see the primer and syntax sections in the discussion of statistical graphics using ODS in the *SAS/STAT User's Guide*.

Modifying Templates for Automatic Graphs

The graphs that are produced by the analytical procedures are created from compiled STATGRAPH templates written in GTL. For each graph that is created by a procedure, a template has been defined by the procedure writers and shipped with SAS. These templates can be found in the appropriate sub-folder of the SASHELP.TMPLMST item store (issue the ODSTEMPLATE command to open the Templates window). Users wanting to make persistent changes to these automatic graphics can do so by editing and recompiling these definitions of the graphs. Understanding the structure of these templates requires knowledge of GTL, as described in this User's Guide. Changes to these templates should follow the guidelines presented in the discussion on statistical graphics using ODS in the *SAS/STAT User's Guide*.

The ODS Graphics Editor

After an ODS graph is created, some users might want to edit and/or customize the graphical output for presentation to their audience or for inclusion in other documents. These changes could be something as simple as editing the graph title, or adding a footnote to the graph. Although you could edit the associated template using GTL and then regenerate the graph, you can use the ODS Graphics Editor to make simple, customized changes to the ODS Graphics output. The Editor is an interactive, GUI-based tool that is specifically designed for this purpose. Using this tool, you can

- edit or add titles and footnotes to the graphs
- change graph styles and visual attributes, such as marker shapes, line patterns, colors, and so on
- add free-form text, arrows, lines, and other graph elements to call out various elements of the results.

Changes made to a graph with the ODS Graphics Editor do not affect the template that defined the graph. For more information on the ODS Graphics Editor, please see the *SAS/GRAPH ODS Graphics Editor User's Guide*.

Creating Graphs Using SAS/GRAPH Statistical Graphics Procedures

As seen so far, you can obtain analytical graphs automatically from SAS analytical procedures. Furthermore, you can edit or customize these graphs with the ODS Graphics Editor, all without any need to learn the GTL syntax.

However, frequently you might need to get a better understanding of the data in a study or survey by creating preliminary graphical views of the data. Such views might be necessary before a decision can be made about the detailed analysis process.

Also, your task might require data analysis using multiple procedures and some custom data management. After such analysis process, the results contained in the output data sets might need to be displayed as custom graphs.

Many such graphs can be created using the Statistical Graphics (SG) Procedures. The SG procedures are a set of graphics procedures that leverage the power of GTL behind the scenes to create commonly used graphs using a simple and concise syntax. The following new SG procedures are available with SAS 9.2:

- The SGPLOT procedure for creating single-cell graphs.
- The SGPANEL procedure for creating multi-cell classification panels.
- The SGSCATTER procedure for creating multi-cell comparative scatter plots.

The SG procedures also provide additional data summarization features that are not provided by GTL. For many users, these procedures are the right set of tools to use for meeting their needs, without deploying the full power of GTL.

For more information on the set of SG procedures, see the *SAS/GRAPH Statistical Graphics Procedures Guide*.

The Graph Template Language

Overview

At the heart of ODS Graphics lies the Graph Template Language. For example, all of the graphs that are created by the SAS analytical procedures and by the SAS/GRAPH Statistical Graphics Procedures are generated using GTL. Users who need to go beyond the graphs created by these SAS procedures can use GTL directly to design their graphs using the TEMPLATE and SGRENDER procedures. To successfully create or modify GTL templates, you need the information in this User's Guide, which helps you understand important concepts and offers many complete code examples illustrating often used features. You also need access to the *SAS/GRAPH Graph Template Language Reference*, which is the language dictionary for GTL.

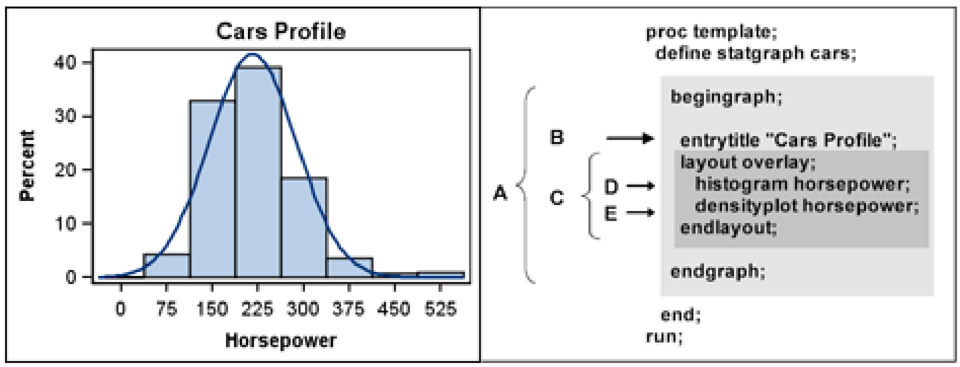
Creating a graph using GTL is a two step process:

1. Define the structure of the graph using the GTL syntax in a STATGRAPH template that is specified on the TEMPLATE procedure. Compile and save this template.
2. Create the graph by running the SGRENDER procedure to associate the appropriate data with the template.

Defining the Graph Template

GTL uses a structured "building-block" approach to defining a graph. The syntax provides a set of layout, plot, and other statements to define the graph. Figure 1.1 on page 4 shows a graph of car profiles by horsepower and the syntax necessary to define the graph using GTL.

Figure 1.1 *ODS Graph and the Template to Generate It*



The template definition consists of the following parts:

A. The GTL syntax block from BEGINGRAPH to ENDGRAPH.

B. The title for the graph.

C. The LAYOUT OVERLAY block. The results of the statements in this block are overlaid in the graph.

D. A histogram of the horsepower variable.

E. A density plot of the horsepower variable.

Running the program in Figure 1.1 on page 4 creates the CARS template and saves it in an item store.

Creating the Graph

To create the graph in Figure 1.1 on page 4, the SGRENDER procedure is run to associate the appropriate data set with the compiled template:

```
proc sgrender data=sashelp.cars template=cars; run;
```

In the template definition, the HORSEPOWER variable is used explicitly for the HISTOGRAM and DENSITYPLOT statements. The explicit reference in the template to a variable named HORSEPOWER requires that the data set have a numeric column named HORSEPOWER. Chapter 14, "Using Dynamics and Macro Variables to Make Flexible Templates," on page 251 shows you how to make the template more flexible.

The GTL syntax supports a variety of layout, plot, and other statements to create a wide range of graphs. Details on all these statements and options are covered in this User's Guide and in the *SAS/GRAPH Graph Template Language Reference*.

When GTL is Needed

- The graphs that are created by the analytical procedures use predefined GTL templates. These templates are designed by SAS procedure writers and shipped with SAS. Every graph created by these procedures has a corresponding template stored in the SASHELP.TMPLMST item store. To customize these templates, you must develop a basic understanding of the GTL.

- Often analysts need graphs of the data before an analysis can be started. Or, the results of a complex analysis involving multiple procedures or DATA steps need to be presented as graphs. Although many of these tasks can be accomplished using the SG Procedures, those procedures do not provide many of the advanced layout capabilities of GTL. To create such custom graphs, you must develop a basic understanding of the GTL.

ODS Graphics and SAS/GRAPH

Beginning in SAS 9.2, the Graph Template Language and ODS Graphics provide new ways of creating graphics. This system is completely independent from the traditional SAS/GRAPH procedures in many significant ways. Some of the architectural differences are as follows:

- GTL has a layout-centric architecture. Each graph comprises components such as plots, insets, and legends that can be combined in flexible ways inside layout containers that can build complex graphs.
- Several layout types are available, some that produce a graph in a single cell and others that produce a graph as a panel of cells. In most cases, the components used in the single-cell graphs can also be used in the multi-cell graphs.
- Axes, backgrounds, titles, legends, and the other components in a graph are managed by the containers and do not belong to an individual plot.
- Global options are specified in the ODS GRAPHICS statement or in the ODS DESTINATION statement. GTL does not use the traditional SAS/GRAPH global statements, such as SYMBOL, PATTERN, AXIS, LEGEND, and GOPTIONS. Also, SAS TITLE and FOOTNOTE statements do not appear in the graph. GTL has its own statements for titles and footnotes. (However, the specialized SGPLOT, SGPANEL, and SGLATTICE procedures are specified with SAS/GRAPH syntax and do accommodate the TITLE and FOOTNOTE statements. They generate GTL behind the scene.)
- A plot statement is available for generating each plot type so that you do not have to specify the plot type by setting an interpolation option on a SYMBOL statement.
- All graphical attributes for markers, lines, color, and so on, are derived by default from the active ODS Style. For more information on ODS styles, see Chapter 6, “Managing Graph Appearance: General Principles,” on page 83.
- GTL supports the use of transparency and anti-aliasing for creating modern graphs.
- The rendering technology for GTL is not based on device drivers. The graphics area is not partitioned in cell-based units. Graphical content gracefully scales up and down as the size of the output is changed.
- Markers, lines, and fonts are scaled using DPI.
- GTL produces all output in industry standard output formats, such as PNG, GIF, JPEG, and so on. GTL does not create GRSEG entries in a catalog.
- The Annotate facility is not supported by the GTL (although you can annotate ODS Graphics output using the ODS Graphics Editor).

Sample of ODS Graphics Output

The following graphs provide a small sample of the diverse output you can produce with ODS Graphics:

Figure 1.2 *PROC LIFETEST (SAS/STAT)*

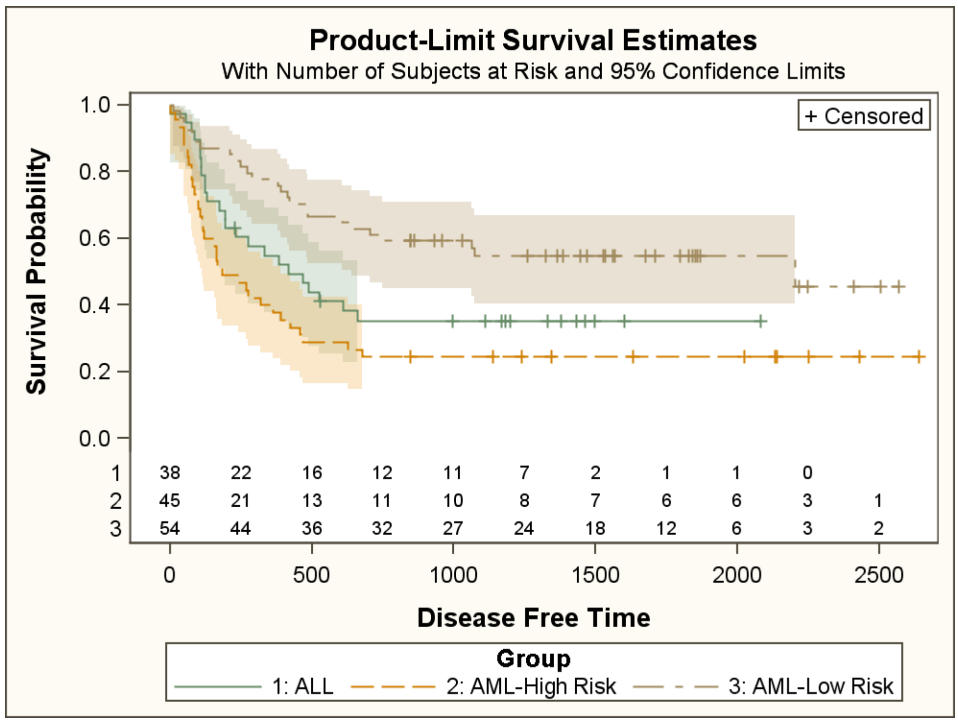


Figure 1.3 *PROC SGSCATTER (SAS/GRAPH)*

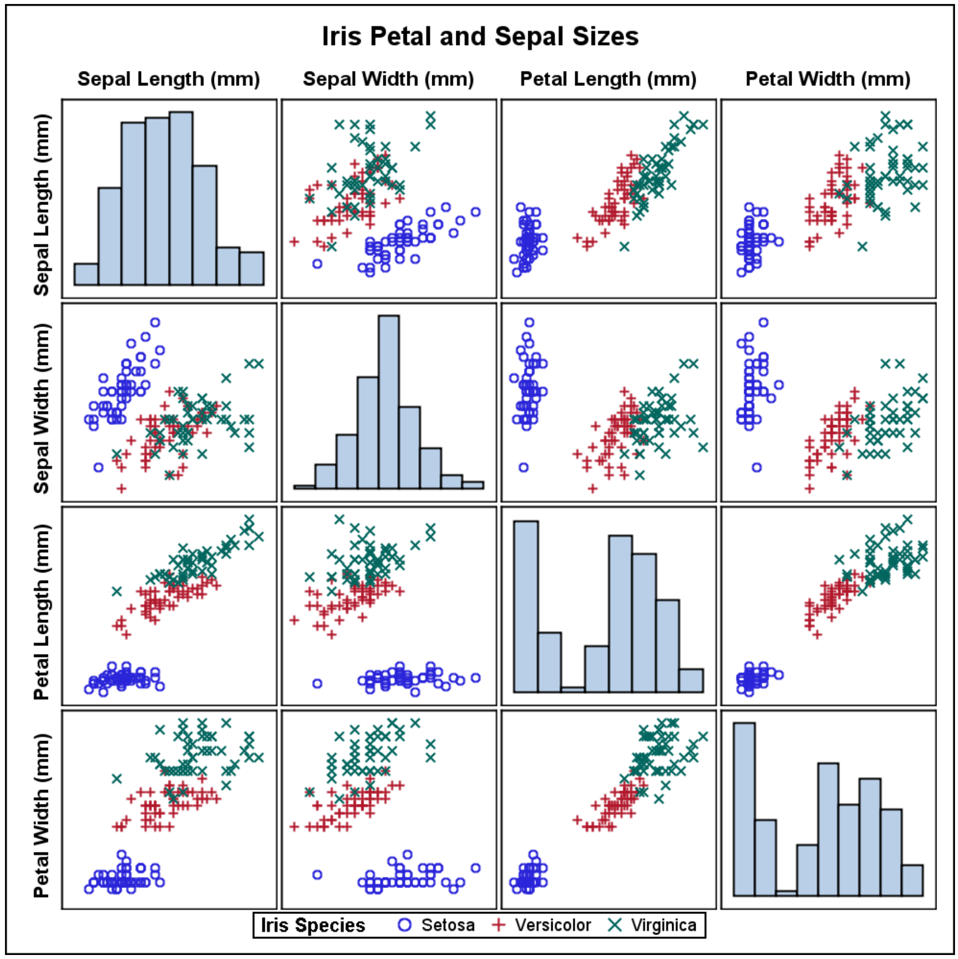
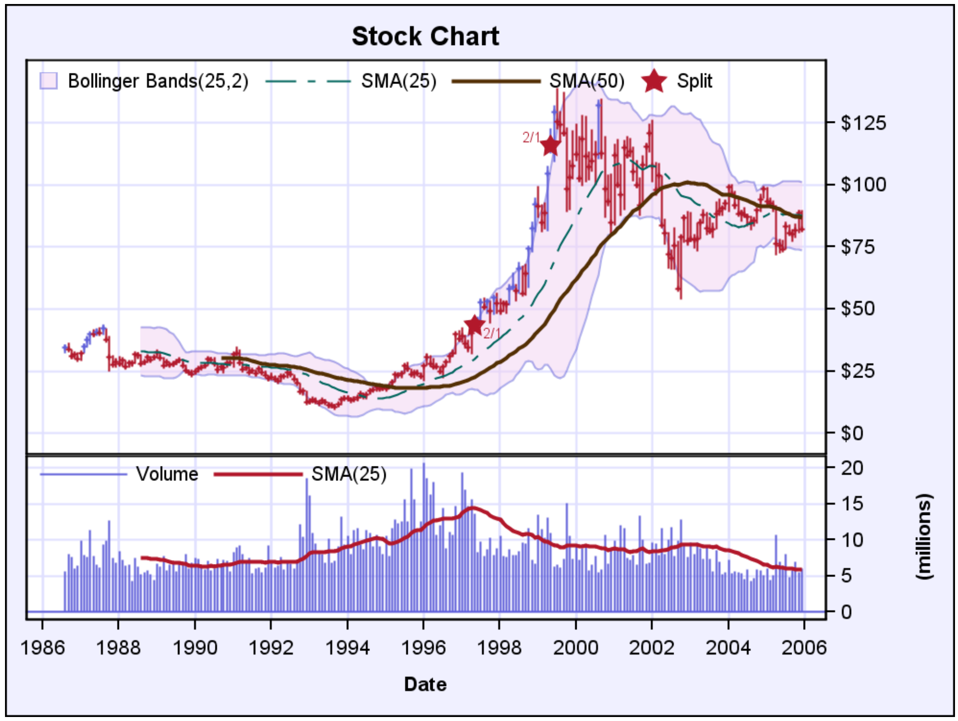


Figure 1.4 *Custom Template Rendered with PROC SGRENDER (SAS/GRAPH)*



Chapter 2
Quick Start

Steps for Creating a Graph Using GTL

This chapter provides an overview of how graphs are created with the GTL and a brief discussion of what is going on behind the scenes. All the examples are completely coded. You can copy and paste these programs into an editor of a SAS Session (beginning in SAS 9.2) and follow the steps as they are described.

As you learned in "Defining the Graph Template" on page 4, creating a graph using GTL is a two-step process:

1. Use PROC TEMPLATE to define a STATGRAPH template with GTL syntax. Compile and save this template.

2. Create the graph by running the SGRENDER procedure to associate the appropriate data with the template.

About the Examples in this Documentation

The programs in this documentation often provide all of the code you need to generate the graphs that are shown in the figures. We encourage you to copy and paste the code into your SAS session and generate the graphs for yourself. The graphs that you generate in the

LISTING destination will be rendered in their default 640 pixel by 480 pixel size (except for those examples that show you how to change the graph size).

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

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

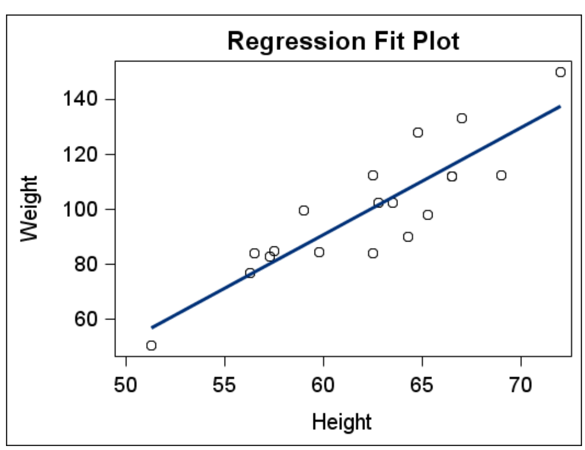
In addition, a custom style was created to further enhance the readability of the smaller graphs. The custom style modifies the supplied LISTING style by further reducing font sizes so that more space is available to the graphical elements in the output, and by making some labeling text bold to enhance the contrast in the graph.

You can use the same techniques when producing your graphical output. The “Managing Your Graphics” and " Controlling Graph Appearance with Modified Styles" chapters explain how to set fonts, DPI, anti-aliasing, and other features that contribute to producing professional-looking graphics of any size in any output format.

Creating a Graph Template

Quick Look at a GTL Graph Definition

To illustrate the steps needed to create a graph, assume that we want to produce a graph showing a linear regression fit for a set of data where HEIGHT is an independent variable and WEIGHT is a dependent variable.




```
proc template;
 define statgraph modelfit;
  begingraph;
   entrytitle "Regression Fit Plot";
   layout overlay;
     scatterplot x=height y=weight;
     regressionplot x=height y=weight;
```

```
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.class
              template=modelfit;
run;
```

You can submit this program to produce the graph. Let us now look in more detail into what this program does.

More Detailed Look at a GTL Graph Definition

The TEMPLATE procedure can produce different kinds of templates, like STYLE, TABLE, COLUMN, and STATGRAPH. The type of template to be created is specified with a DEFINE statement.

define block

```
proc template;
  define statgraph modelfit;
    begingraph;
      entrytitle "Regression Fit Plot";
      layout overlay;
        scatterplot x=height y=weight;
        regressionplot x=height y=weight;
      endlayout;
    endgraph;
  end;
run;
```

The DEFINE STATGRAPH statement and its matching END statement indicate that a graphics template named MODELFIT is to be created. The template name can be a simple one-level name or a multi-level name such as GRAPHS.MODELFIT or PROJECT.STUDY3.MODELFIT indicating a folders where the MODELFIT template is to be stored.

graph block

```
proc template;
  define statgraph modelfit;
    begingraph;
      entrytitle "Regression Fit Plot";
      layout overlay;
        scatterplot x=height y=weight;
        regressionplot x=height y=weight;
      endlayout;
    endgraph;
  end;
run;
```

The BEGINGRAPH statement and its matching ENDGRAPH statement define the outermost container for the graph. It supports options for sizing the graph. Within this block you can use various statements that define the content of the graph.

ENTRYTITLE and ENTRYFOOTNOTE statements can be used to specify graph title lines and graph footnote lines, if needed.

layout block

```
proc template;
  define statgraph modelfit;
    begingraph;
      entrytitle "Regression Fit Plot";
      layout overlay;
        scatterplot x=height y=weight;
        regressionplot x=height y=weight;
      endlayout;
    endgraph;
  end;
run;
```

The LAYOUT OVERLAY statement and its matching ENDLAYOUT statement define the type of graphical layout to be used. The OVERLAY layout allows the contained plots to be overlaid. It manages the plot layers and queries all contained plots to decide the axis types, axis labels, and axis ranges.

graph content

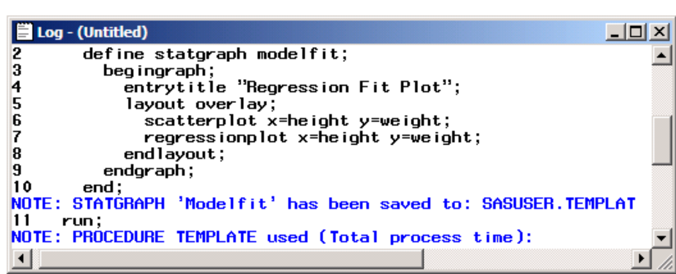
```
proc template;
  define statgraph modelfit;
    begingraph;
      entrytitle "Regression Fit Plot";
      layout overlay;
        scatterplot x=height y=weight;
        regressionplot x=height y=weight;
      endlayout;
    endgraph;
  end;
run;
```

Both the SCATTERPLOT and REGRESSIONPLOT statements specify HEIGHT for the X variable and WEIGHT for the Y variable. For the regression, X is always used for the independent variable and Y for the dependent variable. By default, a linear regression is used.

For more information on the types of layouts and plots in GTL, see Chapter 3, “Overview of Basic Statements and Options,” on page 19.

Compiling the Template

When you submit your PROC TEMPLATE statements, the template syntax is checked.

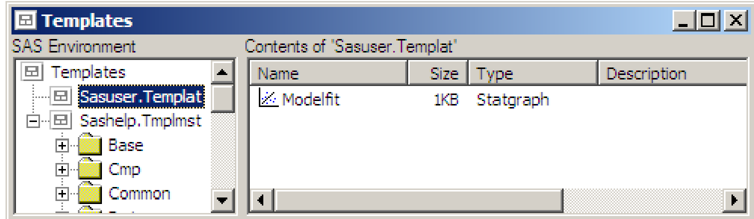

Log - (Untitled)

```
2       define statgraph modelfit;
3         begingraph;
4           entrytitle "Regression Fit Plot";
5           layout overlay;
6             scatterplot x=height y=weight;
7             regressionplot x=height y=weight;
8           endlayout;
9         endgraph;
10      end;
NOTE: STATGRAPH 'Modelfit' has been saved to: SASUSER.TEMPLAT
11    run;
NOTE: PROCEDURE TEMPLATE used (Total process time):
```

If no syntax error is detected, a compiled template named MODELFIT is created and stored physically in the SASUSER.TEMPLAT item store by default. This item store is chosen by default because it is the first item store that can be updated in the current ODS path.

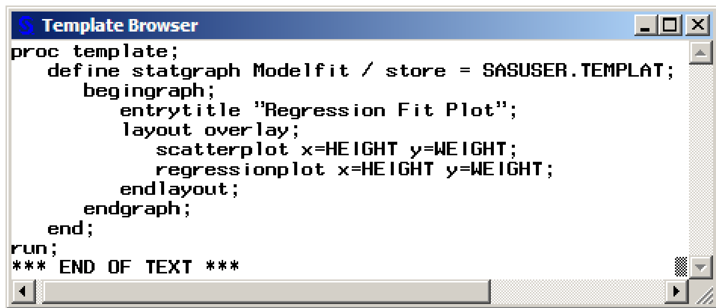
It should be noted that STATGRAPH template syntax requires that any necessary arguments be specified (X= and Y= arguments are required for both the SCATTERPLOT and REGRESSIONPLOT statements), but no checking for the existence of the assigned variables is performed at compile time. Also note that no reference to an input data set appears in the template.

Compiling the template does not produce a graph—it only creates a compiled template that can be executed to produce a graph.



To verify that the template was created, you can issue the ODSTEMPLATE command (ODST, for short). This opens the Templates window where you view all item stores and their contents. All STATGRAPH templates can be identified by the common icon shown above.

You can also browse the source for any compiled template by double-clicking on its name.



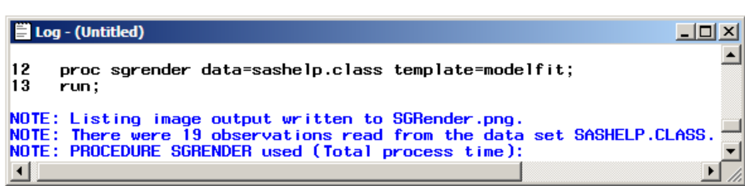
For more information about item stores and PROC TEMPLATE in general, see the *SAS Output Delivery System: User's Guide* in the Base documentation.

Executing the Template to Produce the Graph

To produce a graph, use the SGRENDER procedure:

```
proc sgrender data=sashelp.class template=modelfit; run;
```

The SGRENDER procedure takes two required arguments: DATA= for the input data set and TEMPLATE= for the STATGRAPH template to be used.

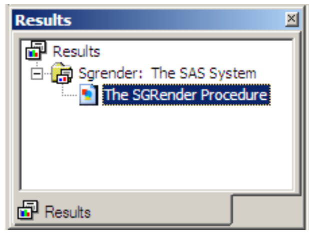


SGRENDER produces the graph by

- building a data object for the template. This data object contains only the requested variables (HEIGHT and WEIGHT) for the scatter points along with any other internally computed values, such as the points on the regression line.
- obtaining default color, line, marker, and font properties from the currently active style.

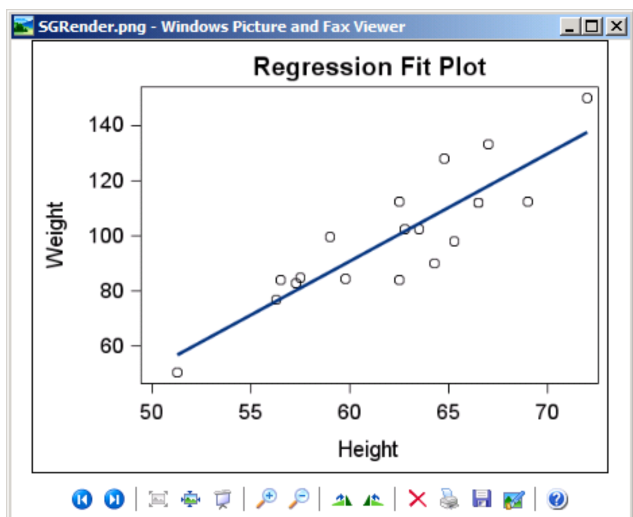
This information, along with the GTL definition of the graph, is then passed to a rendering module that assembles everything and produces an image, which is integrated into the active ODS destination(s).

Minimally, one ODS destination must be open. By default, that destination is LISTING. For this destination, the default is to create an image of type PNG.



Graph output to the LISTING destination is not displayed automatically. To view the output, you can open the Results window (choose **View** ⇨ **Results** from the menu) and select it.

Display 2.1 *Graph that Has Been Opened from the Results Window in MS Windows*



For more information on other features of SGRENDER, see Chapter 18, “Executing Graph Templates,” on page 311.

Managing the Graphical Output

Directing Output to ODS Destinations

All ODS graphics are generated in industry standard formats (PNG, PDF, and so on), depending on the settings for the active ODS destinations. The ODS LISTING destination is on by default, and the default image format for the LISTING destination is PNG.

All ODS destinations such as HTML, PDF, RTF, LATEX, and PRINTER are fully supported. The ODS destinations enable you to

- manage the graphs that are generated by ODS Graphics
- display the output in a variety of forms (HTML, PDF, RTF, ...)
- control the location of stored output files and other features that are relevant for the active destinations.

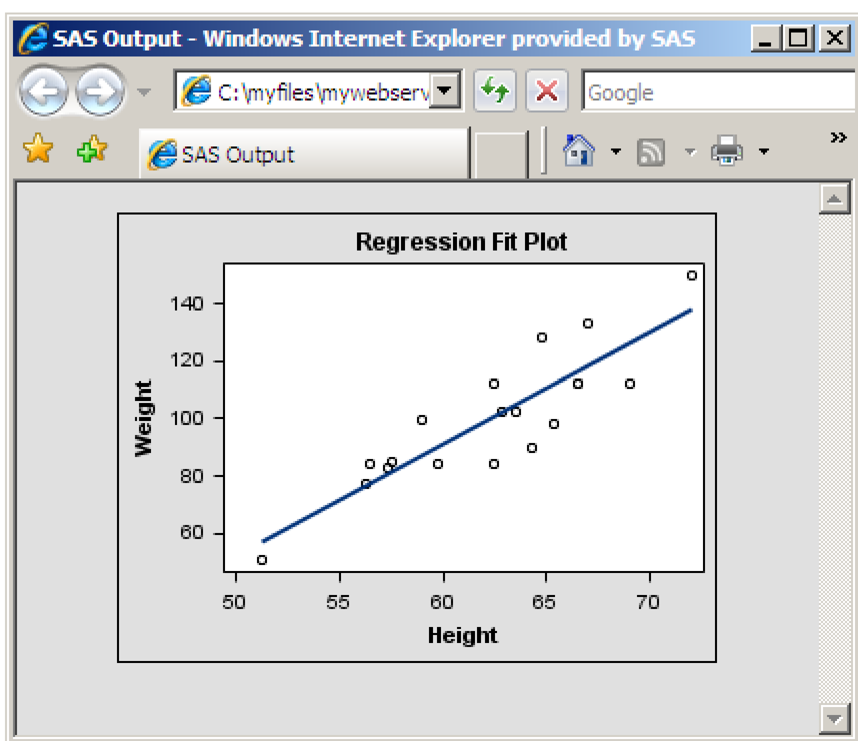
As discussed in "Compiling the Template" on page 12, a compiled template is stored in an item store. Thus, without rewriting or resubmitting the template code, we can render the graph as often as needed during the current SAS session or a future SAS session.

To generate ODS Graphics output for use on the Web, we can direct the output to the HTML destination, which generates an image file for the graph, and also an HTML file that references the image. Thus, output that is generated in the HTML destination is ready for display in a Web browser.

The following ODS HTML statement stores the output files in the folder **`C:\myfiles\mywebserver`**. The code first closes the LISTING destination to avoid creating extra output:

```
ods listing close;
ods html path="C:\myfiles\mywebserver" (url=none) file="modelfit.html" ;
proc sgrender data=sashelp.class template=modelfit; run;
ods html close;
ods listing; /* reopen the listing destination for subsequent output */
```

- The PATH= option specifies storage location **`C:\myfiles\mywebserver`** for output files that are created by the SAS statements, including images from ODS Graphics.
- The FILE= option specifies that SAS output is written to the file **`modelfit.html`**, which is saved in the location specified on PATH=.
- The ODS HTML CLOSE statement closes the HTML destination, which enables you to see your output. By default, the HTML destination uses the DEFAULT style for graphics output ("Modifying Graph Appearance with Styles" on page 16 provides an introduction to ODS styles), which uses a gray background.



See Chapter 19, "Managing Graphical Output," on page 319 for more information about the ODS destinations and the type of output that results from each destination.

Modifying Graph Appearance with Styles

GTL has been designed to be totally integrated with ODS styles.

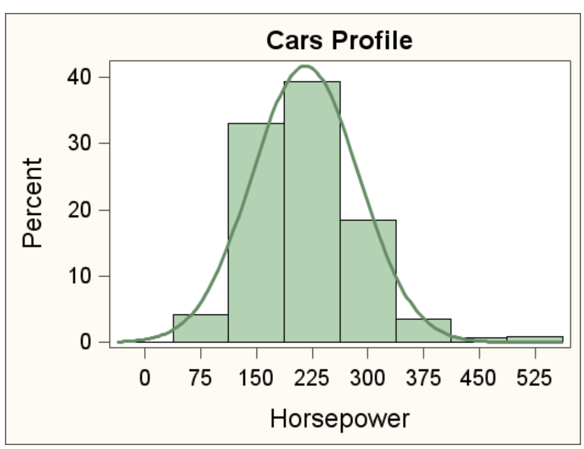
Note: Although every appearance detail of a graph is controlled by the current style by default, you can use GTL syntax options to change the appearance of the graph.

The following template code generates the histogram that was introduced in "Defining the Graph Template" on page 4.

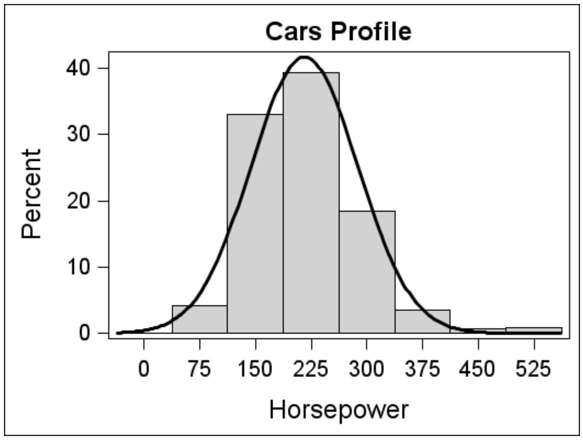
```
proc template;
  define statgraph cars;
    begingraph;
      entrytitle "Cars Profile";
      layout overlay;
        histogram horsepower;
        densityplot horsepower;
      endlayout;
    endgraph;
  end;
run;
```

Every ODS destination has a style that it uses by default. For the listing destination, the default style is LISTING. To modify the appearance of the graph, you can change its style by specifying the STYLE= option in the ODS destination statement before running the SGRENDER procedure:

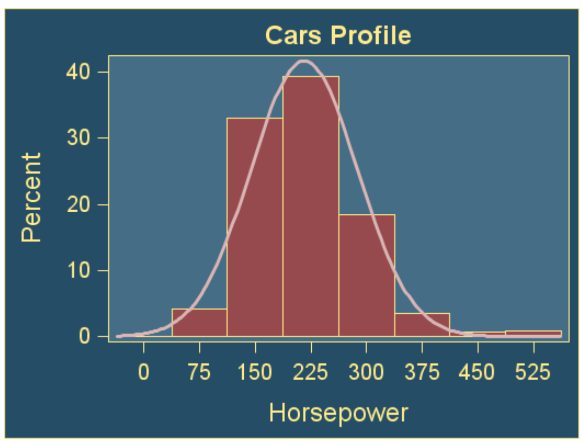
```
ods listing style=analysis ;
proc sgrender data=sashelp.cars template=cars;
run;
```




```
ods listing style=journal ;
proc sgrender data=sashelp.cars template=cars;
run;
```




```
ods listing style=astronomy ;
proc sgrender data=sashelp.cars template=cars;
run;
```



For more information on how the appearance of the graph is determined and the ways you can modify it, see Chapter 6, “Managing Graph Appearance: General Principles,” on page 83 and Chapter 17, “Managing the Graph Appearance with Styles,” on page 299.

Controlling Physical Aspects of the Output

The ODS GRAPHICS statement provides options that control the physical aspects of your graphs, such as the graph size and the name of the output image file.

The LISTING destination's default image size of 640 pixels by 480 pixels (4:3 aspect ratio) for ODS Graphics is set in the SAS Registry. You can change the graph size using the ODS GRAPHICS statement’s WIDTH= and/or HEIGHT= options. To name the output image file, use the IMAGENAME= option.

The following ODS GRAPHICS statement sets a 320 pixel width for the graph and names the output image *modelfitgraph*:

```
ods graphics / width=320px imagename="modelfitgraph" ;

proc sgrender data=sashelp.class template=modelfit;
run;

ods graphics / reset ;
```

- The WIDTH= option sets the image width to 320 pixels. Because no HEIGHT= option is used, SAS uses the design aspect ratio of the graph to compute the appropriate height. (The width of 320px is half the default width, so SAS will set the height to 240px, which is half the default height.) In general, it is good practice to specify only one sizing option without the other — just the WIDTH= option or just the HEIGHT= option. That way SAS will maintain the design aspect ratio of the graph, which might be important for many graphs.
- The IMAGENAME= option in the first ODS GRAPHICS statement sets the name of the output image file to *modelfitgraph*.
- The RESET option in the second ODS GRAPHICS statement resets all ODS GRAPHICS options to their default state. If the options are not reset, all subsequent graphs would be 320 pixels wide and image names would be assigned incremental names (modelfitgraph1, modelfitgraph2, and so on) every time a graph is produced.

For more information on the details of managing image name, image size, image format, and DPI., see Chapter 19, “Managing Graphical Output,” on page 319.

Chapter 3
Overview of Basic Statements and Options

Introduction to GTL Statements

GTL encompasses a large number of statements and options. This chapter provides an organizational framework to help you think about the language. Just as you can think of the SAS language syntax in terms of Statements, Functions, Formats, and System options, you can apply a classification scheme to GTL. A general understanding of the GTL helps you write your templates with more confidence and efficiency. Some of the terminology introduced here will appear often in other chapters.

Statements

All GTL statements have the following syntax:

KEYWORD(s) *required argument(s)* < / *option(s)*>

Examples:

```
/* This statement uses two keywords, no required arguments,
          and no options */
LAYOUT OVERLAY;

/* This statement uses one keyword and two required arguments */
SCATTERPLOT X=height Y=weight;

/* This statement specifies a required argument.
   Required arguments do not have to be name-value pairs. */
HISTOGRAM weight;

/* This statement uses one option.
   Options are specified after a slash (/) and are usually
   name-value pairs. */
SCATTERPLOT X=height Y=weight / GROUP=age;
```

Blocks

A block is a pair of statements that indicate the beginning and end of a syntax unit. Typically, other statements are nested within the block. GTL has many specialized block constructs.

Examples:

```
/* This is a valid block. No nested statements are required. */
LAYOUT OVERLAY;
ENDLAYOUT;

/* This block has no restrictions on the number of nested statements. */
LAYOUT OVERLAY;
  SCATTERPLOT X=height Y=weight;
  REGRESSIONPLOT X=height Y=weight;
ENDLAYOUT;

/* This block allows only nested ROWAXIS statements. */
ROWAXES;
  ROWAXIS / LABEL="Row 1";
  ROWAXIS / LABEL="Row 2";
ENDROWAXES;

/* Blocks support nested blocks */
CELL;
  CELLHEADER;
    ENTRY "Cell 1";
  ENDCELLHEADER;
  LAYOUT OVERLAY;
    HISTOGRAM weight;
    DENSITYPLOT weight;
  ENDLAYOUT;
ENDCELL;
```

Whenever blocks are nested, there exists a "Parent - Child" relationship. In the previous example, the CELL block is the parent of the CELLHEADER block and LAYOUT OVERLAY block. This is important because most blocks have rules about what statements they might contain, and they also have nesting restrictions. For example, a CELLHEADER block, if used, must be the direct child of a CELL block. Only one CELLHEADER block can be used per CELL block. To improve code readability, nested blocks are indented in source programs.

Categories of Statements

Overview

GTL statements generally fall into two main categories:

- Plot, Legend, and Text statements that determine what items are drawn in the graph.
- Layout statements that determine how or where the items in the graphs are placed.

Plot Statements—Terminology and Concepts

Overview

GTL has numerous plot statements that can be combined with one another in many different ways. In future releases of GTL, new layout and plot statements will be added to supplement those now available. GTL has been designed as a high-level toolkit that enables you to create a large variety of graphs by combining its constructs in different ways. As you might imagine, not all combinations of statements are possible, and most of the invalid combinations are caught during template compilation. Rather than trying to create graphs by trial and error, it is recommended that you understand a few basic "rules of assembly" to guide your efforts and make the language easier to work with. To that end, some new terminology is useful.

Plot Terminology

Computed Plots

Computed plots internally perform computational transformations on the input data and, as necessary, add new columns to a data object in order to render the requested plot. For example, a LOESSPLOT requires two numeric columns of raw input data (X=*column* and Y=*column*). A loess fit line is computed for these input point pairs, a new set of points on a fit line is generated, and a new column that contains the computed points is added to the data object. A smoothed line is drawn through the computed points. Most computed plots have several options to control the computation performed. Another form of computed plot is one with user-defined data transformations. For example, you can use an EVAL( ) function to compute a new column such as **`Y= eval(log10(`*`column`*`))`**. This transforms *column* values into corresponding logarithmic values. Why is it important to know whether a plot is computed? Certain layouts such as PROTOTYPE currently do not allow computed plots to be included.

Parameterized Plots

Parameterized plots simply render the input data they are given. They are useful whenever you have input data that does not need to be preprocessed or that has already been summarized (possibly an output data set from a procedure like PROC FREQ). For example, BARCHARTPARM draws one bar per input observation: the X= *column* provides the bar tick value and the Y=*column* provides the bar length. So a bar chart with five bars requires a data set with five observations and two variables. A parameterized bar chart statement is useful when the computed BARCHART statement does not perform the type of computation you want, and you have done the summarization yourself. Many parameterized plots have a "PARM" suffix added to their name. Another common situation is when you want to draw a fit line and a confidence band from a set of data that already has the appropriate set of (X,Y) point coordinates. For these situations you would use a SERIESPLOT statement for the fit line and a BANDPLOT statement for the confidence band. Why is it important to know whether a plot is parameterized? Parameterized plots ensure that no additional computation will take place on the input data. Thus, input data that does not meet the special requirements on the parameterized plot might result in bad output or a blank graph.

Stand-alone Plots

A stand-alone plot is one that can be drawn without any other accompanying plot. In general, a plot is stand-alone if its input data defines a range of values for all axes that are needed to display the plot. For example, the observations plotted in a SCATTERPLOT normally span a certain data range in both X and Y axes. This information is necessary to successfully draw the axes and the markers. Why is it important to know which plots are stand-alone? Because most layouts need to know the extents of the X and Y axis to draw the plot.

Dependent Plots

A dependent plot is one that, by itself, does not provide enough information for the axes that are needed to successfully draw the plot. For example, the REFERENCELINE statement draws a straight line perpendicular to one axis at a given input point on the same axis. Because there is only one point provided, there is not enough information to determine the full range of data for this axis. Furthermore, no information is provided for the data range of the second axis. Thus, a REFERENCELINE statement does not provide enough information by itself to draw the axes and the plot. Such a plot needs to work with another "Stand-alone" plot, which will provide the necessary information to determine the data extents of the two axes.

Primary Plot

When you overlay two or more plots, the layout container determines the type of axis to use, the data range of all axes, and the default format and label to use for each axis. By default, the first encountered stand-alone plot is used to decide the axis type and axis format and label. In some cases, you desire a certain overlay stacking and must order your statements accordingly. This might result in undesirable axis properties. By adding the PRIMARY=TRUE option to a stand-alone plot, you can request that this plot be used to determine axis type and axis format and label. A dependent plot cannot be designated as primary.

Graphics Types

GTL supports both 2D and 3D graphics. Currently there are only two 3D plot statements (SURFACEPLOTPARM and BIHISTOGRAM3DPARM). 3D plot statements must be used in a 3D layout. 2D plot statements cannot be used in a 3D layout, and 3D plot statements cannot be used in a 2D layout. For more information on layouts, see "Layout Containers" on page 30.

Plot Statements Categorized by Type

Plot statements are generally categorized as stand-alone or dependent, computed or parameterized, and 2D or 3D. The following tables show the distribution of plots in these categories.

Table 3.1 *Stand-alone, 2D, Computed Plots*

2D PLOTS: COMPUTED		
Statement	**Required Arguments**	**Comments**
BARCHART	One *column*	Horizontal or vertical.
BOXPLOT	One *numeric-column*	Horizontal or vertical.
HISTOGRAM	One *numeric-column*	Horizontal or vertical.
DENSITYPLOT	One *numeric-column*	Theoretical distribution curve (for example, NORMAL or KDE).
REGRESSIONPLOT	Two *numeric-columns*	Fit plot using linear, quadratic, or cubic regression.
LOESSPLOT	Two *numeric-columns*	Fit plot using loess.
PBSPLINEPLOT	Two *numeric-columns*	Fit plot using Penalized B-spline.
ELLIPSE	Two *numeric-columns*	Confidence or prediction ellipse for a set of points.
SCATTERPLOTMATRIX	Two or more *numeric-columns*	Grid of scatter plots. Might include computed ellipses, histograms, density curves.

Table 3.2 *Stand-alone, 2D, Parameterized Plots*

2D PLOTS: NONCOMPUTED / PARAMETERIZED		
Statement	**Required Arguments**	**Comments**
BANDPLOT	Three *columns*, at least two numeric limits	Area bounded by two straight or curved lines.
BARCHARTPARM	Two *columns*, Y must be numeric	Horizontal or vertical. Summarized data provided by user.
BLOCKPLOT	Two *columns*	Strip of X- axis aligned rectangular blocks containing text. The X data must be sorted.
BOXPLOTPARM	One *numeric-column* and one *string-column*	Horizontal or vertical. Needs special data format.
CONTOURPLOTPARM	Three *numeric-columns*	Draws contour plot from pre-gridded data. Basic "gridding" feature is provided using an option.
ELLIPSEPARM	Five *numbers* or *numeric-columns*	Draws ellipse given center, slope, semi-major and semi-minor axis lengths.

2D PLOTS: NONCOMPUTED / PARAMETERIZED		
Statement	**Required Arguments**	**Comments**
FRINGEPLOT	One *numeric-column*	Draws a short line segment of equal length along the X or X2 axis for each observation's X value.
HISTOGRAMPARM	*Two numeric-columns*	Horizontal or vertical. The Y data must be non-negative.
NEEDLEPLOT	Two *columns*, Y must be numeric	Draws parallel, vertical line segments connecting data points to a baseline.
SCATTERPLOT	Two *columns*	Draws markers at data point locations.
SERIESPLOT	Two *columns*	Draws line segments to connect a set of data points.
STEPPLOT	Two *columns*, Y must be numeric	Draws stepped line segments to connect a set of data points.
VECTORPLOT	At least two and up to four *numeric-columns*, X and Y origins can be numeric constants.	Creates directed line segment(s) based on pairs of data points.

Table 3.3 *Stand-alone, 3D, Parameterized Plots*

3D PLOTS: NONCOMPUTED / PARAMETERIZED		
Statement	**Required Arguments**	**Comments**
SURFACEPLOTPARM	Three *numeric-columns*	Smooth surface.
BIHISTOGRAM3DPARM	Three *numeric-columns*	Bivariate histogram. The Z data must be non-negative.

Table 3.4 *Dependent Plots*

Statement	**Required Arguments**	**Comments**
MODELBAND	CLM or CLI name of associated fit plot	Confidence bands. Used only in conjunction with a fit plot.
DROPLINE	(X,Y) point location, two *columns*, or one value and one *column*	Draws a perpendicular line from a data point to a specified axis.

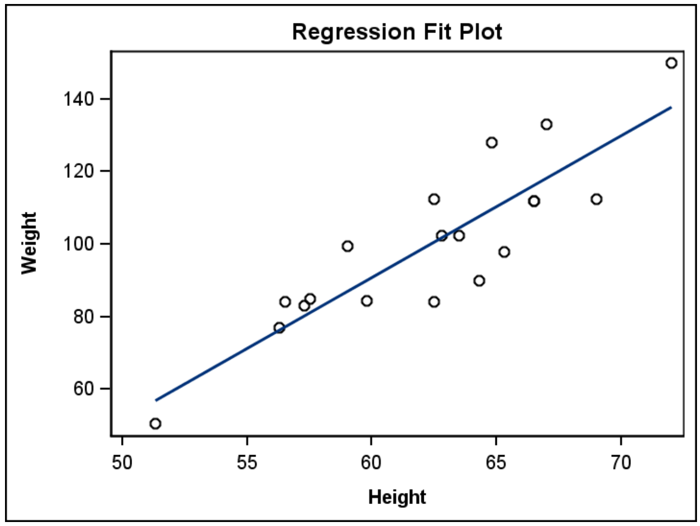
Statement	Required Arguments	Comments
LINEPARM	(X,Y) point location and *slope*. The three values can be provided in any combination of *number* and *numeric-column*	Draws line(s) given a data point and the slope of the line.
REFERENCELINE	X or Y location, *column*	Draws line(s) perpendicular to an axis.

Plot Concepts

To illustrate the use of the different types of plot statements, consider the following template. In this template, named MODELFIT, a SCATTERPLOT is overlaid with a REGRESSIONPLOT. The REGRESSIONPLOT is a computed plot because it takes the input columns (HEIGHT and WEIGHT) and transforms them into two new columns that correspond to points on the requested fit line. By default, a linear regression (DEGREE=1) is performed with other statistical defaults. The model in this case is WEIGHT=HEIGHT, which in the plot statement is specified with **`X=HEIGHT`** (independent variable) and **`Y=WEIGHT`** (dependent variable). The number of observations generated for the fit line is around 200 by default.

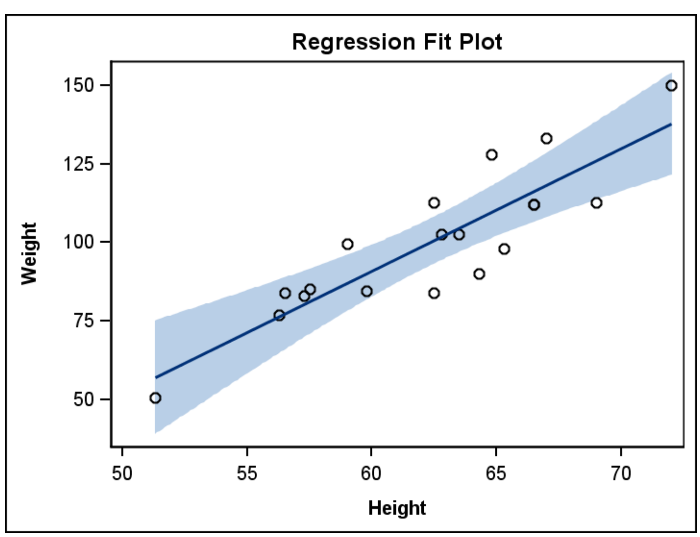
Note: Plot statements have to be used in conjunction with Layout statements. To simplify our discussion, we will continue using the most basic layout statement: LAYOUT OVERLAY. This layout statement acts as a single container for all plot statements placed within it. Every plot is drawn on top of the previous one in the order that the plot statements are specified, with the last one drawn on top.

```
proc template;
  define statgraph modelfit;
    begingraph;
    entrytitle "Regression Fit Plot";
      layout overlay;
        scatterplot x=height y=weight /
                    primary=true;
        regressionplot x=height y=weight;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.class
              template=modelfit;
run;
```



The REGRESSIONPLOT statement can also generate sets of points for the upper and lower confidence limits of the mean (CLM), and for the upper and lower confidence limits of individual predicted values (CLI) for each observation. The CLM="*name*" and CLI="*name*" options cause the extra computation. However, the confidence limits are not displayed by the regression plot. Instead, you must use the dependent plot statement MODELBAND, with the unique name as its required argument. Notice that the MODELBAND statement appears first in the template, ensuring that the band will appear behind the scatter points and fit line. A MODELBAND statement must be used in conjunction with a REGRESSIONPLOT, LOESSPLOT, or PBSPLINEPLOT statement.

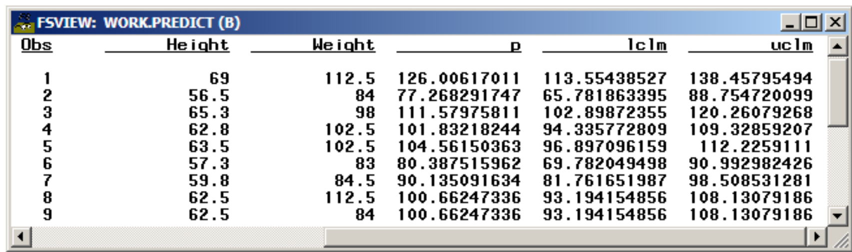
```
layout overlay;
  modelband "myclm" ;
  scatterplot x=height y=weight /
    primary=true;
  regressionplot x=height y=weight /
    alpha=.01 clm="myclm" ;
endlayout;
```



This is certainly the easiest way to construct this type of plot. However, you might want to construct a similar plot from an analysis by a statistical procedure that has many more options for controlling the fit. Most procedures create output data sets that can be used directly to create the plot you want. Here is an example of using non-computed, stand-alone plots to build the fit plot. First choose a procedure to do the analysis.

```
proc reg data=sashelp.class noprint;
  model weight=height / alpha=.01;
  output out=predict predicted=p lclm=lclm uclm=uclm;
run; quit;
```

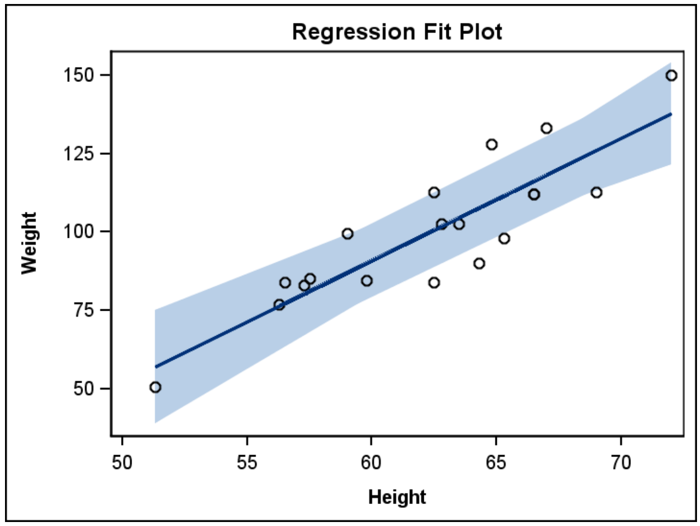
The output data set, PREDICT, contains all the variables and observations in SASHELP.CLASS plus, for each observation, the computed variables P, LCLM, and UCLM.


FSVIEW: WORK.PREDICT (B)

Obs	Height	Weight	p	lclm	uclm
1	69	112.5	126.00617011	113.55438527	138.45795494
2	56.5	84	77.268291747	65.781863395	88.754720099
3	65.3	98	111.57975811	102.89872355	120.26079268
4	62.8	102.5	101.83218244	94.335772809	109.32859207
5	63.5	102.5	104.56150363	96.897096159	112.2259111
6	57.3	83	80.387515962	69.782049498	90.992982426
7	59.8	84.5	90.135091634	81.761651987	98.508531281
8	62.5	112.5	100.66247336	93.194154856	108.13079186
9	62.5	84	100.66247336	93.194154856	108.13079186

Now the template can use simple, non-computed SERIESPLOT and BANDPLOT statements for the presentation of fit line and confidence bands.

```
proc template;
  define statgraph fit;
    begingraph;
      entrytitle "Regression Fit Plot";
      layout overlay;
        bandplot x=height
          limitupper=uclm
          limitlower=lclm /
          fillattrs=GraphConfidence;
        scatterplot x=height y=weight /
          primary=true;
        seriesplot x=height y=p /
          lineattrs=GraphFit;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=predict template=fit;
run;
```

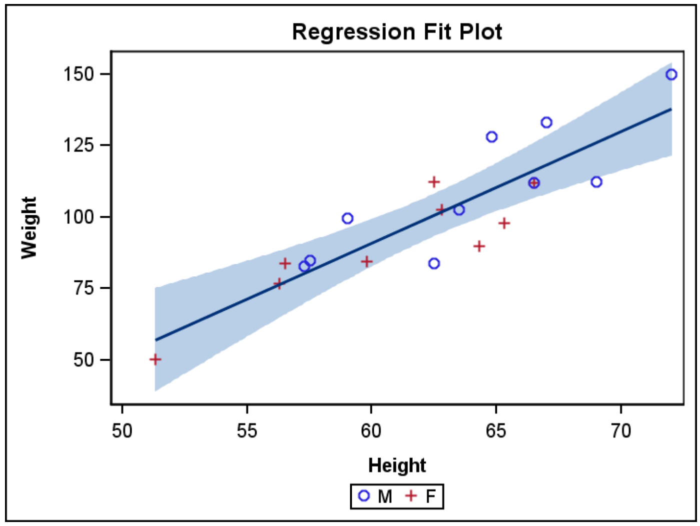


Legend Statements

GTL supports two types of legends: a discrete legend that is used to identify graphical features such as grouped markers, lines, or overlaid plots; and a continuous legend that shows the range of numeric variation as a ramp of color values. Legend statements are dependent on one or more plot statements and must be associated with the plot(s) that they describe. The basic strategy for creating legends is to "link" the plot statement(s) to a legend statement by assigning a unique, case-sensitive name to the plot statement on its NAME= option and then referencing that name on the legend statement.

Statement	Required Arguments	Comments
DISCRETELEGEND	Name(s) of associated plot(s)	Traditional legend with entries for grouped markers/lines or overlaid plots.
CONTINUOUSLEGEND	Name of an associated plot	Shows a numeric scales with a color ramp. Used in conjunction with contours, surfaces, and scatter plots.

```
layout overlay;
  modelband "clm";
  scatterplot x=height y=weight /
    primary=true
    group=sex name="s" ; /* the name is case-sensitive */
  regressionplot x=height y=weight /
    alpha=.01 clm="clm";
  discretelegend "s" ;   /* case must match the case on NAME= */
endlayout;
```



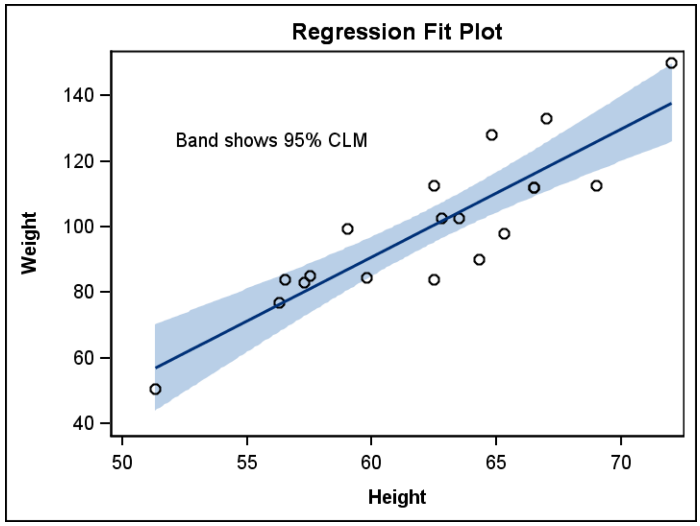
For more information, see Chapter 8, "Adding Legends to a Graph," on page 117.

Text Statements

GTL supports statements that add text to predefined locations of the graph. SAS Title and Footnotes statements do not contribute to the graph. However, there are comparable ENTRYTITLE and ENTRYFOOTNOTE statements. Like Title and Footnote statements, multiple instances of these statements can be used to create multi-line text.

Statement	Required Arguments	Comments
ENTRYTITLE	String	Text to appear above graph. The ENTRYTITLE statement is specified inside the BEGINGRAPH block but outside of the outermost layout.
ENTRYFOOTNOTE	String	Text to appear below graph. The ENTRYFOOTNOTE statement is specified inside the BEGINGRAPH block but outside of the outermost layout.
ENTRY	String	Text to appear within graph. The ENTRY statement is specified inside a layout block.

```
layout overlay;
  modelband "clm";
  scatterplot x=height y=weight /
    primary=true;
  regressionplot x=height y=weight /
    alpha=.05 clm="clm";
  entry "Band shows 95% CLM" /
    autoalign=auto;
endlayout;
```



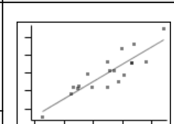
For more information, seeChapter 7, “Adding and Changing Text in a Graph,” on page 101.

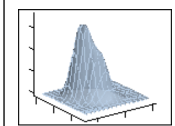
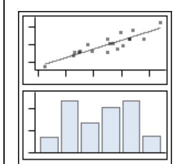
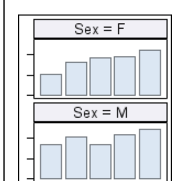
Layout Containers

Layout statements, a key feature of the GTL, form "containers" that determine how the plots, legends and texts items are drawn in the graph. GTL supports many different layout statements that are suitable for different usage. However, these fall into two main categories.

- Single-cell layout statements place the plots, legends, and entries in a common region. The statements that are placed within these "overlay" containers are processed in order. Each plot is drawn on top of the previous plot, with the last one drawn on top.
- Multi-cell layout statements partition the graph region into multiple smaller "cells." Each cell can be populated by an individual plot, an overlay, or a nested multi-cell layout. The layout of the "cells" is determined by the user, or by classification variables.

Layout blocks always begin with the LAYOUT keyword followed by a keyword indicating the purpose of the layout. All layout blocks end with an ENDLAYOUT statement. The following table summarizes the available layouts.

Layout (Description)	Graphics Allowed and Cells Produced	Comments	Example
OVERLAY (Single Cell)	2D One cell	General purpose layout for superimposing 2D plots	
OVERLAYEQUATED (Single Cell)	2D One cell	Specialized OVERLAY with equated axes	
PROTOTYPE (Single Cell)	2D One cell	Specialized LAYOUT used only as child layout of DATAPANEL or DATALATTICE	

Layout (Description)	**Graphics Allowed and Cells Produced**	**Comments**	**Example**
OVERLAY3D (Single Cell)	3D One cell	General purpose 3D layout for superimposing 3D plots.	
LATTICE (Advanced Multi-cell)	2D or 3D One or more cells	All cells must be predefined. Axes can be shared across columns or rows, and they can be external to the grid. Many grid labeling and alignment features.	
GRIDDED (Simple Multi-cell)	2D or 3D One or more cells	All cells must be predefined. Axes independent for each cell. Very simple multi-cell container.	
DATAPANEL (Classification Panel)	2D One or more cells	Displays a panel of similar graphs based on data subsetted by classification variable(s). Number of cells is based on crossings of *n* classification variable(s).	
DATALATTICE (Classification Panel)	2D One or more cells	Displays a panel of similar graphs based on data subsetted by classification variable(s). Number of cells is based on crossings of one or two classification variables.	

To learn more about layouts, refer to the appropriate chapter:

- Chapter 4, "Using a Simple Single-cell Layout," on page 39 (OVERLAY)
- Chapter 12, "Using an Equated Layout," on page 225 (OVERLAYEQUATED)
- Chapter 13, "Using 3D Graphics," on page 233 (OVERLAY3D)
- Chapter 9, "Using a Simple Multi-cell Layout," on page 143 (GRIDDED)
- Chapter 10, "Using an Advanced Multi-cell Layout," on page 155 (LATTICE)
- Chapter 11, "Using Classification Panels," on page 185 (DATAPANEL, DATALATTICE, PROTOTYPE)

Features Supported by Layout, Legend, and Text Statements

All layout, legend, and text statements have a general set of features that include those listed in the following tables. For more information on these and other options, see the chapters specific to the layouts, text statements, and legends. Also see the *SAS/GRAPH: Graph Template Language Reference*.

Backgrounds

OPAQUE= FALSE | TRUE	Whether the background is transparent or not. By default, OPAQUE=FALSE
BACKGROUNDCOLOR= *color*	If the background is opaque, a color can be assigned to it.

Borders

BORDER= FALSE | TRUE	Whether a border is displayed. By default, BORDER=FALSE.
BORDERATTRS= (*line-options*)	If the border is displayed, its line properties can be set.

Padding

PAD= *number*	Whether extra space is added inside the border. By default, layouts and legends have PAD=0, while text statements have PAD=(LEFT=3px RIGHT=3px) as the default.
PAD=(<TOP=*number*> <BOTTOM=*number*> <LEFT=*number*> <RIGHT=*number*>	

Positioning

HALIGN= LEFT | CENTER | RIGHT	When a layout or legend is nested inside an overlay-type layout, or for ENTRY statements, a position can be specified relative to the container. ENTRYTITLE and ENTRYFOOTNOTE statements have fixed vertical positions but can be adjusted horizontally.
VALIGN= TOP | CENTER | BOTTOM	

Features Supported by Many Plot Statements

Plot Features to Be Displayed

All plots have a standard set of features to display. Most plots can show a different feature set. For example, a HISTOGRAM can display bars that are outlined , filled, or both outlined and filled. A SERIESPLOT displays a line and, if requested, point markers.

DISPLAY=(*feature* …)	Specifies the plot features to be displayed. Features are plot specific.

Plot Appearance

Depending on the display features, there are options to control the appearance of the features.

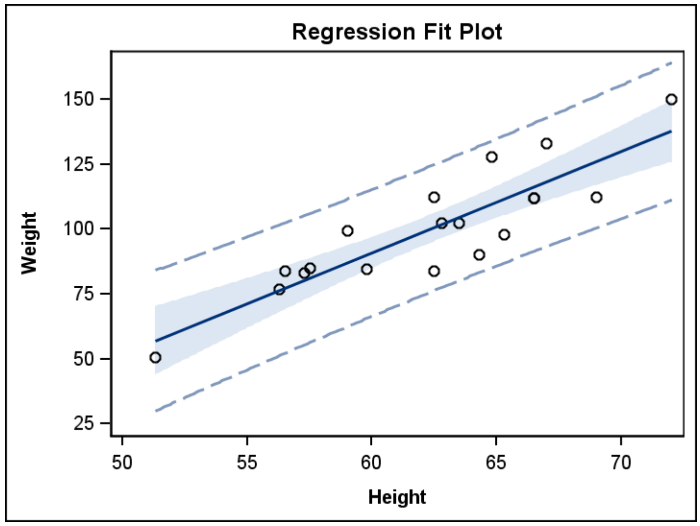
MARKERATTRS=(*marker-options*)	Specifies the symbol, size, color, and weight of markers.
LINEATTRS= (*line-options*)	Specifies the pattern, thickness, and color of lines.
TEXTATTRS= (*text-options*)	Specifies the text color, font, font size, font weight, and font style.
FILLATTRS= (*fill-options*)	Specifies the fill color and transparency.

Plot Transparency

Transparency can be applied to plots that display markers, lines, or filled areas.

DATATRANSPARENCY= *number*	Specifies the degree of transparency. Default is 0 (fully opaque). 1 is fully transparent.

```
layout overlay;
  modelband "cli" / display=(outline)
    outlineattrs=GraphPrediction
    datatransparency=.5 ;
  modelband "clm" / display=(fill)
    fillattrs=GraphConfidence
    datatransparency=.5 ;
  scatterplot x=height y=weight /
    primary=true;
  regressionplot x=height y=weight /
    alpha=.05 clm="clm" cli="cli"; endlayout;
```



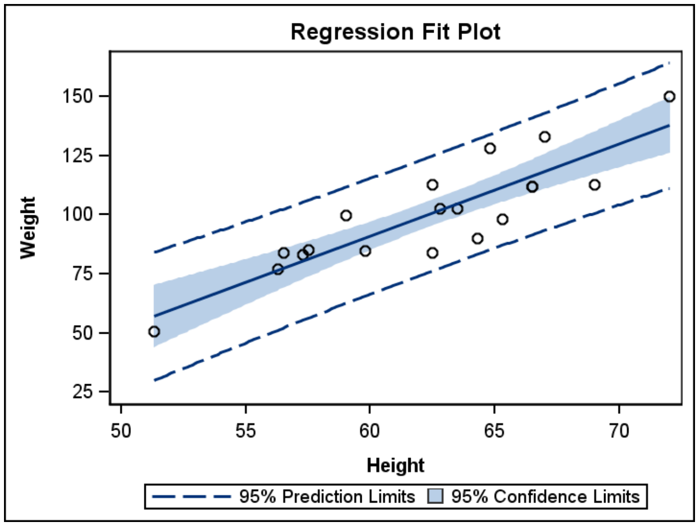
For more information, see Chapter 6, "Managing Graph Appearance: General Principles," on page 83.

Plot Identification

In GTL, legends and some dependent plots (for example, MODELBAND) require a reference (association) with a plot. The association is established by 1) naming the plot, and 2) referring to the plot name within the legend or dependent plot statement.

NAME= "*string*"	Specifies a unique name for a plot in order to associate it with another statement.
LEGENDLABEL= "*string*"	Specifies a description of a plot to appear in a legend.

```
layout overlay;
  modelband "cli" / display=(outline)
    outlineattrs=GraphPrediction
    name="predict"
    legendlabel="95% Prediction Limits" ;
  modelband "clm" / display=(fill)
    fillattrs=GraphConfidence
    name="conf"
    legendlabel="95% Confidence Limits" ;
  scatterplot x=height y=weight /
    primary=true;
  regressionplot x=height y=weight /
    alpha=.05 clm="clm" cli="cli";
  discretelegend "predict" "conf" ;
endlayout;
```



For more information, seeChapter 8, "Adding Legends to a Graph," on page 117.

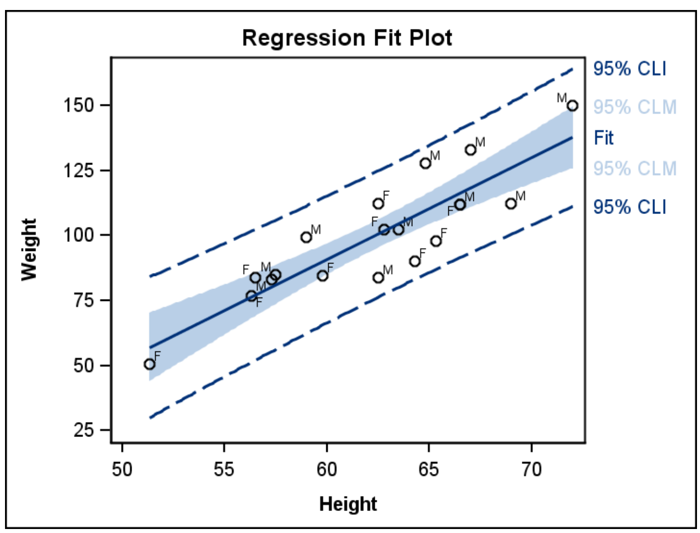
Labels for Plot Features

Most plots have one or more options that enable you to display descriptive labels or data values for points, lines, bars, or bands.

DATALABEL= *column*	Specifies a column to label data points in a scatter plot, series plot, needle plot, stepplot, or vectorplot.
DATALABELATTRS= *text-properties*	Specifies text properties for data labels.
CURVELABEL= "*string*" | *column*	Specifies a string or a string-column to label one or more lines in a REFERENCELINE, DENSITYPLOT, LINEPARM, REGRESSIONPLOT, LOESSPLOT, PBSPLINEPLOT, SERIESPLOT, or STEPPLOT statement.
CURVELABELUPPER= "*string*" | *column* CURVELABELLOWER= "*string*" | *column*	Specifies a string or a string-column to label one or more lines in a BANDPLOT or MODELBAND statement.
CURVELABELATTRS= *text-properties*	Specifies text properties for curve label(s).
CURVELABELLOCATION= INSIDE | OUTSIDE	Specifies whether the curve label(s) are located inside or outside the plot area.
CURVELABELPOSITION=	Specifies positioning options for the curve label(s).

```
layout overlay;
  modelband "cli" / display=(outline)
    outlineattrs=GraphPrediction
    curvelabelupper="95% CLI"
    curvelabellower="95% CLI"
    curvelabelattrs=
```

```
        (color=GraphPrediction:Contrastcolor)
        curvelabellocation=outside ;

      modelband "clm" / display=(fill)
        fillattrs=GraphConfidence
        curvelabelupper="95% CLM"
        curvelabellower="95% CLM"
        curvelabelattrs=
        (color=GraphConfidence:Color)
        curvelabellocation=outside ;

      scatterplot x=height y=weight /
        primary=true datalabel=sex ;

      regressionplot x=height y=weight /
        alpha=.05 clm="clm" cli="cli"
        curvelabel="Fit"
        curvelabelattrs=
        (color=GraphFit:ContrastColor)
        curvelabellocation=outside ;
    endlayout;
```



For more information, see Chapter 7, “Adding and Changing Text in a Graph,” on page 101.

Grouping

Many plots support a GROUP= option, which causes visually different markers, lines, or bands to be displayed for each distinct data value of the specified column. You can vary the appearance of group values with the INDEX= option.

GROUP= *column*	Specifies a group column, always treated as having discrete values. For an example use, see the example for “Legend Statements” on page 28.
INDEX= *positive-integer-column*	Specifies an integer column that associates each distinct data value to a predefined graphical style element GraphData1, GraphData2, …

For more information on using groups, see Chapter 6, “Managing Graph Appearance: General Principles,” on page 83.

Axis Assignment

All 2D plots have four potential axes: X, X2, Y, and Y2. You can choose which axes any plot uses. Axis options are typically specified on LAYOUT statement containing the plot.

XAXIS= X | X2	Specifies whether the plot's X= column is displayed on the X or X2 axis.
YAXIS= Y | Y2	Specifies whether the plot's Y= column is displayed on the Y or Y2 axis.

For more information, see Chapter 5, “Managing Axes in an OVERLAY Layout,” on page 53.

Data Tips

Data tips (or tooltips) are text balloons that appear in HTML pages when you move your mouse pointer over a plot component such as a line, marker, or filled area of a graph. To obtain default data tips, simply specify **`ODS GRAPHICS / IMAGEMAP;`** as well as the ODS HTML destination. You can customize the data tip information.

ROLENAME= (*role=column* …)	Creates additional roles to customize data tips.
TIP= (*role-names*)	Specifies which plot roles are used for data tips.
TIPFORMAT=(*role=format* …)	Specifies a format to be applied to the data for a plot role.
TIPLABEL=(*role*="*string*" …)	Specifies a label to be applied to the column for a plot role.

For more information and an example, see “Controlling Data Tips” on page 337.

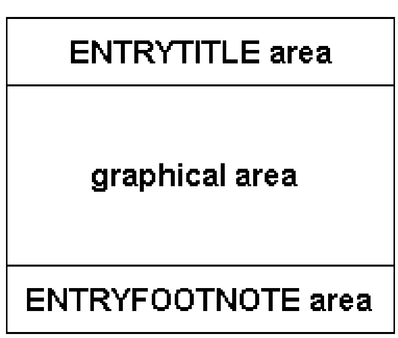
Chapter 4

Using a Simple Single-cell Layout

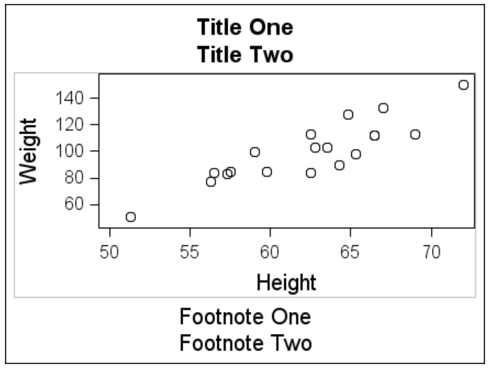
The LAYOUT OVERLAY Statement

The LAYOUT OVERLAY statement builds a 2D, single-cell graph by overlaying the results of the statements that are contained in the layout block. This layout is one of several possible layout containers in GTL. Other chapters provide detailed information on the other layout types. It is recommended that you learn about this type of layout first, because most of the other layout chapters contrast their feature sets with those of the OVERLAY layout.

The outermost layout block of any template defines the content of the graphical area, which is represented in the following schematic:



The graph in this next figure was defined by an OVERLAY layout with its border turned on. The layout contains a simple scatter plot. The boundaries of the layout container are shown by a light gray border. Everything within this border is managed by the layout.



The OVERLAY layout container controls

- which statements (plot, legend, text) can be included in the layout block
- which statements can be combined in the plot area bounded by the axes
- various axis features
  - which axes are used (there are four available: X and Y, as well X2 and Y2)
  - which axis types are used (axis types are LINEAR, DISCRETE, LOG, and TIME)
  - axis label, axis data range, ticks, and tick values
  - other axis features such as offsets
- border, padding, and background properties
- positioning and alignment of all contained plots, text, legends, and nested layouts
- default appearance of the generated plots (CYCLEATTRS= *option*).

The layout container also queries the contained statements for options that might change the default internal rules for combining plots.

Common Overlay Combinations

After you become familiar with the plot statements GTL offers, you will see them as basic components that can be stacked in many ways to form more complex plots. For example, there is no "BARLINE" statement in GTL. You design this kind of graph by overlaying a SERIESPLOT on a BARCHART or BARCHARTPARM.

```
proc template;
  define statgraph barline;
    begingraph;
      entrytitle "Overlay of REFERENCELINE, BARCHARTPARM and SERIESPLOT";
      layout overlay;
        referenceline y=25000000 / curvelabel="Target";
        barchartparm x=year y=retail;
        seriesplot x=year y=profit / name="series";
        discretelegend "series";
      endlayout;
    endgraph;
  end;
run;

/* compute sums for each product line */
```

```
proc summary data=sashelp.orsales nway;
  class year;
  var total_retail_price profit;
  output out=orsales sum=Retail Profit;
run;

proc sgrender data=orsales template=barline;
  format retail profit comma12.;
run;
```

The output reflects the requested stacking order.

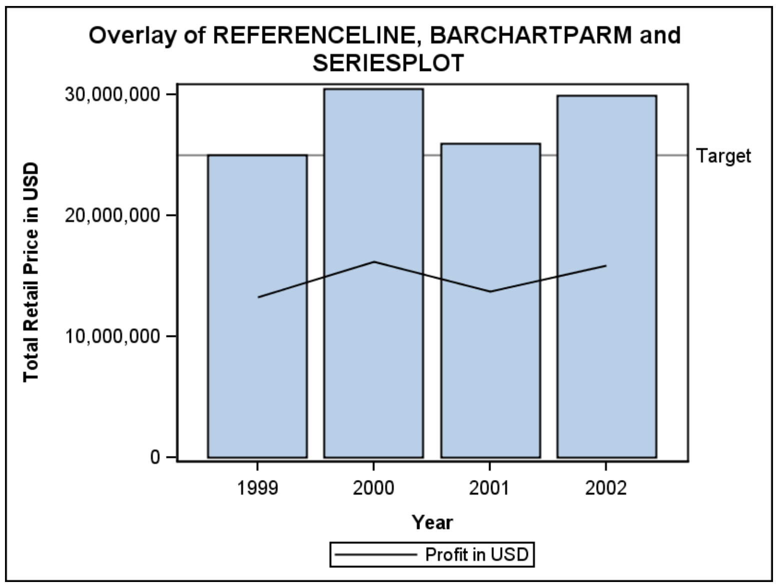
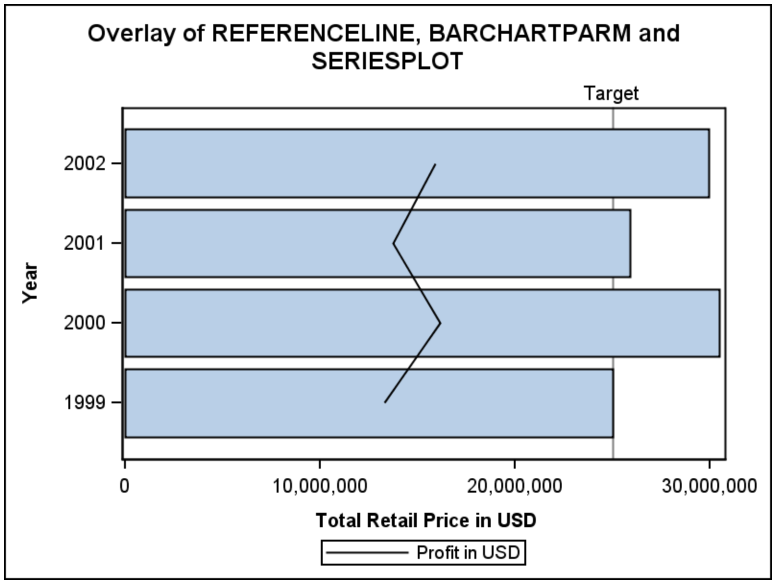


Chart Orientation. When creating a bar chart, it is sometimes desirable to rotate the chart from vertical to horizontal. GTL does not provide separate statements for vertical and horizontal charts—each is considered to be the same plot type with a different orientation. To create the horizontal version of the bar-line graph, you need to specify ORIENT=HORIZONTAL on the BARCHARTPARM statement:

```
proc template;
  define statgraph barline;
    begingraph;
      entrytitle "Overlay of REFERENCELINE, BARCHARTPARM and SERIESPLOT";
      layout overlay;
        referenceline x=25000000 / curvelabel="Target";
        barchartparm x=year y=retail / orient=horizontal ;
        seriesplot x=profit y=year / name="series"
          legendlabel="Profit in USD";
        discretelegend "series";
      endlayout;
    endgraph;
  end;
run;
```

Here, the Y axis becomes the category (DISCRETE) axis, and the X axis is used for the response values. Both the REFERENCELINE and SERIESPLOT reflect this directly by changing the variables that are mapped to the X and Y axes. The variable mapping for BARCHARTPARM remains the way it was, but we add the ORIENT=HORIZONTAL option to swap the axis mappings. The data set up and SGRENDER step are unchanged.



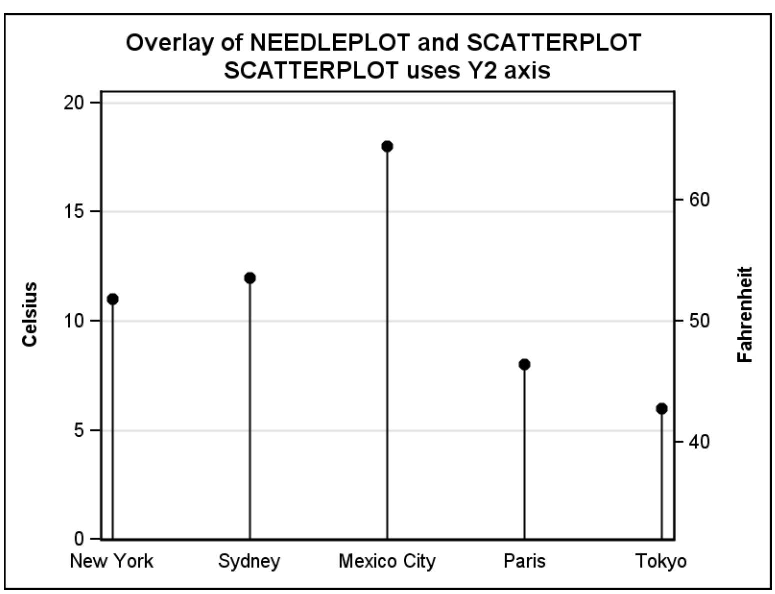
This same strategy would be used to create a horizontal box plot or histogram. If you wanted to reverse the ordering of the Y axis, you could add the REVERSE=TRUE option to the Y-axis options:

```
layout overlay / yaxisopts=(reverse=true);
```

Multiple Axes. Sometimes you have equivalent data in different scales (currency, measurements, and so on), or comparable data in the same scale that you want to display on independent opposing axes.

The OVERLAY layout supports up to four independent axes, with a Y2 opposing the Y axis to the right, and an X2 axis opposing the X axis at the top of the layout container.

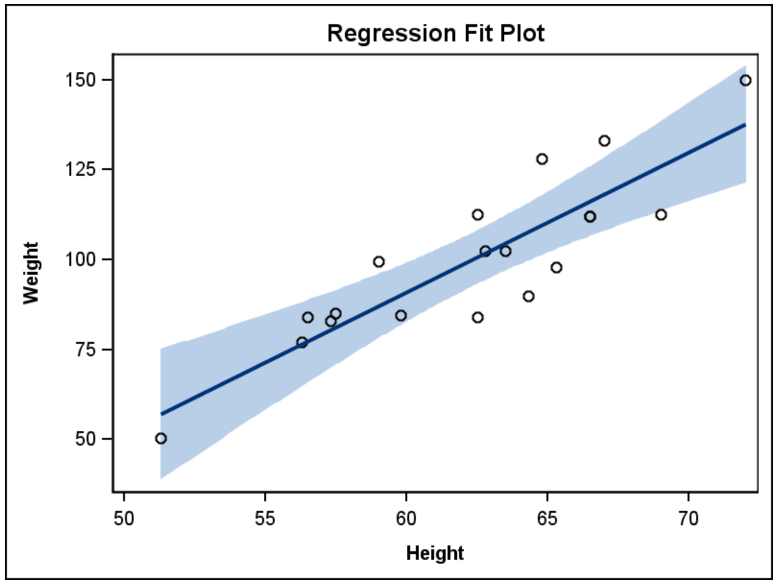
The following is a complete program to generate this type of graph. We would like to display Fahrenheit temperatures on a separate Y2 axis from the Y axis used to display Celsius temperatures. For this particular example, it is not necessary to have input variables for both temperatures because an EVAL function can be used to compute a new column of data within the context of the template.



At this point, the most important concept to understand about template code is that an independent axis can be created by mapping data to it. Notice that the SCATTERPLOT statement uses the YAXIS=Y2 option. This causes the Y2 to axis to be displayed and scaled with the computed variable representing Fahrenheit values. It is important to note that multiple plots in an overlay share the same axis (such as the X-Axis). Hence, the options to control the axis attributes are not found on the plot statements, but rather in the LAYOUT statement. Most of the Y and Y2 axis options are included to force the tick marks for the two different axis scales to exactly correspond. This example and many other axis issues are discussed in detail in Chapter 5, "Managing Axes in an OVERLAY Layout," on page 53.

```
data temps;
  input City $1-11  Celsius;
datalines;
New York     11
Sydney       12
Mexico City  18
Paris         8
Tokyo         6
run;

proc template;
 define statgraph Y2axis;
  begingraph;
   entrytitle "Overlay of NEEDLEPLOT and SCATTERPLOT";
   entrytitle "SCATTERPLOT uses Y2 axis";
   layout overlay /
       xaxisopts=(display=(tickvalues))
       yaxisopts=(griddisplay=on offsetmin=0
         linearopts=(viewmin=0  viewmax=20
           thresholdmin=0 thresholdmax=0))
       y2axisopts=(label="Fahrenheit" offsetmin=0
         linearopts=(viewmin=32 viewmax=68
           thresholdmin=0 thresholdmax=0)) ;
     needleplot  x=City y=Celsius;
     scatterplot x=City y=eval(32+(9*Celsius/5)) / yaxis=y2
       markerattrs=(symbol=circlefilled);
   endlayout;
  endgraph;
 end;
run;

proc sgrender data=temps template=y2axis;
run;
```

Often, the input data's organization will affect your choice of plot statements in an OVERLAY layout. In Chapter 3, "Overview of Basic Statements and Options," on page 19, you saw that you would choose different plot statements for a fit plot, depending on the nature of the input data for a fit plot.



Computed versus Paramterized Plots. If the data set has numeric columns for raw data values of Height and Weight, the simplest way to create the fit line and confidence bands is with a REGRESSIONPLOT, LOESSPLOT, or PBSPLINEPLOT statement and a MODELBAND statement. All of these computed plot statements generate the values of new columns corresponding to the points of the fit line and band boundaries.

```
layout overlay;
  modelband "myclm";
  scatterplot    x=height y=weight / primary=true;
  regressionplot x=height y=weight / alpha=.01 clm="myclm";
endlayout;
```

If you have data computed by an analytic procedure that provides points on the fit line and bands, you would choose a SERIESPLOT and BANDPLOT for the graph. This technique is required when the desired fit line can't be computed by the REGRESSIONPLOT, LOESSPLOT, or PBSPLINEPLOT statement options.

```
layout overlay;
  bandplot x=height limitupper=uclm limitlower=lclm /
    fillattrs=GraphConfidence;
  scatterplot x=height y=weight / primary=true;
  seriesplot  x=height y=p / lineattrs=GraphFit;
endlayout;
```

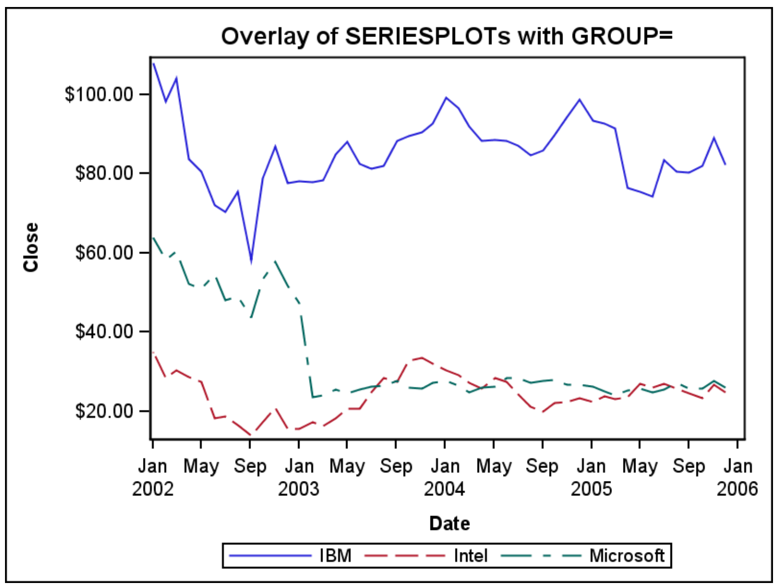
Also, notice that additional options are used to set the appearance of the fit line and band to match the defaults for REGRESSIONPLOT and MODELBAND.

Grouped Data. Another common practice is to overlay series lines for comparisons. If your data contains a classification variable in addition to X and Y variables, you could use one SERIESPLOT statement with a GROUP= option:

```
proc template;
  define statgraph seriesgroup;
    begingraph;
    entrytitle "Overlay of SERIESPLOTs with GROUP=";
      layout overlay;
        seriesplot x=date y=close / group=stock name="s";
        discretelegend "s";
      endlayout;
    endgraph;
```

```
  end;
run;

proc sgrender data=sashelp.stocks template=seriesgroup;
   where date between "1jan2002"d and "31dec2005"d;
run;
```

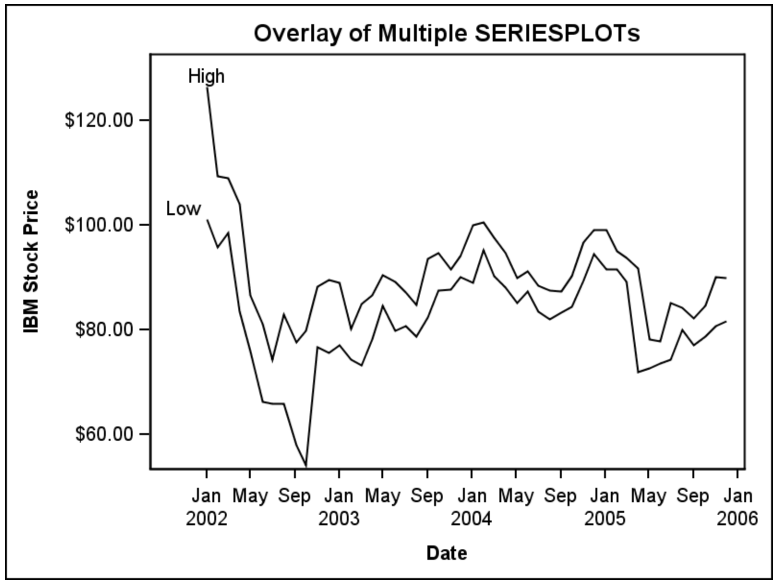
By default when you use a GROUP= option with a plot, the plot automatically cycles through appearance features (colors, line styles, and marker symbols) to distinguish group values in the plot. The default features that are assigned to each group value are determined by the current style. For the following graph, the default colors and line styles of the LISTING style are used:



Multiple Response Variables. If your data has multiple response variables, you could create a SERIESPLOT overlay for each response. In such situations, you often need to adjust the Y axis label.

```
proc template;
 define statgraph series;
  begingraph;
    entrytitle "Overlay of Multiple SERIESPLOTs";
    layout overlay / yaxisopts=(label="IBM Stock Price") ;
      seriesplot x=date y=high / curvelabel="High";
      seriesplot x=date y=low / curvelabel="Low";
    endlayout;
  endgraph;
 end;
run;


proc sgrender data=sashelp.stocks template=series;
   where date between "1jan2002"d and "31dec2005"d
         and stock="IBM";
run;
```

Notice that, by default, each overlaid plot in this situation has the same appearance properties.



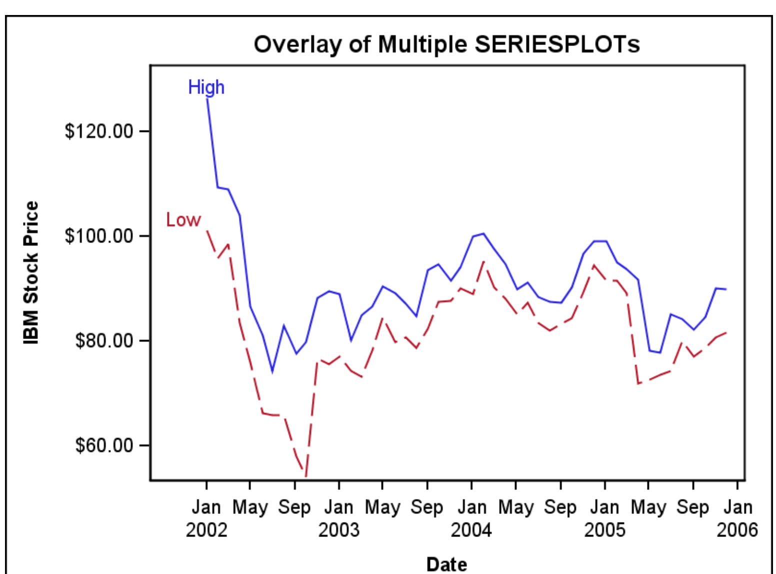
Appearance Options. In cases when multiple plots have the same appearance, you can use plot options to adjust the appearance of individual plots. For example, to adjust the series lines from the previous example, you can use the LINEATTRS= option:

```
layout overlay / yaxisopts=(label="IBM Stock Price");
  seriesplot x=date y=high / curvelabel="High" lineattrs=GraphData1 ;
  seriesplot x=date y=low  / curvelabel="Low"  lineattrs=GraphData2 ;
endlayout;
```

You can also use the CYCLEATTRS= option, which is an option of the LAYOUT OVERLAY statement that might cause each statement to acquire different appearance features from the current style.

```
layout overlay / yaxisopts=(label="IBM Stock Price") cycleattrs=true ;
  seriesplot x=date y=high / curvelabel="High";
  seriesplot x=date y=low / curvelabel="Low";
endlayout;
```

Either coding produces the following graph:



For additional information on how set the appearance features of plots, see see Chapter 6, "Managing Graph Appearance: General Principles," on page 83 and Chapter 17, "Managing the Graph Appearance with Styles," on page 299.

How Plots are Overlaid

The following sections explain in more detail how the overlay process works and why some overlay constructs might not generate the graph you expect.

Statements Allowed in the Overlay Container

If you were to randomly place GTL statements within a LAYOUT OVERLAY block, you would often get compile errors. The following basic rules indicate which statements can be used within the layout block:

- all 2D plot statements except SCATTERPLOTMATRIX
- statements such as ENTRY, DISCRETELEGEND, and CONTINUOUSLEGEND
- GRIDDED, LATTICE, and overlay-type layout blocks can be nested.

However, the following restrictions apply:

- 3D plot statements cannot be included. Place these statements in a LAYOUT OVERLAY3D block.
- ENTRYTITLE or ENTRYFOOTNOTE statements cannot be included. Place these statements outside the outermost layout block.
- Other layout types such as PROTOTYPE, DATALATTICE, and DATAPANEL layouts cannot be nested in an OVERLAY layout.

Restrictions on Allowed Statements

Even among the statements that are valid within an OVERLAY layout, some restrictions apply to their use. For example, some dependent statements must be accompanied by at least one stand-alone plot statement, such as SCATTERPLOT or SERIESPLOT, in order to produce a usable graph. See Chapter 3, "Overview of Basic Statements and Options," on page 19 for lists of stand-alone and dependent statements.

For example, if you were to execute a template with the following layout block, it would produce an empty graph at runtime.

```
proc template;
  define statgraph test;
    begingraph;
      layout overlay;
        referenceline x=10;
      endlayout;
    endgraph;
  end;
run;
proc sgrender data=sashelp.class template=test;
run;
```

```
WARNING: A blank graph is produced. For possible causes, see the graphics
        template language documentation.
```

The GTL Reference documentation for the REFERENCELINE statement states an important requirement that explains why this graph is empty:

A REFERENCELINE statement can be used only within 2D overlay-type layouts (OVERLAY, OVERLAYEQUATED, or PROTOTYPE). A stand-alone plot statement that provides a sufficient data range for determining axis extents must be included in the layout. For example, a REFERENCELINE statement can be used with a scatter plot or a histogram statement.

Restrictions on Statement Combinations

Certain combinations of contained statements produce unexpected results. This section examines why these combinations do not produce the expected graph.

Consider the following template code, which generates a warning in the log:

```
proc template;
  define statgraph test;
    begingraph;
      layout overlay;
        boxplot x=age y=weight;
        regressionplot x=age y=weight;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.class template=test;
run;
```

```
WARNING: REGRESSIONPLOT statement cannot be placed under a layout OVERLAY with a
         discrete axis. The plot will not be drawn.
```

When multiple statements that potentially contribute to axis construction are placed in the layout, the layout must verify that all data that is mapped to a particular axis is of the same type (all numeric, or all character, or all time). In addition, the layout must verify that each plot can use the requested axis type(s). In this case, the first statement in the layout is a BOXPLOT. Statements such as BOXPLOT, BOXPLOTPARM, BARCHART, and BARCHARTPARM treat the X=column as a categorical variable (regardless of data type) and build a DISCRETE (categorical) axis. Therefore, because BOXPLOT is the first statement in this example layout, it determines that the X axis is set to DISCRETE, and subsequent plots must be compatible with a discrete axis.

Many computed plots, such as REGRESSIONPLOT, LOESSPLOT, and ELLIPSE, require both X and Y axes to be of LINEAR type, which is a standard numeric interval axis type. Had you specified a SCATTERPLOT instead of a REGRESSIONPLOT, there would be no problem because a SCATTERPLOT can be displayed on either a DISCRETE or LINEAR X and Y axis. The end result of this example is a graph containing only the box plot output.

In this next example, the REGRESSIONPLOT and BOXPLOT statements have been switched:

```
layout overlay;
  regressionplot x=age y=weight;
  boxplot x=age y=weight;
endlayout;
```

```
WARNING: BOXPLOT statement has a conflict with the axis type. The plot will not
be drawn.
```

In this case, the REGRESSIONPLOT (first plot) has fixed the type of the X axis to be LINEAR. Now the BOXPLOT is blocked because it needs a DISCRETE X axis. The end result of this example is a graph containing only the regression line.

Because a SCATTERPLOT can be included on either LINEAR or DISCRETE axes, you might think the following combination is valid:

```
layout overlay;
  scatterplot x=age y=weight;
  boxplot x=age y=weight;
endlayout;
```

```
WARNING: BOXPLOT statement has a conflict with the axis type. The plot will not
be drawn.
```

In this case, the SCATTERPLOT (first statement) sets the X or Y axis type to LINEAR if the variable for that axis is numeric—even though the data might be categorical in nature. However, if the variable is character, the SCATTERPLOT must use a DISCRETE axis. So, once again the BOXPLOT is not displayed. If you switch the statements, both plots are drawn because after the X axis is fixed to be DISCRETE, the SCATTERPLOT can display numeric values on a DISCRETE axis.

When a character variable is used, the axis-type conflict often does not arise. The following combination works regardless of statement order. In either case, the DISCRETE X axis will display a combination of AGE values with box plots above and SEX values with scatter points above.

```
layout overlay;
  scatterplot x=sex y=weight;
  boxplot x=age y=weight;
endlayout;
```

Avoiding Plot Conflicts

In GTL, it is important to know what types of axes a given plot requires or can support. If you understand the basic ideas behind previous examples, you can use the following additional GTL syntax to avoid some of the problems caused by the first plot statement deciding the axis type:

- use the PRIMARY=TRUE option on a plot statement to ensure that plot is used to determine the axis type
- declare an axis type on the layout block.

Most non-dependent plot statements support the PRIMARY= option. By default, PRIMARY=TRUE for the first plot and PRIMARY=FALSE for the rest of the plots in the layout. On a per-axis basis, only one plot in an overlay can use PRIMARY=TRUE. If multiple plots specify PRIMARY=TRUE for the same axis, the last one encountered is considered primary. The plot that is designated as primary by default defines the axis types for the axes it uses, regardless of its order within the layout block. This is useful when you want a certain stacking order for the plots, but don't want the first plot to set the axis features, such as axis type and default axis label.

In the following example, the BOXPLOT sets the X axis to be DISCRETE and the Y axis to be LINEAR:

```
layout overlay;
  scatterplot x=age y=weight;
  boxplot x=age y=weight / primary=true ;
endlayout;
```

All layouts that manage axes provide options that enable you to control the axis features. The following example shows how to declare an axis type for the X axis. Any plot in the layout that cannot support a discrete axis will be dropped. Also note that specifying an axis type overrides the default axis type that is derived from the primary plot. Axis options are discussed in detail in Chapter 5, “Managing Axes in an OVERLAY Layout,” on page 53.

```
layout overlay / xaxisopts=(type=discrete) ;
  scatterplot x=age y=weight;
  boxplot x=age y=weight;
endlayout;
```

Some plot combinations can never be used. A histogram and bar chart look similar, but they have different data and axis requirements. The histogram must use a linear X axis and the bar chart must use a discrete X axis. The two plot types can never be overlaid.

```
layout overlay;
  barchart x=age;
  histogram age;
endlayout;
```

```
WARNING: HISTOGRAM statement has a conflict with the axis type. The plot will not
be drawn.
```

```
layout overlay;
  histogram age;
  barchart x=age;
endlayout;
```

```
WARNING: BARCHART statement has a conflict with the axis type. The plot will not
be drawn.
```

Plots with Incompatible Data

All plot statements have required argument(s) that map input data column(s) to one or more axes. Many plot statements have restrictions on the variable type (numeric or character) that can be used for the required arguments.

For example, the HISTOGRAM statement accepts only a numeric variable for the required argument. Consider the following template:

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

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

If you were to create this template, you would not see a compilation error or warning because no variables are checked at compile time. However, you would see warnings in the log when the template is executed:

```
proc sgrender data=sashelp.class template=test;
run;
```

```
WARNING: Invalid data passed to BIN. Variable must be numeric.
WARNING: The histogram statement will not be drawn because one or more of the
         required arguments were not supplied.
WARNING: A blank graph is produced. For possible causes, see the graphics
         template language documentation.
```

In general, GTL produces a graph whenever possible. Plots in the overlay that can be drawn will be drawn. Plots are not drawn if they have incompatible data for the required arguments or if they cannot support the existing axis type(s). Hence, you might get a graph with some or none of the requested plot overlays.

The same strategy extends to plot options that have incompatible data. In the following example, the wrong variable name was used for the GROUP= option. In the data, the column is named SEX, not GENDER. This is not regarded as an error condition—the bar chart will be drawn without groups.

```
layout overlay;
  barchart x=age / group=gender;
endlayout;
```

Chapter 5

Managing Axes in an OVERLAY Layout

Introduction to Axis Management

When you write GTL programs, all axes are automatically managed for you. For example, in a LAYOUT OVERLAY block, the overlay container decides

- which axes are displayed

- the axis type of each axis (linear, time, ...)
- the data range of each axis
- the label of the axis
- other axis characteristics, some of which are derived from the current style.

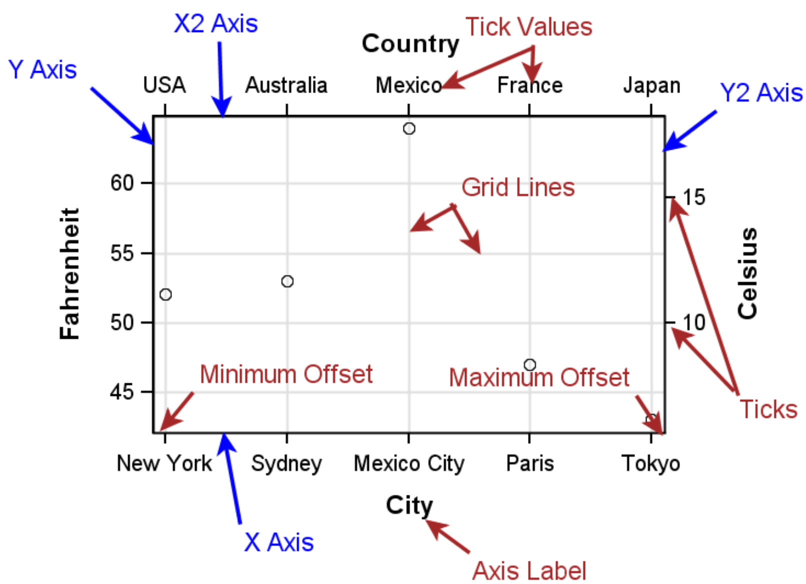
Usually, the internal techniques that are used to manage axes produce good default axes. Occasionally, you might find some feature you want to change. Layout statements provide many axis options that change the default axis behavior. This chapter shows how axes are managed by default and the programming options that are available to you for changing that behavior.

Note: This chapter discusses axis features that are specific to an OVERLAY layout when it is the outermost layout and not nested in another layout. Nesting layouts sometimes causes interactions that affect the axis features. You should read this chapter before reading about other layout types because this chapter provides the basic principles of axis management. Be aware, though, that the other layout types (for example, OVERLAYEQUATED, OVERLAY3D, LATTICE, DATAPANEL and DATALATTICE) also control axes. Many of these layouts have similar although not identical options to the OVERLAY layout. See the chapters on these other layouts for detailed discussions on how they manage axes.

Axis Terminology

The OVERLAY container has up to four independent axes (X, Y, X2, Y2) that can be used in various combinations. Each axis has the following features, which can be selectively displayed using the option or setting that is shown in parentheses.

- axis line (LINE)
- axis label (LABEL)
- tick marks (TICKS)
- tick values (TICKVALUES)
- grid lines drawn perpendicular to the axis at tick marks (GRIDDISPLAY=)
- gaps at the beginning and end of the axis (OFFSETMIN= and OFFSETMAX=).



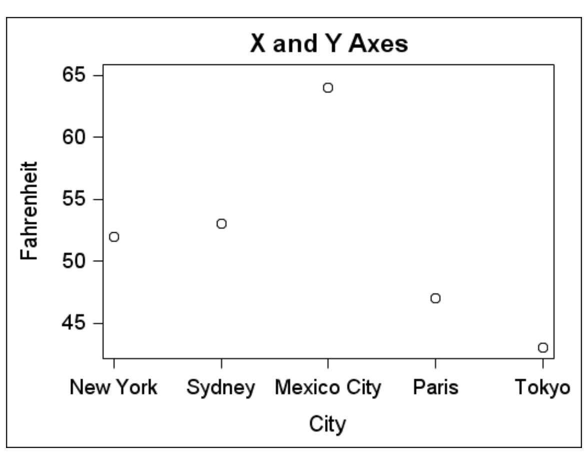
How Plot Statements Affect Axis Construction

Primary and Secondary Axes. The LAYOUT OVERLAY container supports two horizontal (X and X2) and two vertical (Y and Y2) axes. The bottom axis (X) and the left axis (Y) are the default axes, referred to as the primary axes. The top axis (X2) and the right axis (Y2) are referred to as the secondary axes and are displayed only if they are requested. For example, consider this simple layout block:

```
layout overlay;
  scatterplot x=city y=fahrenheit;
endlayout;
```

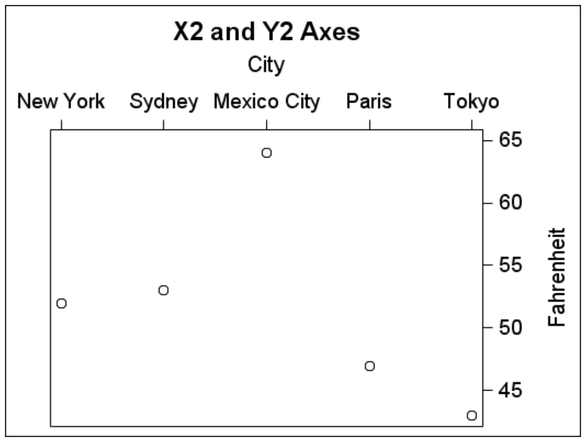
Explicitly, the layout block means the following:

```
layout overlay;
  scatterplot x=city y=fahrenheit / xaxis=x yaxis=y ;
endlayout;
```



The defaults result in an XY plot having only two axes, X and Y. However, you can request that either the X or Y columns be mapped to the X2 or Y2 axis. The XAXIS= option can be set to X or X2. Similarly, the YAXIS= option can be set to Y or Y2:

```
layout overlay;
  scatterplot x=city y=fahrenheit / xaxis=x2 yaxis=y2 ;
endlayout;
```

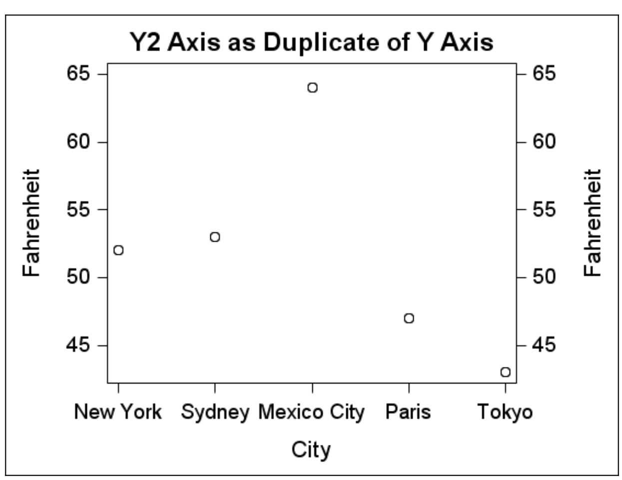


A single plot statement can activate one horizontal and/or one vertical axis. It cannot activate both horizontal or both vertical axes. Thus, to see both a Y and Y2 axis based on the same Y column, you could specify an additional plot statement:

```
layout overlay;
  scatterplot x=city y=fahrenheit / xaxis=x yaxis=y ;
  scatterplot x=city y=fahrenheit / xaxis=x yaxis=y2 ;
endlayout;
```

This layout could be more compactly written as follows:

```
layout overlay;
  scatterplot x=city y=fahrenheit;
  scatterplot x=city y=fahrenheit / yaxis=y2 ;
endlayout;
```

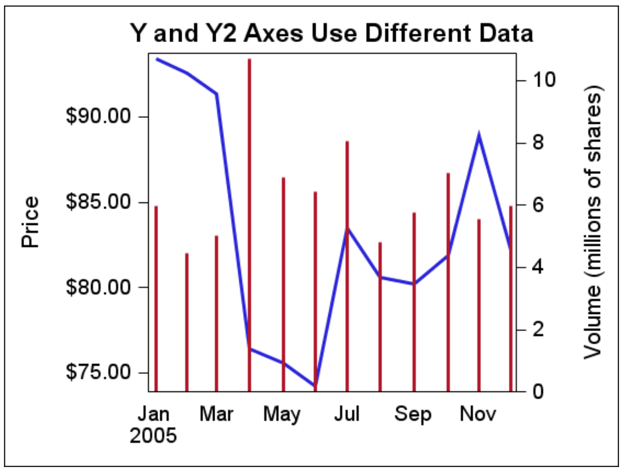


Note that this coding produces two overlaid scatter plots, each with five markers. Because the five (X,Y) value pairs and the five (X,Y2) value pairs are identical, the Y and Y2 axes are identical and the markers are exactly superimposed. However, it is not necessary to create a second plot when you want the secondary axis to be a duplicate of the primary axis. A more direct way to accomplish this is shown in "Specifying Axis Options" on page 58.

The next two examples show the independent nature of primary and secondary axes. In each case, a different data column is mapped to the Y and Y2 axes.

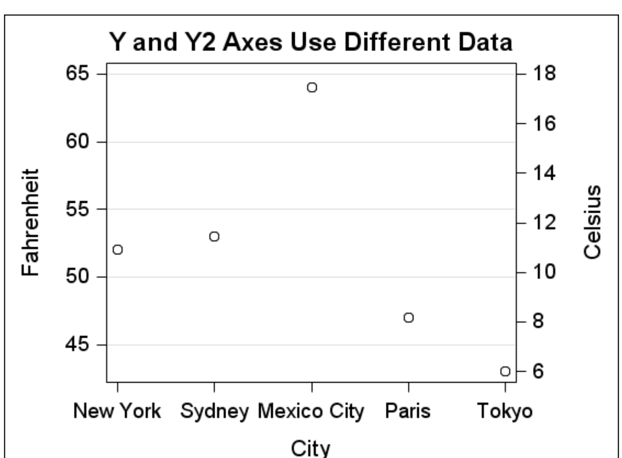
```
layout overlay;
  seriesplot x=date y=price;
  needleplot x=date y=volume / yaxis=y2;
endlayout;
```

As the following figure shows, the primary and secondary Y axes are independently scaled and there is not a necessary connection between the units or data ranges of either axis.



In the next example, even though the Y and Y2 variables are different, the primary and secondary Y axes represent the same data range in different units. In such cases, the positioning of the tick values on each axis should be coordinated so that the grid lines represent the same temperature on each axis. "Apply Axis Thresholds" on page 64 provides example code that shows how to coordinate the tick value positions.

```
layout overlay;
  scatterplot x=city y=fahrenheit;
  scatterplot x=city y=celsius  / yaxis=y2 ;
endlayout;
```



Specifying Axis Options

To set axis options on the LAYOUT OVERLAY statement, you use the following syntax. Notice that each axis has its own separate set of options, and that the option specifications must be enclosed in parentheses. GTL frequently uses parentheses to bundle options that modify a specific feature. These are called "option bundles."

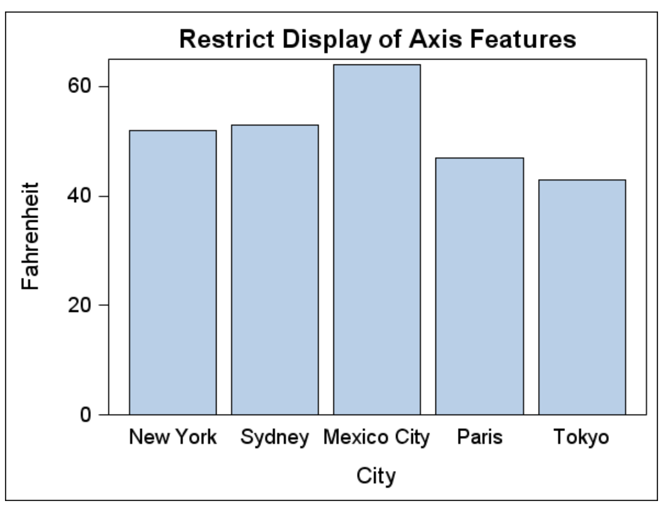
```
layout overlay / xaxisopts =(options) yaxisopts =(options)
  x2axisopts=(options) y2axisopts=(options);
```

If you specify the X2AXISOPTS= or Y2AXISOPTS= options but there is no data mapped to these axes, the option bundles are ignored.

One of the basic options that you can set for any axis is DISPLAY= *keyword* | (*feature-list*). Four features are available for the *feature-list*: LINE, TICKS, TICKVALUES, and LABEL. The keywords STANDARD and ALL are equivalent to specifying the full list: (LINE TICKS TICKVALUES LABEL). You can also use DISPLAY=NONE to completely suppress all parts of the axis.

Example: Some plots don't really need TICKS on all axes. The follow axis option eliminates the ticks on the X axis by omitting the TICKS value on the *feature-list*.

```
layout overlay / xaxisopts=( display=( line label tickvalues ) );
  barchartparm x=city y=fahrenheit;
endlayout;
```

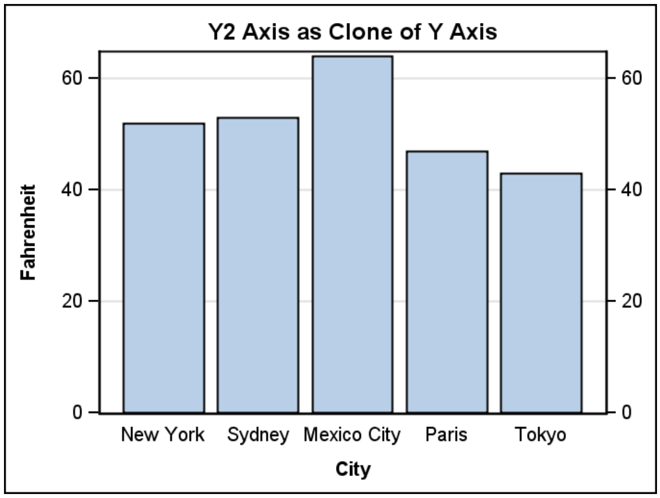


Let's return to the common situation where you want a duplicated Y2 axis. Here is the most efficient way to do it:

```
layout overlay / yaxisopts=( displaysecondary=standard );
  barchartparm x=city y=fahrenheit;
endlayout;
```

This specification creates the Y2 axis as a duplicate of the Y axis: all features are displayed without having to map data to the Y2 axis. You can also restrict the secondary axis features that are displayed by specifying a list of the features you want to be displayed. The values available for the DISPLAYSECONDARY= option are the same as those of the DISPLAY= option. The following example specifies that the secondary axis label will not be displayed. It also requests that grid lines be displayed on the Y axis:

```
layout overlay / xaxisopts=( display=( line label tickvalues ) )
    yaxisopts=( displaysecondary=( ticks tickvalues line )
      griddisplay=on );
  barchartparm x=city y=fahrenheit;
endlayout;
```



Default Axis Construction and Related Options

Determine Axis Type

To determine axis types, the OVERLAY container examines all of the stand-alone plot statements that are specified. It also examines whether an axis type has been specified with the TYPE= setting on an axis option (for example, on XAXISOPTS=). If there is only one stand-alone plot, or a plot is designated as PRIMARY, the rules are simple:

- If the plot statement that is mapped to an axis treats data values as discrete (such as the X= column of the BARCHART or BOXPLOT statement), the axis type is DISCRETE for that axis, regardless of whether the data column that is mapped to the axis is character or numeric. A DISCRETE axis has tick values for each unique value in a data column.
- If the plot statement that is mapped to an axis bases the axis type on the data type of the assigned values, a DISCRETE axis is created when the column type is character. Otherwise, a TIME or LINEAR axis is created.
- If the plot statement that is mapped to an axis specifies a numeric column and the column has a date, time, or datetime format associated with it, the axis type is TIME. See "TIME Axes" on page 72 for examples. Otherwise, the numeric axis type is LINEAR, the general numeric axis type. See "LINEAR Axes" on page 69 for examples.
- A LOG axis is never automatically created. To obtain a LOG axis, you must explicitly declare the axis type with the TYPE=LOG option. See "LOG Axes" on page 75 for examples.
- If a TYPE= *axis-type* option is specified, that is the type used. Plots that cannot support that axis type will not be drawn.

When the overlay container has multiple plots that generate axes, GTL can determine default axis features for the shared axes, or you can use the PRIMARY= option on one of the plot statements to specify which plot you want the GTL to use.

- If no plot is designated as primary, the data columns that are associated with the first plot that generates an axis are considered primary on a per-axis basis.
- If PRIMARY=TRUE for a plot within an overlay-type layout, that plot's data columns and type will be used to determine the default axis features, regardless of where this plot statement occurs within the layout block.
- Only one plot can be primary on a per-axis basis. If multiple plots specify PRIMARY=TRUE for the same axis, the last one encountered is considered primary.

Example: For the following layout block, the BARCHART is considered the primary plot because it is the first stand-alone plot that is specified in the layout and no other plot has been set as the primary plot. A BARCHART requires a discrete X-axis. You cannot change the axis type. It does not matter whether QUARTER is a numeric or character variable. Because the SERIESPLOT can use a discrete axis, the overlay will be successful.

```
layout overlay;
  barchart   x=quarter y=actualSales;
  seriesplot x=quarter y=predictedSales;
endlayout;
```

Example: For the following layout block, the first SERIESPLOT is considered primary. If the QUARTER variable is numeric and has a date format, then the X-axis type will be TIME. If the variable is numeric, but does not have a date format, then the axis type will be LINEAR. If the variable is character, then the axis type will be DISCRETE.

```
layout overlay;
  seriesplot x=quarter y=predictedSales;
  seriesplot x=quarter y=actualSales;
endlayout;
```

Example: For the following layout block, the X-axis is DISCRETE because it was declared to be DISCRETE and this does not contradict any internal decision about axis type because both SERIESPLOT and BARCHART support a discrete axis. It does not matter whether QUARTER is a numeric or character variable.

```
layout overlay / xaxisopts=(type=discrete);
  seriesplot x=quarter y=predictedSales;
  barchart   x=quarter y=actualSales;
endlayout;
```

Example: For the following layout block, the SERIESPLOT is the primary plot. If QUARTER is a character variable, a discrete axis is used and the overlay is successful. However, if QUARTER is a numeric variable, either a TIME or LINEAR axis is used, the BARCHART overlay fails, and a message is written to the log.

```
layout overlay;
  seriesplot x=quarter y=predictedSales;
  barchart   x=quarter y=actualSales;
endlayout;
```

```
WARNING: BARCHART statement has a conflict with the axis type. The plot will not
be drawn.
```

Apply Axis Options

Each of the four possible axes (X, Y, X2, Y2) can be managed with a set of options that apply to axes of any type. In addition, option bundles are available for managing each specific axis type. For example, the following syntax shows the option bundles that are available on the LAYOUT OVERLAY statement's XAXISOPTS= option:

LAYOUT OVERLAY </ XAXISOPTS=(*general-options* LINEAROPTS=(*options*) DISCRETEOPTS=(*options*) TIMEOPTS=(*options*) LOGOPTS=(*options*)) >;

These same bundles are available for the other axes using the following LAYOUT OVERLAY options:

YAXISOPTS= (*same-as-xaxisopts*)
Y2AXISOPTS=(*same-as-xaxisopts*)
X2AXISOPTS=(*same-as-xaxisopts*)

You can specify as many type-specific option bundles as you want, but only the bundle that corresponds to the axis type will be used for a given template execution.

Determine Axis Data Range

After the type of each axis is determined in the layout, the data ranges of all plot statements that contribute to an axis are compared. For LINEAR, TIME, and LOG axes, the minimum of all minimum values and the maximum of all maximum values are derived as a "unioned" data range. For a DISCRETE axis, the data range is the set of all unique values from the sets of all values. The VIEWMIN= and VIEWMAX= options for LINEAR, TIME, and LOG axes can be used to change the displayed axis range. For examples, see “LINEAR Axes” on page 69, “TIME Axes” on page 72, and “LOG Axes” on page 75.

Determine Axis Label

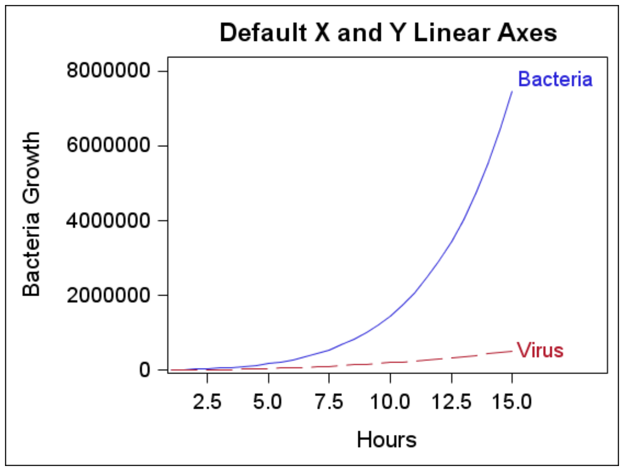
The default axis label is determined by the primary plot. If a label is associated with the data column, the label is used. If no column label is assigned, the column name is used for the axis label. Each set of axis options provides LABEL= and SHORTLABEL= options that can be used to change the axis label. By default, the font characteristics of the label are set by the current style, but the plot statement's LABELATTRS= option can be used to change the font characteristics. See “Axis Appearance Features Controlled by the Current Style” on page 81. The following examples show how axis labels are determined and how to set an axis label.

Consider the following data set, which contains information about bacteria and virus growth:

```
data growth;
  do Hours=1 to 15 by .5;
    Bacteria= 1000*10**( sqrt(Hours ));
    Virus= 1000*10**(log(hours));
    label bacteria="Bacteria Growth" virus="Virus Growth";
    output;
  end;
run;
```

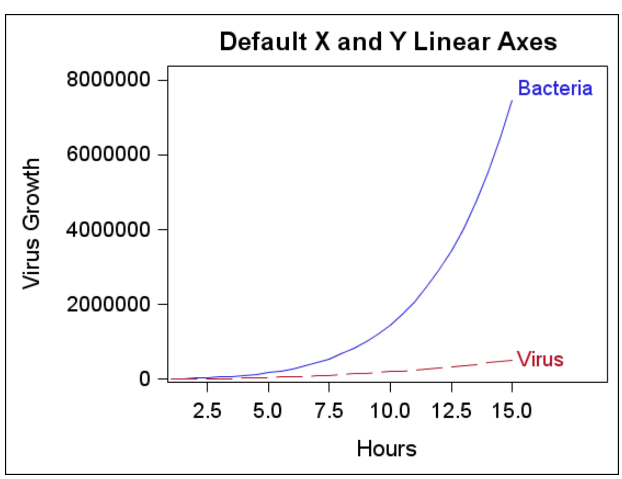
To plot the growth trend for both Bacteria and Virus in the same graph, we can use a simple overlay of series plots. Whenever two or more columns are mapped to the same axis, the primary plot determines the axis label. In the following example, the first SERIESPLOT is primary by default, so its columns determine the axis labels. In this case, the Y-axis label is determined by the BACTERIA column.

```
layout overlay / cycleattrs=true;
  seriesplot x=Hours y=Bacteria/ curvelabel="Bacteria";
  seriesplot x=Hours y=Virus / curvelabel="Virus";
endlayout;
```



If we designate another plot statement as "primary," its X= and Y= columns will be used to label the axes. The PRIMARY= option is useful when you desire a certain stacking order of the overlays, but you want the axis characteristics to be determined by a plot statement that is not the default primary plot statement. In the following example, the second SERIESPLOT is set as the primary plot, so its columns determine the axis labels. In this case, the Y-axis label is determined by the VIRUS column.

```
layout overlay / cycleattrs=true;
  seriesplot x=Hours y=Bacteria/ curvelabel="Bacteria";
  seriesplot x=Hours y=Virus / curvelabel="Virus" primary=true ;
endlayout;
```

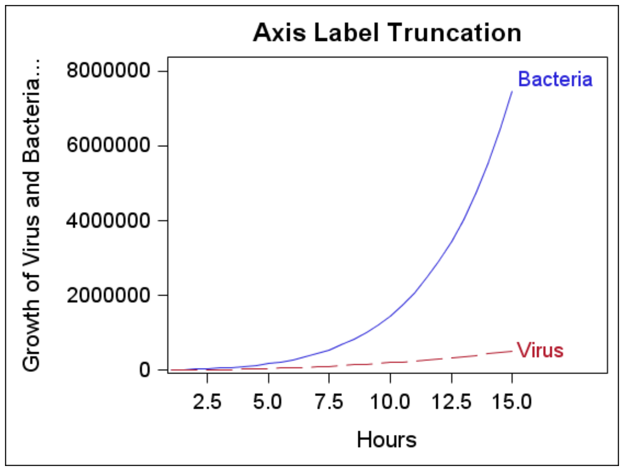


In the previous two examples, allowing the primary plot to determine the Y-axis label did not result in an appropriate label because a more generic label is needed. To achieve this, you must set the axis label yourself with the LABEL= option.

```
layout overlay / cycleattrs=true
    yaxisopts=(label="Growth of Virus and Bateria Cultures") ;
```

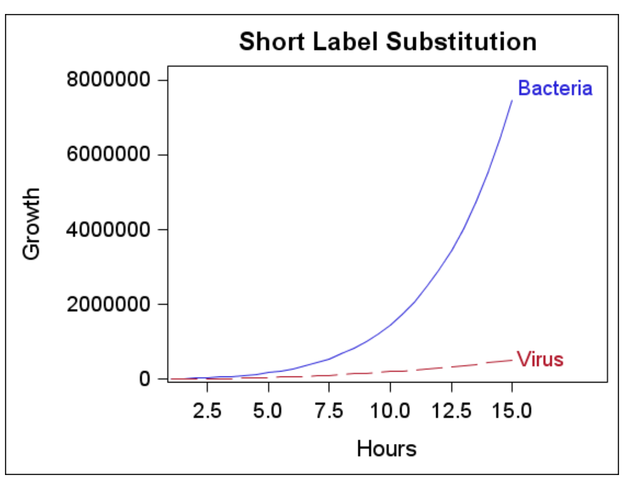
```
  seriesplot x=Hours y=Bacteria/ curvelabel="Bacteria";
  seriesplot x=Hours y=Virus / curvelabel="Virus";
endlayout;
```

Short Labels. If the data column's label is long, or if you supply a long string for the label, the label might be truncated if it won't fit in the allotted space. This might happen when you create a small graph or when the font size for the axis label is large.



As a remedy for these situations, you can specify a "backup" label with the SHORTLABEL= option. The short label will be displayed whenever the default label or the LABEL= string won't fit.

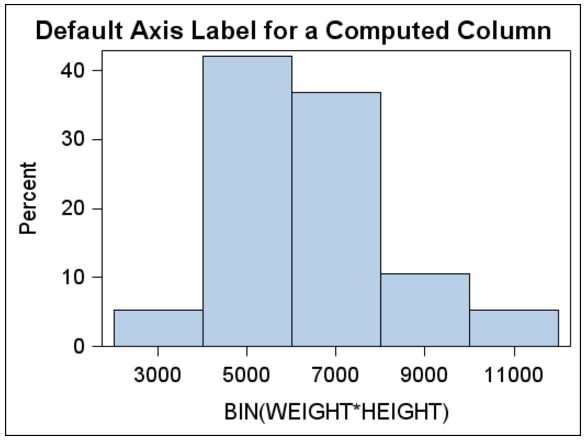
```
layout overlay / cycleattrs=true
    yaxisopts=(label="Growth of Virus and Bacteria Cultures"
               shortlabel="Growth" );
  seriesplot x=Hours y=Bacteria/ curvelabel="Bacteria";
  seriesplot x=Hours y=Virus / curvelabel="Virus";
endlayout;
```



Computed Columns. Another situation where you might want to control the axis label is when a computed column is used.

```
layout overlay;
  histogram eval(weight*height);
endlayout;
```

You can use an EVAL expression to compute a new column that can be used as a required argument. Such columns have manufactured names in the associated data object. The name is based on the input column(s) and the functional transformation that was applied to the input column. In this example, BIN(WEIGHT*HEIGHT) is the manufactured name.



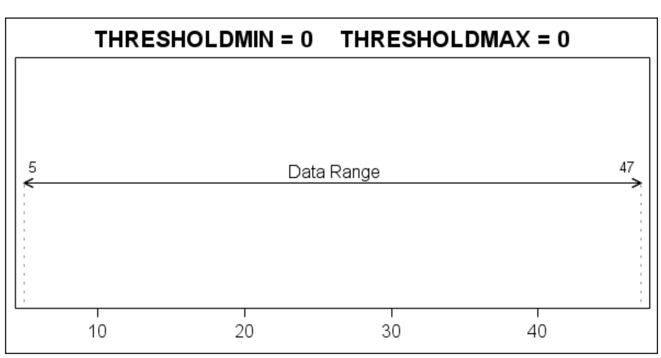
Determine Axis Tick Values

The tick values for LINEAR and TIME axes are calculated according to an internal algorithm that produces good tick values by default. This algorithm can be modified or bypassed with axis options. For examples, see "LINEAR Axes" on page 69 and "LOG Axes" on page 75.

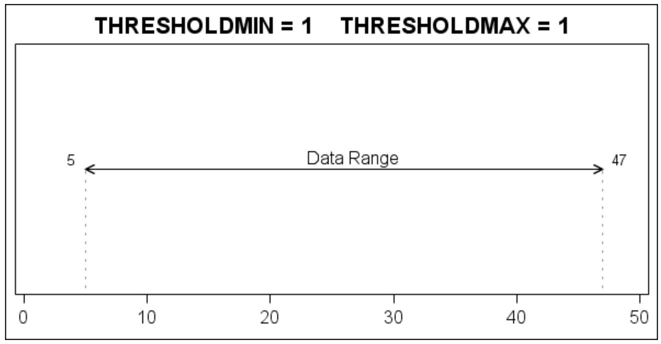
By default, the font characteristics of the tick values are set by the current style. You can set alternative font characteristics with the TICKVALUEATTRS= option. For more information, see "Axis Appearance Features Controlled by the Current Style" on page 81.

Apply Axis Thresholds

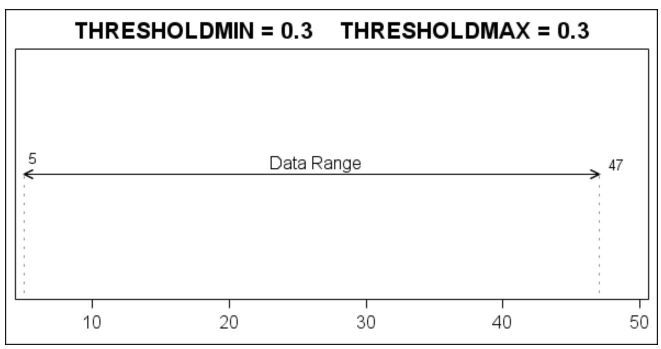
For LINEAR axes only, part of the default axis construction computes a small number of "good" tick values for the axis. This list might include "encompassing" tick values that go beyond the data range on both the lower or upper side of the axis. The THRESHOLDMIN= and THRESHOLDMAX= options of LINEAROPTS = ( ) can be used to establish rules for when to add encompassing tick marks. In the following example, the data range is 5 to 47. When the THRESHOLDMIN=0 and THRESHOLDMAX=0, the lowest and highest tick marks will always be at or inside the data range. Notice that the lowest tick mark is 10 and the highest tick mark is 40.



When the THRESHOLDMIN=1 and THRESHOLDMAX=1, the lowest and highest tick marks will always be at or outside the data range. Notice that the lowest tick mark is 0 and the highest tick mark is 50.



When the thresholds are set to any value between 0 and 1, a computation is performed to determine whether an encompassing tick is added. The default value for both thresholds is .3. Notice that the highest tick mark is 50 and the lowest tick mark is 10. In this case, an encompassing tick was added for the highest tick but not for the lowest tick.



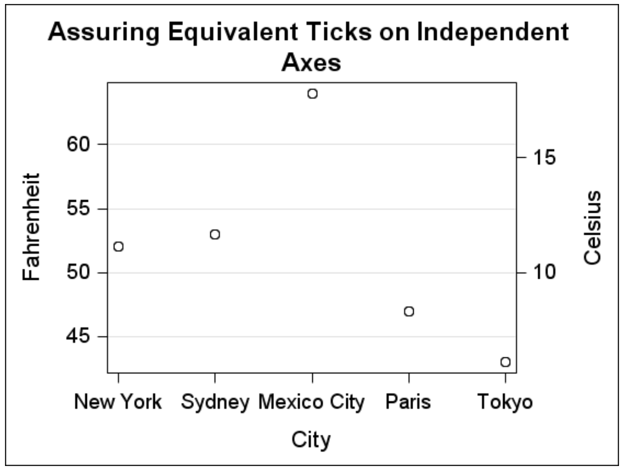
At the high end of the axis, there is a tick mark at 40. The THRESHOLDMAX= option determines whether a tick mark should be displayed at 50. The threshold distance is calculated by multiplying the THRESHOLDMAX= value (0.3) by the tick interval value (10), which equals 3. Measuring the threshold distance 3 down from 50 yields 47, so if the highest data value is between 47 and 50, a tick mark will be displayed at 50. In this case, the highest data value is 47 and it is within the threshold, so the tick mark at 50 is displayed.

At the low end of the axis, there is a tick mark at 10. The THRESHOLDMIN= option determines whether a tick mark should be displayed at 0. The threshold distance is calculated by multiplying the THRESHOLDMIN= value (0.3) by the tick interval value (10), which equals 3. Measuring the threshold distance of 3 up from 0 yields 3, so if the lowest data value is between 0 and 3, a tick mark will be displayed at 0. In this case, the lowest data value is 5 and it is not within this threshold, so the tick mark at 0 is not displayed.

Thresholds are important when you want the Y and Y2 (or X and X2) axes to have ticks marks located at equivalent locations on different scales. By preventing "encompassing" ticks from being drawn, you can ensure that the axis ranges for the two axes correctly align. The following example accepts the default minimum and maximum data values for each axis. Notice that the five scatter points for each plot are superimposed exactly.

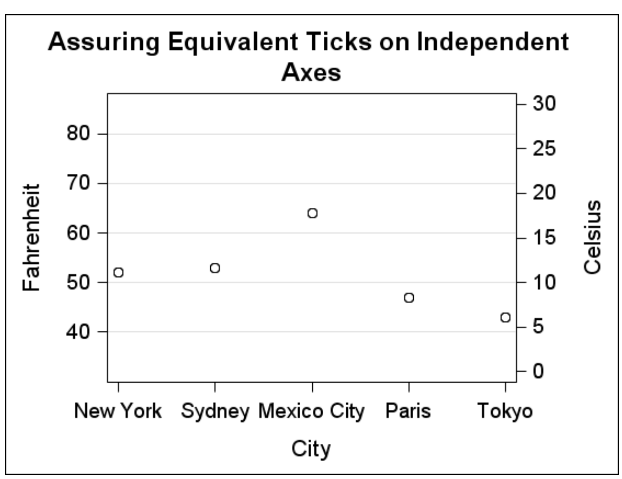
```
layout overlay /
  yaxisopts=(griddisplay=on
             linearopts=(integer=true thresholdmin=0 thresholdmax=0 ))

  y2axisopts=(linearopts=(integer=true thresholdmin=0 thresholdmax=0 ));
```

```
  scatterplot  x=city y=fahrenheit;
  scatterplot  x=city y=celsius / yaxis=y2;
endlayout;
```



In the following example, the axes have different but equivalent ranges that are established with the VIEWMIN= and VIEWMAX= options (32F <==> 0C and 86F <==> 30C).

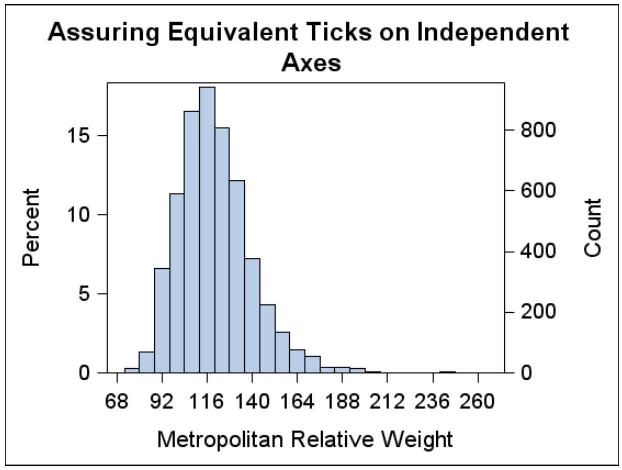
```
layout overlay /
    yaxisopts= (griddisplay=on
                linearopts=(integer=true thresholdmin=0 thresholdmax=0
                            viewmin=32 viewmax=86 ))

    y2axisopts= (linearopts=(integer=true thresholdmin=0 thresholdmax=0
                             viewmin=0 viewmax=30 ));

  scatterplot  x=City y=Fahrenheit;
  scatterplot  x=City y=Celsius / yaxis=y2;
endlayout;
```



This next example creates equivalent ticks for a computed histogram. We want to ensure that the percentage and actual count correspond on the Y and Y2 axes.

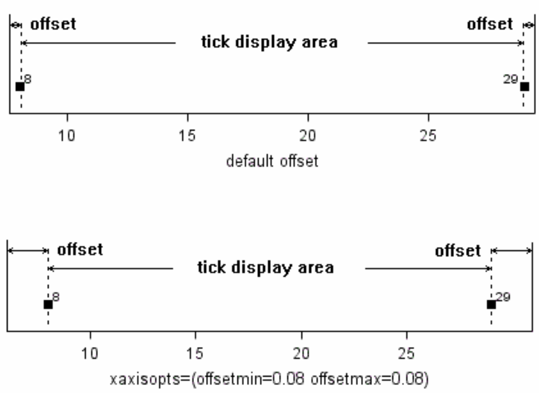
```
layout overlay / yaxisopts=(linearopts=( thresholdmin=0 thresholdmax=0 ))
    y2axisopts=(linearopts=( thresholdmin=0 thresholdmax=0 ));
  histogram mrw  / scale=percent;
```

```
  histogram mrw  / scale=count yaxis=y2;
endlayout;
```

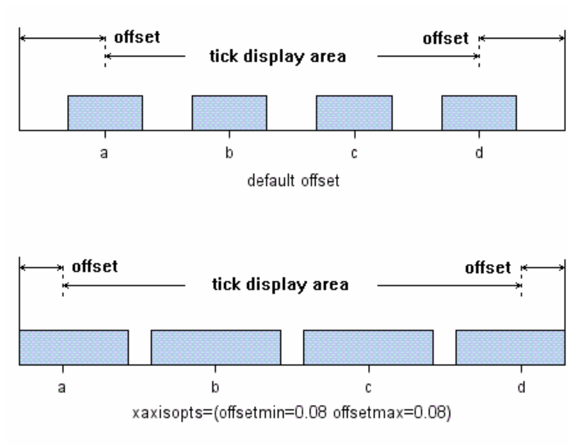


Apply Axis Offsets

In addition to axis thresholds, there are also axis offsets. Offsets are small gaps that are potentially added to each end of an axis: before the start of the data range and after the end of the data range. Offsets can be applied to any type of axis. For example, axis offsets are automatically added to allow for markers to appear at the first or last tick without clipping the marker



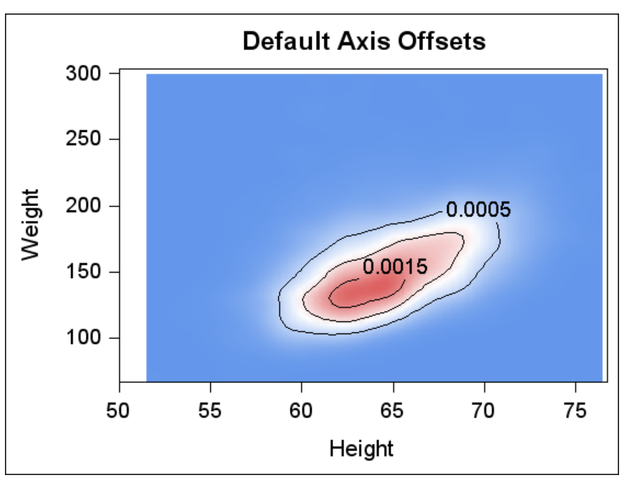
For plots such as box plots, histograms, and barcharts, offset space is added to ensure that the first and last box or bar does not get clipped.



The OFFSETMIN= option on a layout statement controls the distance from the beginning of the axis to the first tick mark (or minimum data value). The OFFSETMAX= option controls the distance between last tick (or maximum data value) and the end of the axis. The offset range is from 0 to 1, and the specified value is used to calculate the offset as a percentage of the full axis length. The default offset reserves just enough space to fully display markers and other graphical features near the ends of an axis.

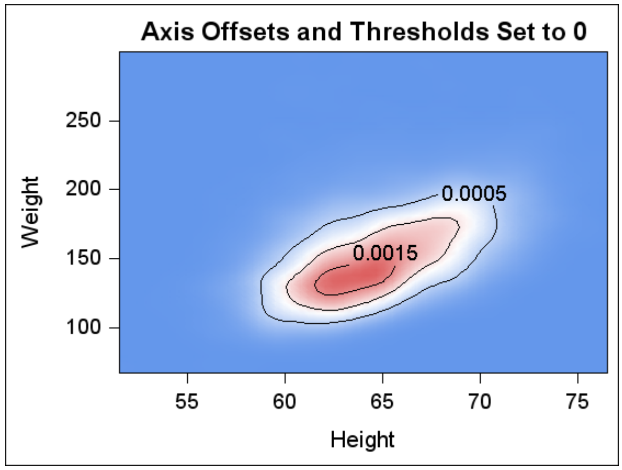
For some plots, the axis offsets are not desirable. To illustrate this, consider the contour plot below. Notice that the entire plot area between minimum and maximum data values is filled with colors that correspond to a Z value. The narrow white bands around the top and right edges of the filled area and the axis wall boundaries are due to the default axis offsets.

```
layout overlay;
  contourplotparm x=height y=weight z=density;
endlayout;
```



To eliminate the "extra" gaps at the ends of the axes, we can set axis offsets and thresholds to zero. An offset is a value between 0 and 1 that represents a percentage of the length of the axis.

```
layout overlay / xaxisopts=( offsetmin=0 offsetmax=0
                   linearopts=( thresholdmin=0 thresholdmax=0 ))
                 yaxisopts=( offsetmax=0 offsetmax=0
                   linearopts=( thresholdmin=0 thresholdmax=0 ));
  contourplotparm x=height y=weight z=density;
endlayout;
```

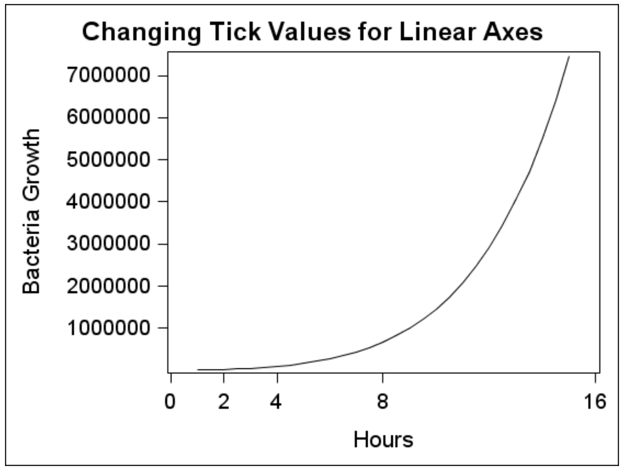


LINEAR Axes

Setting the Axis Data Range and Tick Values

For a LINEAR axis, you can set the tick values in several ways. If you use TICKVALUELIST = (*values*) or TICKVALUESEQUENCE= (*start-end increment*) syntax, the values that you specify will be used as long as those values are within the actual range of the data. Notice in the following example that the smallest and largest tick values on the Y axis are not what was requested because the Y-axis data range did not include 0 or 8000000. To extend (or reduce) the axis data range, you can use the VIEWMIN= and VIEWMAX= sub-options of the LINEAROPTS= option. Notice that because the X-axis was extended with these options, all the specified tick values were used. The X-axis also illustrates that the tick values do not have to be uniformly spaced. (Please note that choosing tick values in this manner does NOT create a log scale. See "LOG Axes" on page 75 for information about log axes.)

```
layout overlay /
    xaxisopts=(linearopts=( viewmin=0 viewmax=16
                tickvaluelist=(0 2 4 8 16)  ))
    yaxisopts=( linearopts=
               ( tickvaluesequence=(start=0 end=8e6 increment=1e6) ));
  seriesplot x=Hours y=Bacteria;

endlayout;
```



Formatting Axis Tick Values

Linear axes use special techniques that provide the generation of "good" tick values that are based on the data range. If a tick value format is not specified, the column formats provide a "hint" on how to represent the tick values, but those formats do not generally control the representation or precision of the tick values.

To force a given format to be used for a linear axis, you can use syntax similar to the following, where you specify any SAS numeric format:

```
linearopts=(tickvalueFormat= best6. )
```

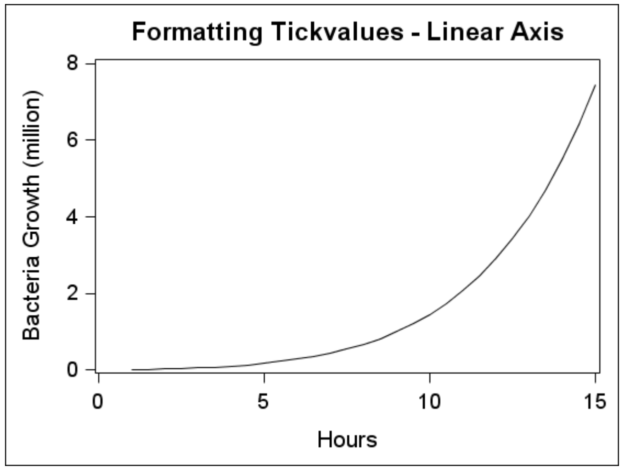
Note: GTL currently honors most but not every SAS format. For a list of supported formats, see Appendix A4, “SAS Formats Not Supported,” on page 357.

If you simply want the column format of the input data column to be directly used, specify the following:

```
linearopts=(tickvalueFormat=data)
```

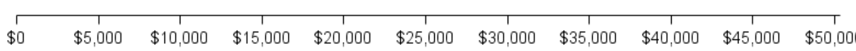
There are special options to control tick values. INTEGER=TRUE calculates good integers to use as tick values given the range of the data. EXTRACTSCALE=TRUE can be used to extract some factor of ten from all tick values in order to reduce the overall width of the tick values and improve legibility. The extracted factor is concatenated to the existing axis label. In the following example, a factor of 1000000 (million) is extracted from the Y-axis values and the text (million) is appended to the axis label.

```
layout overlay /  xaxisopts=(linearopts=(integer=true))
    yaxisopts=(linearopts=(tickvalueFormat= (extractScale=true) ));
  seriesplot x=Hours y=Bacteria;
endlayout;
```



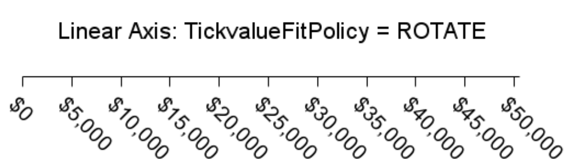
Avoiding Tick Value Collisions

Another intelligent feature that axes have is to change the display of tick values whenever the tick value text becomes too crowded. For example, the axis below comfortably shows eleven tick values:

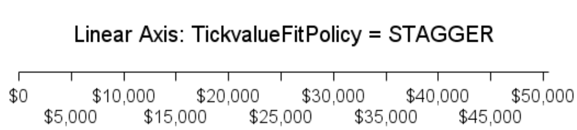


If the size of the graph decreases or the font size for the tick values increases, the axis ticks and tick values will automatically be "thinned" by removing alternating ticks and tick values. LINEAROPTS = (TICKVALUEFITPOLICY=THIN) is the default action for linear axes:

You can set TICKVALUEFITPOLICY=ROTATE which angles the tick value text 45 degrees:



You can set TICKVALUEFITPOLICY=STAGGER which creates alternating tick values on two rows.



You can set TICKVALUEFITPOLICY to a compound policy ROTATETHIN, STAGGERTHIN, or STAGGERROTATE. The compound policies attempt the second policy if the first policy does not work well. These policies are available for X and X2 axes. The only fit policy for the Y and Y2 axes is THIN.

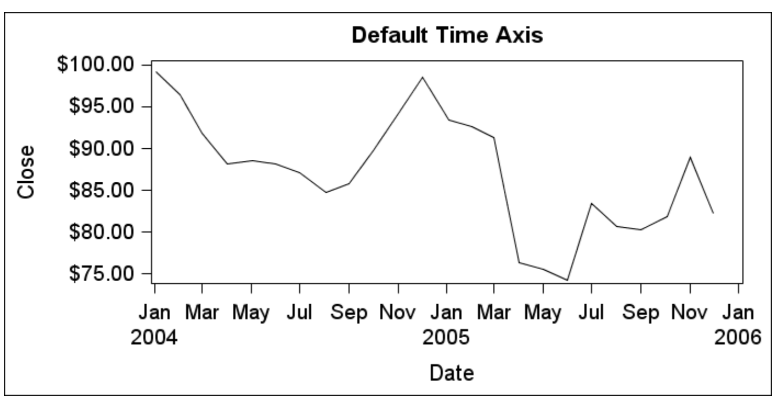
Note: The TICKVALUEFITPOLICY= is never applied unless a tick value collision situation is present. That is, you cannot force tick values to be rotated or staggered if there is no collision situation.

TIME Axes

Overview

TIME axes are numeric axes that display SAS date or time values in an intelligent way. Such axes are created whenever the primary plot has a SAS date, time, or datetime format associated with a column that is mapped to an axis. In the following example, the DATE variable has a SAS date format associated with it. By default, the TIME axis decides an appropriate tick value format and an interval to display. Notice that, in the default case, when the X or X2 axis is a TIME axis, the space that is used for the tick values is conserved by splitting the values at appropriate date or time intervals and extracting larger intervals. In this example, the column format for the DATE variable could be MMDDYY or any other date-type format. The actual format serves only as a hint and is not used directly, unless requested.

```
layout overlay;
  seriesplot x=date y=close;
endlayout;
```



Note: In this example, the data range for DATE was from 1Jan2004 to 1Dec2005. The TIME axis chose the interval of MONTH to display tick values. Had the data range been larger, say 1Jan1998 to 1Dec2005, the TIME axis would choose a larger interval, YEAR, to display by default.

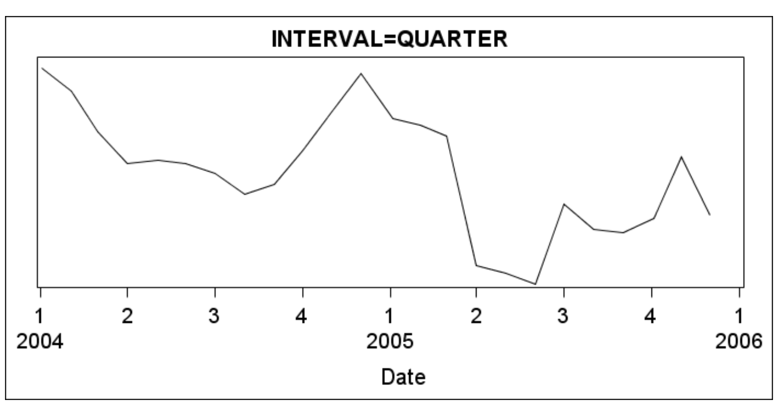
Setting the Tick Values

Using the INTERVAL= option, you can select different date or time intervals to display. The default interval is AUTO, which chooses an appropriate interval, based on the data and the column format.

Value on INTERVAL=	Unit	Tick Interval	Default Tick Value Format
AUTO	DATE, TIME, or DATETIME	automatically chosen	automatically chosen
SECOND	TIME or DATETIME	second	TIME8.
MINUTE	TIME or DATETIME	minute	TIME8.
HOUR	TIME or DATETIME	hour	TIME8.
DAY	DATE or DATETIME	day	TIME9.
TENDAY	DATE or DATETIME	ten days	TIME9.
WEEK	DATE or DATETIME	seven days	TIME9.
SEMIMONTH	DATE or DATETIME	1st and 16th of each month	TIME9.
MONTH	DATE or DATETIME	month	MONYY7.
QUARTER	DATE or DATETIME	three months	YYQC6.
SEMIYEAR	DATE or DATETIME	six months	MONYY7.
YEAR	DATE or DATETIME	year	YEAR4.

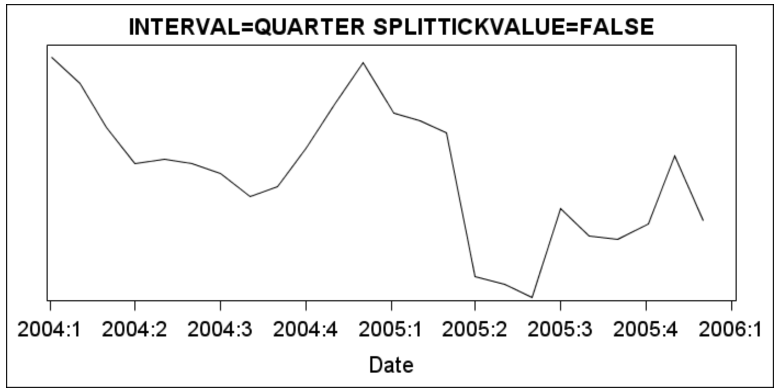
The following example specifies that tick values should occur at quarter intervals:

```
layout overlay / xaxisopts=( timeopts=(interval=quarter) );
```



You can turn off the splitting feature with the SPLITTICKVALUE=FALSE option. Notice that each tick value uses more space.

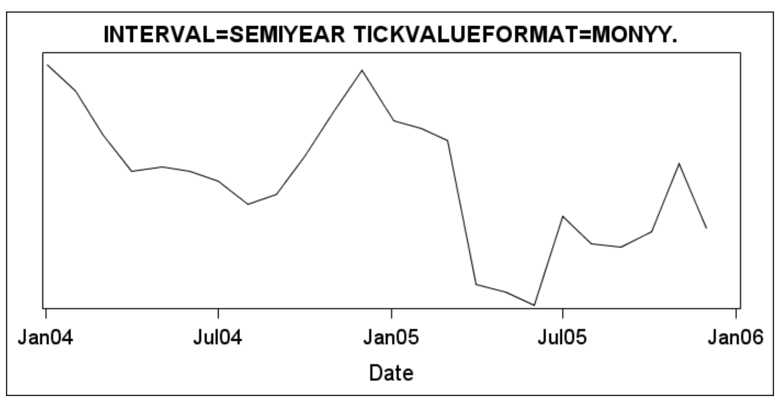
```
layout overlay / xaxisopts=(timeopts=(interval=quarter
                                      splittickvalue=false ));
```



Formatting Axis Tick Values

As with LINEAR axes, you can force a specific format for tick values with the TICKVALUEFORMAT= option, which also turns off the tick splitting feature. If you specify TICKVALUEFORMAT=DATA, the format is associated with the column that is used. Or you can specify a format:

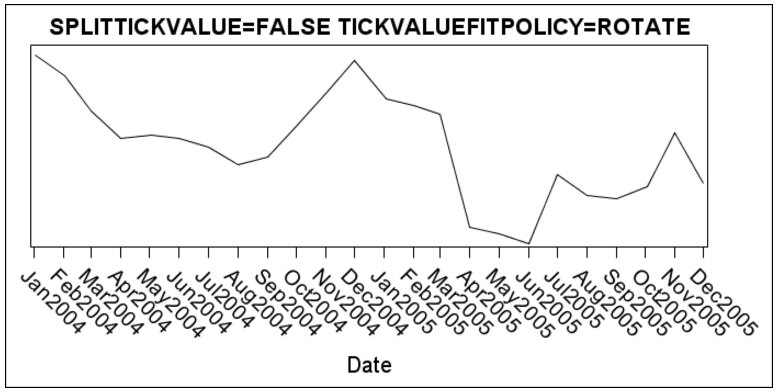
```
layout overlay / xaxisopts=(timeopts=(interval=semiyear
                                      tickvalueformat=monyy. ));
```



Avoiding Tick Value Collisions

As with LINEAR axes, you can specify a tick value fitting policy for a TIME axis. The following policies are available: THIN, ROTATE, STAGGER, ROTATETHIN, STAGGERTHIN, and STAGGERROTATE when tick values are not split. The default policy is THIN .

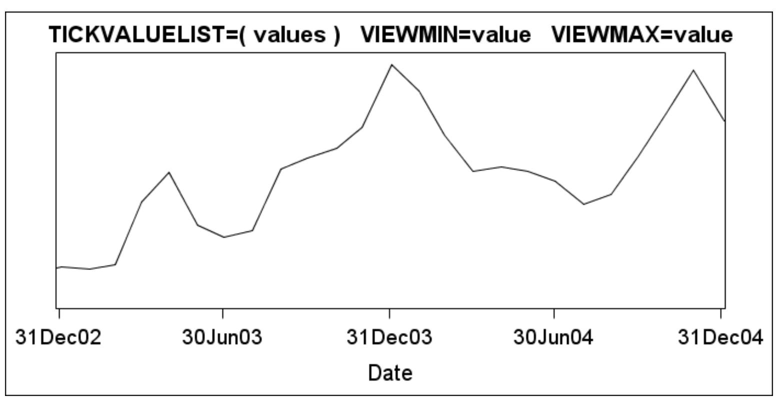
```
layout overlay / xaxisopts=(timeopts=(interval=month
                                      splittickvalue=false
                                      tickvaluefitpolicy=rotate ));
```



Setting the Axis Data Range

As with LINEAR axes, you can force specific tick values to be displayed with the TICKVALUELIST= option. The VIEWMIN= and VIEWMAX= options control the data range of the axis. If you specify TICKVALUEFORMAT=DATA, the format that is associated with the column is used.

```
layout overlay / xaxisopts=(timeopts=( tickvalueformat=data
  viewmin="31Dec2002"d viewmax="31Dec2004"d
  tickvaluelist=("31Dec2002"d "30Jun2003"d
                 "31Dec2003"d "30Jun2004"d "31Dec2004"d) ));
```

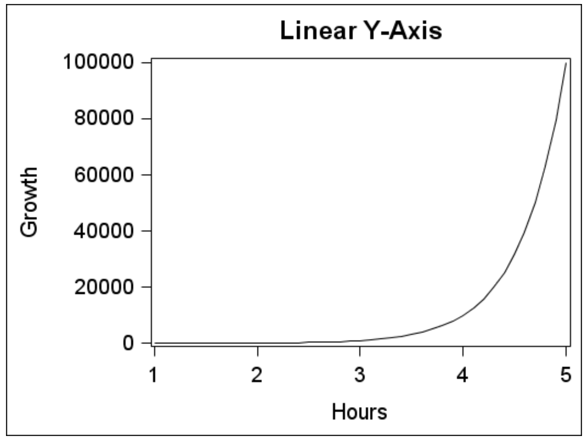


LOG Axes

Overview

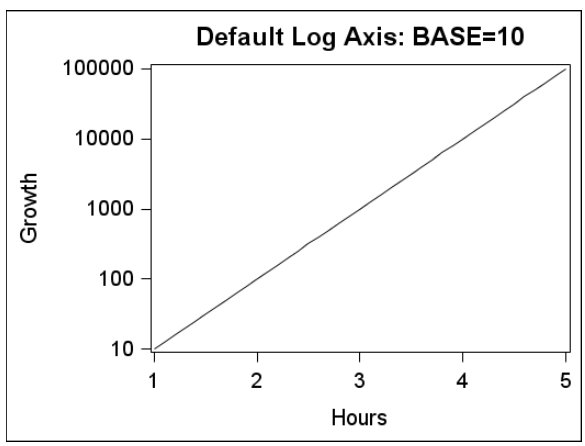
An axis displaying a logarithmic scale is very useful when your data values span orders of magnitude. For example, when you plot your growth data with a linear axis, you suspect that the growth rate is exponential.

```
layout overlay;
  seriesplot x=Hours y=Growth;
endlayout;
```



To confirm this, you can request a log axis, which is never drawn by default. Instead, you must request it with the TYPE=LOG axis option. Any of the four axes can be a log axis.

```
layout overlay / yaxisopts=(type=log);
```



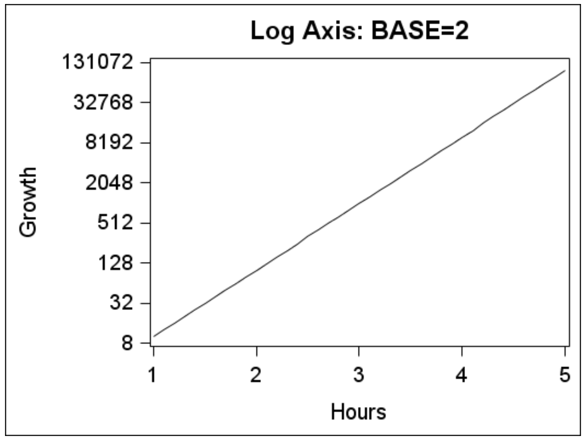
The numeric data that is used for a log axis must be positive. If zero or negative values are encountered, a linear axis is substituted and the following note is written to the log:

```
NOTE: Log axis cannot support zero or negative values in the data range. The axis
type will
      be changed to LINEAR.
```

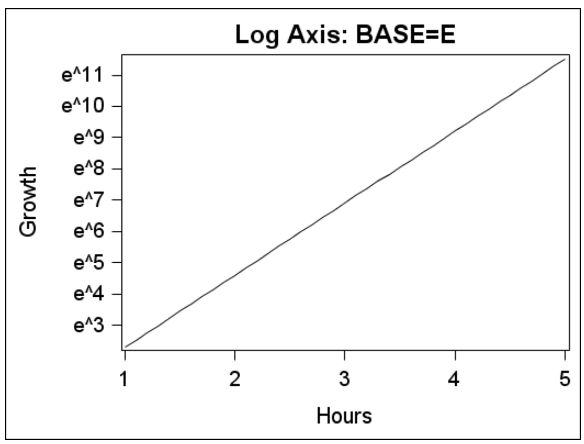
Setting the Log Base

You can show a log axis with any of three bases: 10, 2 and E (natural log). The default log base is 10. To set another base, use the BASE= *suboption* setting of the LOGOPTS= option.

```
layout overlay / yaxisopts=(type=log logopts=(base=2) );
```




```
layout overlay / yaxisopts=(type=log logopts=(base=e) );
```



Setting the Tick Intervals

Log axes support the TICKINTERVALSTYLE= option, which provides different styles for displaying tick values:

AUTO
: A LOGEXPAND, LOGEXPONENT, or LINEAR representation is chosen automatically, based on the range of the data. When the data range is small (within an order of magnitude), a LINEAR representation is typically used. Data ranges that encompass several orders of magnitude typically use the LOGEXPAND or LOGEXPONENT representation. AUTO is the default.

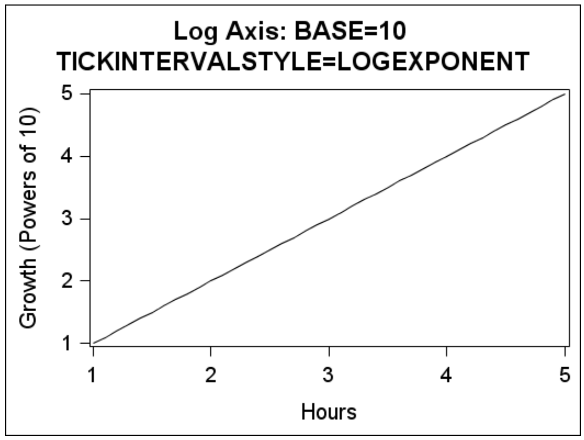
LOGEXPAND
: Major ticks are placed at uniform intervals at integer powers of the base. By default, a BEST6. format is applied to BASE=10 and BASE=2 tick values. This means that, depending on the range of data values, you might see very large or very small values written in exponential notation (10E6 instead of 1000000). The preceding examples with a log axis show TICKINTERVALSYTLE=LOGEXPAND.

LOGEXPONENT
: Major ticks are placed at uniform intervals at integer powers of the base. The tick values are only the integer exponents for all bases.

LINEAR
: Major tick marks are placed at non-uniform intervals, covering the range of the data.

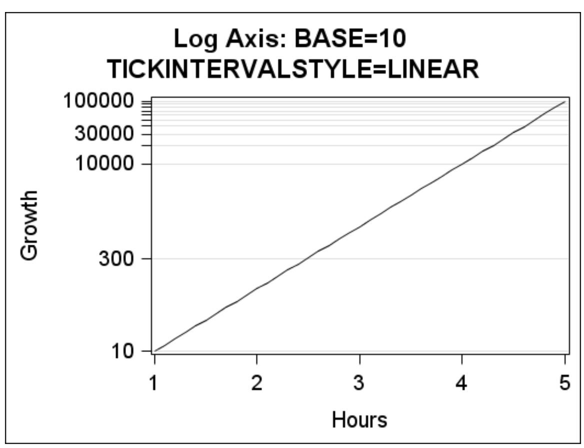
When using TICKINTERVALSTYLE=LOGEXPONENT, it might not be clear what base is being used. You should consider adding information to the axis label to clarify the situation:

```
layout overlay / yaxisopts=(type=log label="Growth (Powers of 10)"
  logopts=(base=10 tickintervalstyle=logexponent));
```



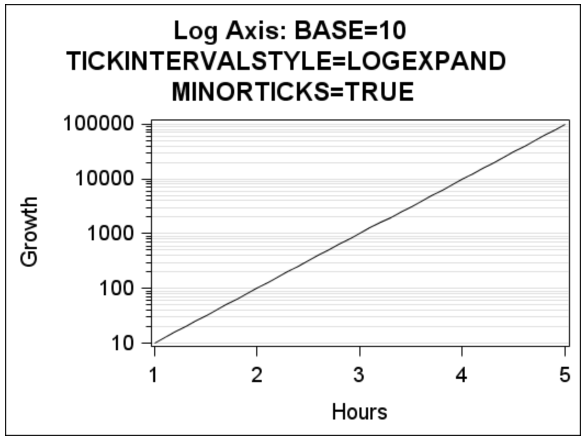
When using TICKINTERVALSTYLE=LINEAR, it is visually helpful to turn on the grid lines:

```
layout overlay / yaxisopts=(type=log  griddisplay=on
  logopts=(base=10 tickintervalstyle=linear));
```



When using BASE=10 and TICKINTERVALSTYLE=LOGEXPAND or TICKINTERVALSTYLE=LOGEXPONENT, you can add minor ticks to emphasize the log scale:

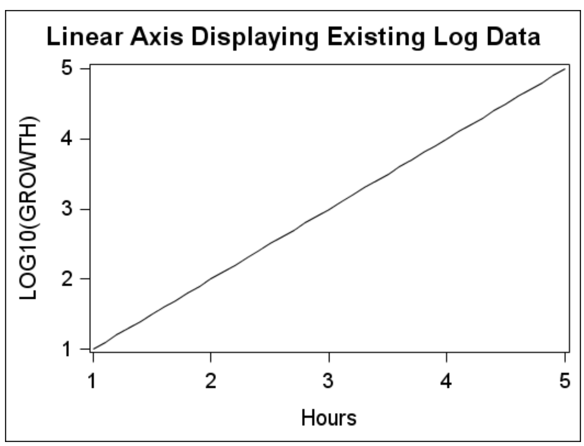
```
layout overlay / yaxisopts=(type=log griddisplay=on
  logopts=(base=10 tickintervalstyle=linear minorticks=true ));
```



As with LINEAR and TIME axes, the data range of a log axis can be set with the VIEWMIN= and VIEWMAX= log options.

If your input data has already been transformed into log values, you should always use a LINEAR axis to display them, not a LOG axis.

```
layout overlay;
  seriesplot x=Hours y=eval(log10(growth));
endlayout;
```



Axis Line versus Wall Outline

The area bounded by the X, Y, X2, and Y2 axes is called the Wall Area or simply the Wall. The wall consists of a filled area (FILL) and a boundary line (OUTLINE). The display of the Wall is independent of the display of axes. When both are displayed, the axes are placed on top of the wall outline. Most frequently, your plots will use only the X and Y axes, not X2 or Y2.

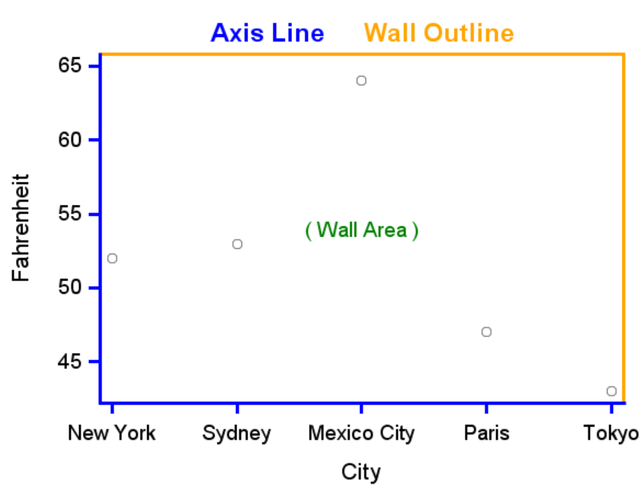
By default, you will see lines that look like X2 and Y2 axis lines, but they are not axis lines. They are the lines of the wall outline, which happens to be the same color and thickness as the axis lines. This can be made apparent by assigning different visual properties to the wall outline and the axis lines.

The GraphAxisLines style element controls the appearance of all axis lines, and the GraphWalls style element controls the wall. The following example shows how you can change the appearance of the axes and wall with a style definition. In the template code, the PROC TEMPLATE block defines a style named AXIS_WALL, and then the ODS LISTING statements sets the AXIS_WALL style as the active style for output that is directed to the LISTING destination:

```
proc template ;
 define style axis_wall;
 parent=styles.listing;
 style graphwalls from graphwalls /
   frameborder=on
   linestyle=1
   linethickness=2px
   backgroundcolor=GraphColors("gwalls")
   contrastcolor= orange;
 style graphaxislines from graphaxislines /
   linestyle=1
   linethickness=2px
   contrastcolor=blue;
 end;
run;

ods listing style=axis_wall ;
```

If a simple GTL template containing the following layout block is executed while the AXIS_WALL style is in effect, you would be able to see that the axis lines are distinct from the wall outlines:

```
layout overlay / walldisplay=(fill outline); /* default walldisplay */
  scatterplot  x=City y=Fahrenheit / datatransparency=.5;
  entry textattrs=(color=green) "( Wall Area )";
endlayout;
```



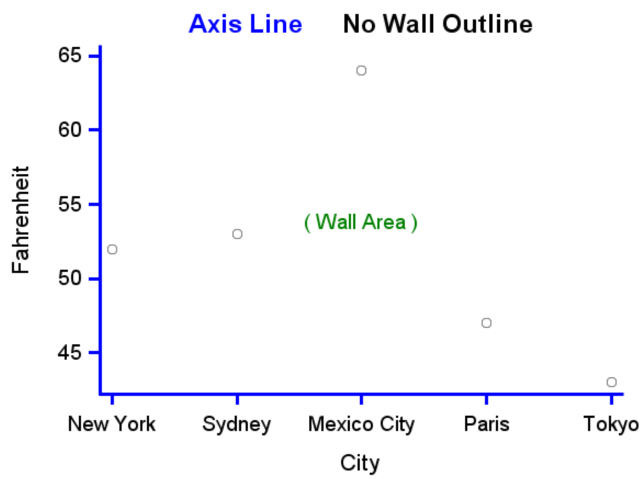
Most styles set the axis lines and the wall outline to be the same color, line pattern, and thickness, so it is impossible to see the difference. Sometimes you might not want to see the wall outline, or you might want to change the wall color. These types of changes can be set on a style or with the WALLDISPLAY= option on the LAYOUT OVERLAY statement. For example, the GTL default for the wall is WALLDISPLAY=(FILL OUTLINE).

The following code fragment shows how to use the style definition to turn off the wall outline:

```
style GraphWalls from GraphWalls /
   frameborder=off;
```

This next code fragment shows how to use GTL to turn off the wall outline:

```
layout overlay / walldisplay=(fill);
```



Axis Appearance Features Controlled by the Current Style

The appearance of graphs produced with GTL is always affected by the ODS style that is in effect for the ODS destination. From an axis perspective, the default appearance of the axis line, ticks, tick values, axis label, and grid lines are controlled by predefined style elements.

Style Element	Style Attributes	Values	Controls
GraphAxisLines	TickDisplay	"ACROSS" "INSIDE" "OUTSIDE"	Tick mark location
	LineStyle	Integer: 1 to 49	Axis line pattern
	LineThickness	Dimension	Axis line and tick thickness
	ContrastColor	Color	Axis line and tick color
GraphGridlines	DisplayOpts	"AUTO" "ON" "OFF"	When to display grid lines
	LineStyle	Integer: 1 to 49	Grid line pattern
* A style font-specification include attributes for FONTFAMILY, FONTWEIGHT, FONTSTYLE, and FONTSIZE			

Style Element	Style Attributes	Values	Controls
	LineThickness	Dimension	Grid line thickness
	ContrastColor	Color	Grid line color
GraphLabelText	Color	Color	Axis label text color
	Font	font-specification *	Axis label font
GraphValueText	Color	Color	Axis tick value text color
	Font	font-specification *	Axis tick value font
* A style font-specification include attributes for FONTFAMILY, FONTWEIGHT, FONTSTYLE, and FONTSIZE			

The following GTL axis options also control the appearance of axis features. When you include these options, the corresponding information from the current style is overridden.

Option	Overrides ...
GRIDDISPLAY=	DisplayOpts attribute of GraphGridLines
GRIDATTRS=	GraphGridLines
LABELATTRS=	GraphLabelText
TICKVALUEATTRS=	GraphValueText
TICKSTYLE=	TickDisplay attribute of GraphAxisLines

Example: Assure that the axis label text appears in bold.

```
layout overlay / xaxisopts=(labelattrs=(weight=bold))
                 yaxisopts=(labelattrs=(weight=bold));
```

Example: Display grid lines.

```
layout overlay / xaxisopts=(griddisplay=on)
                 yaxisopts=(griddisplay=on);
```

Example: Use a dot pattern for grid lines.

```
layout overlay / xaxisopts=(griddisplay=on gridattrs=(pattern=dot))
                 yaxisopts=(griddisplay=on gridattrs=(pattern=dot));
```

Example: Make ticks cross the axes lines.

```
layout overlay / xaxisopts=(tickstyle=across)
                 yaxisopts=(tickstyle=across);
```

For all of the preceding examples, you would add similar coding to the X2AXISOPTS= and Y2AXISOPTS= options if the X2 or Y2 axes are used as independent scales. For complete documentation on the axis options that are available, see the *SAS/GRAPH Graph Template Language Reference*.

Chapter 6

Managing Graph Appearance: General Principles

Default Appearance Features in Graphs

Graphs that are produced with GTL derive their general default appearance features (fonts, colors, line properties, and marker properties) from the current ODS style. The following three images show the same graph that is rendered with three different styles.

Figure 6.1 *ods listing style=default;*

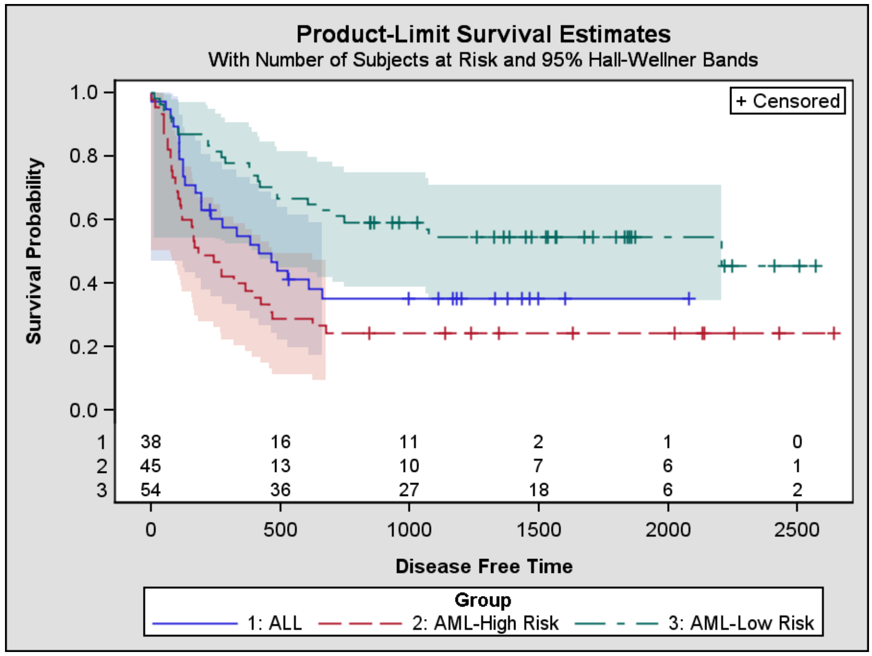


Figure 6.2 *ods listing style=astronomy;*

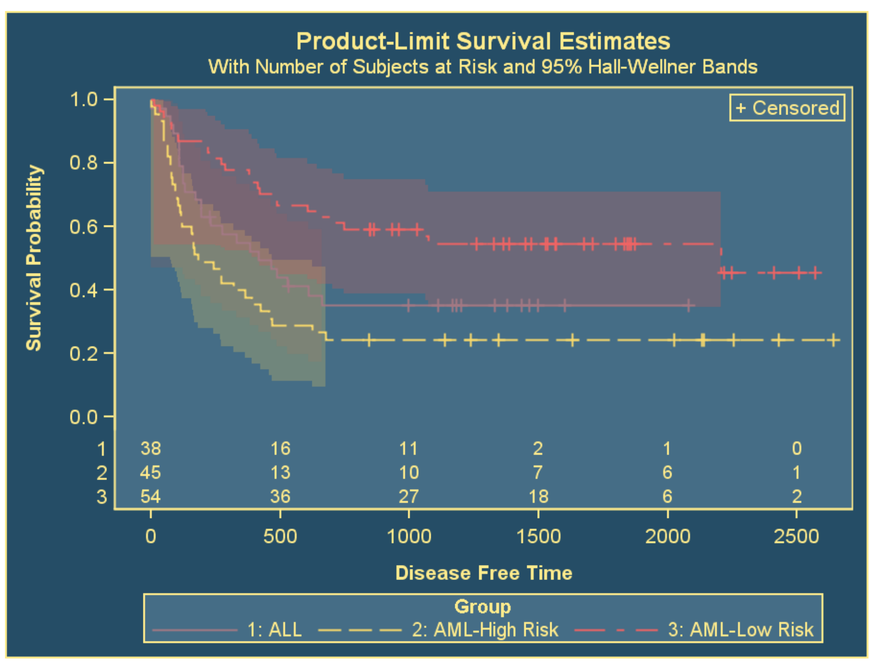
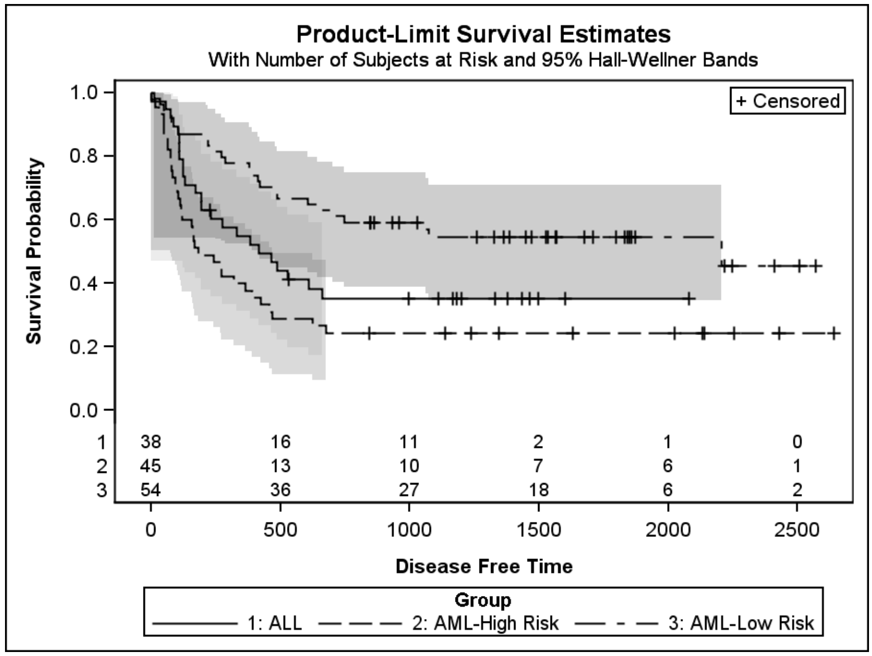


Figure 6.3 *ods listing style=journal;*



An important point to note, here, is that the appearance of the graph changes when the template is executed, not when it is compiled.

Fully one third of all GTL syntax addresses matters of appearance. Yet, most of the examples in this document do not use the appearance syntax because the examples take advantage of the pre-defined styles. Whenever the options in your graph template explicitly change a color or font family, you are locking those decisions into the compiled template. Appearance options in GTL always override any similar appearance settings contained in the style. Thus, setting a fixed font or color appearance option might yield satisfactory results with some styles but not with others. For that reason, the compiled graph and table templates that are included with many SAS procedures do not contain references to fixed fonts and colors.

This chapter shows "best practices" to follow so that your GTL programs integrate style definitions to create the look you desire in your graphics output. The coding strategy that you use will depend on how much style integration you need. If you want to change the appearance of all your graphs or apply a custom style to them, you can define your own style. For details, see Chapter 17, “Managing the Graph Appearance with Styles,” on page 299.

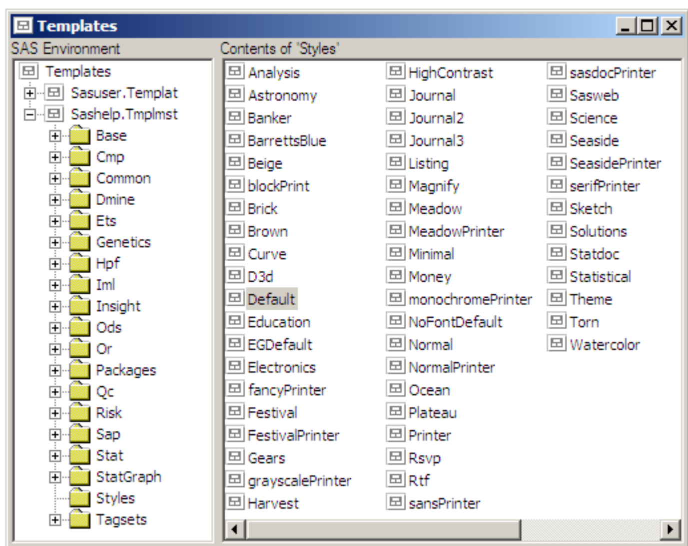
Evaluating Supplied Styles

Over fifty ODS styles are available for use with GTL graphs. These styles are stored in the SASHELP.TMPLMST item store under the STYLES directory. To list the names of all the supplied templates in the SAS Output window, you can submit the following program:

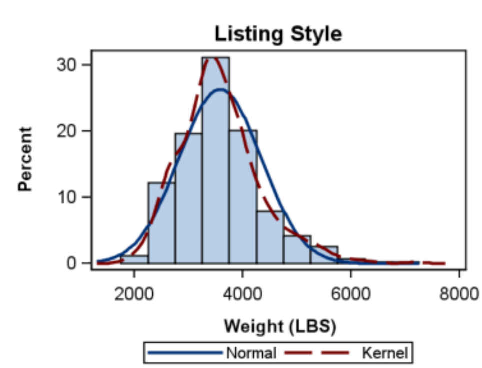
```
proc template;
 path sashelp.tmplmst;
 list styles;
run;
```

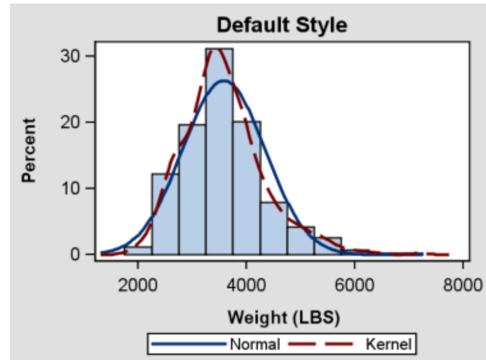
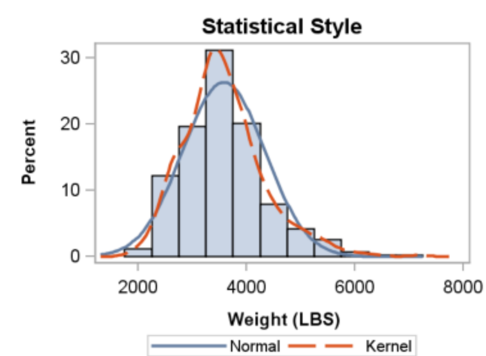
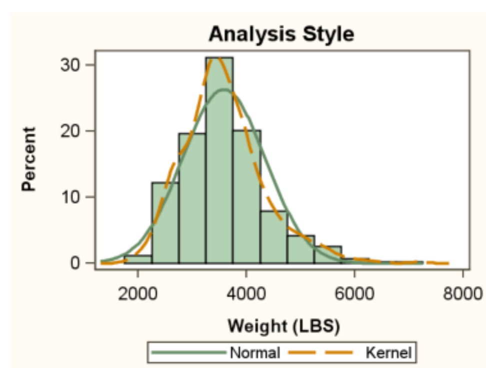
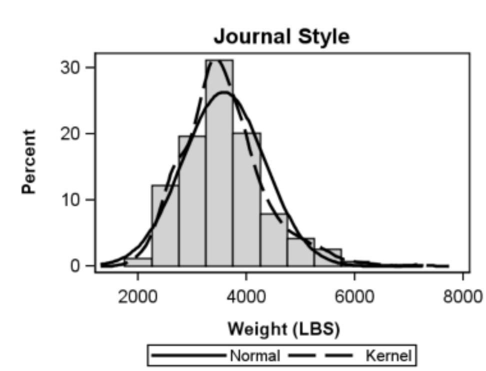
```
Listing of: SASHELP.TMPLMST
Path Filter is: Styles
Sort by: PATH/ASCENDING

Obs    Path                         Type
_________________________________________
 1     Styles                       Dir
 2     Styles.Analysis              Style
 3     Styles.Astronomy             Style
 4     Styles.Banker                Style
 (more )
```

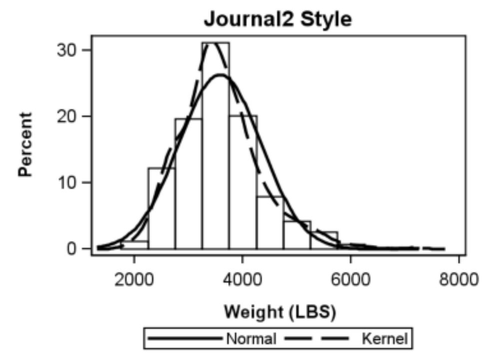
You can also browse the styles interactively using the Templates window. To do so, issue the ODSTEMPLATE command to open the Templates window, and then select STYLES under the SASHELP.TMPLMST item store.



Some of the ODS styles have been around for a long time, before the introduction of ODS Graphics. All styles will work with ODS Graphics, but many of the older ones have not been fully optimized for ODS Graphics. Below is a list of recommended styles and a brief description of each.

Style	**Example**
LISTING • white background • white wall • sans-serif fonts • color used for lines, markers, and filled areas • other colors the same as DEFAULT style	

<table>
<tr><th>Style</th><th>Example</th></tr>
<tr><td>DEFAULT
• gray background
• white wall
• sans-serif fonts</td><td></td></tr>
<tr><td>STATISTICAL
• white background
• white wall
• sans-serif fonts
• contrasting color scheme of blues, reds, greens for markers, lines, and filled areas</td><td></td></tr>
<tr><td>ANALYSIS
• light tan background
• white wall
• sans-serif fonts
• muted color scheme of tans, greens, yellows, oranges and browns for lines, markers, and filled areas</td><td></td></tr>
<tr><td>JOURNAL
• white background
• white wall
• sans-serif fonts
• gray-scale color scheme for markers, lines, and filled areas</td><td></td></tr>
</table>

Style	Example
JOURNAL2 • white background • white wall • sans-serif fonts • black-only scheme for markers and lines • no filled areas—a minimal ink style	

Attributes as Collections of Related Options

Overview

In GTL, the syntax for explicitly setting the properties of a graphical feature is a list of *name-value* pairs that is enclosed in parentheses. For example, to set the X-axis properties, you use the following:

XAXISOPTS=(LABEL="*string*" TYPE=*axis-type …*)

The syntax for setting appearance options is similar. For example, statements such as SERIESPLOT, DENSITYPLOT, REFERENCELINE, DROPLINE, and several others have a LINEATTRS= option:

LINEATTRS= (COLOR=*color* PATTERN=*line-pattern* THICKNESS=*line-thickness*)

As a matter of fact, the properties of any line that you can draw in GTL are specified in exactly the same way, possibly with a different option keyword.

For BANDPLOT and MODELBAND statements, you would use the following option:

OUTLINEATTRS=(COLOR=*color* PATTERN=*line-pattern* THICKNESS=*line-thickness*)

For the BOXPLOT statement, you could use either of the following options:

WHISKERATTRS=(COLOR=*color* PATTERN=*line-pattern* THICKNESS=*line-thickness*)

MEDIANATTRS=(COLOR=*color* PATTERN=*line-pattern* THICKNESS=*line-thickness*)

The list of common options is sometimes called an "attribute bundle." An interesting feature of attribute bundles for appearance-related options is that there are several ways of setting the values of the sub-options:

LINEATTRS Option

The following syntax is the complete syntax for the LINEATTRS= option:

LINEATTRS = *style-element* | *style-element* (*line-options*) | (*line-options*)

By default, a *style-element* is used for the LINEATTRS= setting. For the REFERENCELINE and DROPLINE statements, the default style element is the GraphReference element. What exactly does this mean?

If we look up the GraphReference element in the DEFAULT style (see Appendix A2, "SAS Graph Style Elements for GTL," on page 345 for a complete list of all elements and attributes and their defaults), we find the following:

```
style GraphReference /
  linethickness = 1px
  linestyle = 1
  contrastcolor = GraphColors('greferencelines');
```

This definition is ODS style syntax for an attribute bundle. The following table shows how this definition's style attributes map to GTL options.

Style Attribute	Description	GTL Suboption	Description
LINETHICKNESS	dimension, most often pixels	THICKNESS	dimension, most often pixels
LINESTYLE	numeric; 1 to 46, 1 being a solid line	PATTERN	either 1 to 46 or a pattern name, such as SOLID, DASH, DOT (see "Values for Line Patterns" on page 355 for examples of available line patterns)
CONTRASTCOLOR	color specification	COLOR	color specification

The default specification for REFERENCELINE and DROPLINE statements is **`LINEATTRS=GraphReference`**, which is a shortcut meaning "initialize the three GTL line properties with the corresponding attributes that are defined in a style element." This can be explicitly expressed in GTL as follows:

```
LINEATTRS=( PATTERN   = GraphReference:LineStyle
            THICKNESS= GraphReference:LineThickness
            COLOR     = GraphReference:ContrastColor )
```

In GTL, a style reference is a construct of the form *style-element* : *style-attribute*. This convention is the way to refer to a specific style attribute of a specific style element.

First of all, let's look at what it means to use a different style element for the LINEATTRS= option.

When selecting a different style element, you should make sure that the style element does set line properties (graph style elements do not necessarily define all possible attributes). Some reasonable choices might be GraphDataDefault, GraphAxisLines, GraphGridLines. and GraphBorderLines. You might choose GraphGridLines to force a reference line to match the properties of grid lines (if displayed). When you make this kind of assignment, you really don't know what actual line properties will be used because they might change, depending on how a given style is defined. What you should be confident of is that the grid lines and reference lines will be identical in terms of line properties.

Now let's assume that you want reference lines to be somewhat like a style element, but nevertheless different. This involves an override. Here are some examples:

```
1) LINEATTRS=GraphGridLines(THICKNESS=2px)
2) LINEATTRS=GraphAxisLines(PATTERN=DASH)
3) LINEATTRS=GraphReference(COLOR=GraphAxisLines:ContrastColor)
4) LINEATTRS=(COLOR=GraphAxisLines:ContrastColor)
5) LINEATTRS=(COLOR=BLUE)
```

In example 1, the reference line will look like a grid line (color and pattern), but be thicker (assuming most styles define grid lines as 1px).

In example 2, the reference line will look like an axis line (color and thickness), but it will use the DASH pattern.

In example 3, the reference line will look like a reference line (pattern and thickness), but it will have the color of axis lines.

Example 4 is a short form for example 3. Any time that you don't supply a style element or don’t override all the sub-options, the sub-options not overridden come from the default style references.

Example 5 shows how you can "hard-code" visual properties. This technique is a straightforward way of getting what you want. The results might look good when the DEFAULT or LISTING styles are in effect, but might not look good when the ANALYSIS style is in effect because ANALYSIS does not use any blues in its color scheme.

When specifying the attributes for a line, the available *line-options* can be any one or more of the following settings. The options must be enclosed in parentheses, and each option is specified as a *name* = *value* pair. In all cases, the value can be a *style-reference* in the form *style-element*:*style-attribute* (see Example 3).

COLOR= *style-reference* | *color*
: specifies the line color. If you use a *style-reference*, the *style-attribute* should be a valid attribute, such as COLOR, CONTRASTCOLOR, STARTCOLOR, NEUTRAL, or ENDCOLOR. The convention is to use CONTRASTCOLOR for lines.

PATTERN=*style-reference* | *line-pattern-name* | *line-pattern-number*
: specifies the line pattern. If you use a *style-reference*, the *style-attribute* should be LINESTYLE. Line patterns can be specified as a pattern name or pattern number. See Appendix A3, “Values for Marker Symbols and Line Patterns,” on page 355 for a list of all possible line patterns.

THICKNESS=*style-reference* | *dimension*
: specifies the line thickness. If you use a *style-reference*, the *style-attribute* should be LINETHICKNESS.

MARKERATTRS Option

Much of what is said about line properties in “LINEATTRS Option” on page 88 also applies to marker properties. Some plot statements, such as SERIESPLOT, display a line and can display markers. In those cases, you should use the DISPLAY=(MARKERS) option to turn on the marker display, and also use the MARKERATTRS= option to control the appearance of markers. (The BOXPLOT statement uses OUTLIERATTRS= and MEANATTRS= options).

The following syntax is the complete syntax for the MARKERATTRS= option:

MARKERATTRS=*style-element* | *style-element* (*marker-options*) | (*marker-options*)

The following *marker-options* are available:

COLOR= *style-reference* | *color*

SYMBOL= *style-reference* | *marker-name*

SIZE= *style-reference* | *marker-size*

The following table shows how MARKERATTRS= style attributes map to GTL options.

Style Attribute	Description	GTL Suboption	Description
CONTRASTCOLOR	color specification	COLOR	color specification
MARKERSIZE	dimension, most often pixels	SIZE	dimension, most often pixels
MARKERSYMBOL	string—for example, "circle" or "square"	SYMBOL	predefined keywords such a CIRCLE, SQUARE, TRIANGLE
none		WEIGHT	NORMAL or BOLD

COLOR= *style-reference* | *color*
: specifies the line color. If you use a *style-reference*, the *style-attribute* should be a valid attribute such as COLOR, CONTRASTCOLOR, STARTCOLOR, NEUTRAL, or ENDCOLOR. The convention is to use CONTRASTCOLOR for lines.

SYMBOL=*style-reference* | *marker-name*
: specifies the marker symbol. If you use a *style-reference*, the *style-attribute* should be MARKERSYMBOL. Markers are specified by keywords. See Appendix A3, "Values for Marker Symbols and Line Patterns," on page 355 for a list of all possible markers and their keywords.

SIZE= *style-reference* | *dimension*
: specifies the marker size. If you use a *style-reference*, the *style-attribute* should be MARKERSIZE.

WEIGHT = NORMAL | BOLD
: specifies the marker weight. NORMAL is the default. BOLD makes markers appear heavier or denser.

TEXTATTRS Option

The appearance of all text that appears in a graph can be controlled by the style or with GTL syntax. Title and footnote text in a graph is specified with the ENTRYTITLE and ENTRYFOOTNOTE statements. One or more lines of text can be displayed in the plot area by using one or more ENTRY statements. Each of these statements provides the TEXTATTRS= option for controlling the appearance of that text.

The following syntax is the complete syntax for the TEXTATTRS= option:

TEXTATTRS=*style-element* | *style-element* (*text-options*) | (*text-options*)

Most often the TEXTATTRS=(*text-options*) settings are used to control text font and color properties.

Text can also be specified on numerous options that are available on plot statements and layout statements, and also on various axis options. For example, most plot statements that can display a line provide the CURVELABEL= for labeling the line. Axis options that are available for the layout statements provide the LABEL= option for specifying an axis label. The default appearance of the text in these cases is controlled by styles, but GTL syntax provides the CURVELABELATTRS= and LABELATTRS= options for overriding the defaults. The syntax and use for these options is similar to that of the TEXTATTRS= option. For example, the following syntax is the complete syntax for the LABELATTRS= option:

LABELATTRS=*style-element* | *style-element* (*text-options*) | (*text-options*)

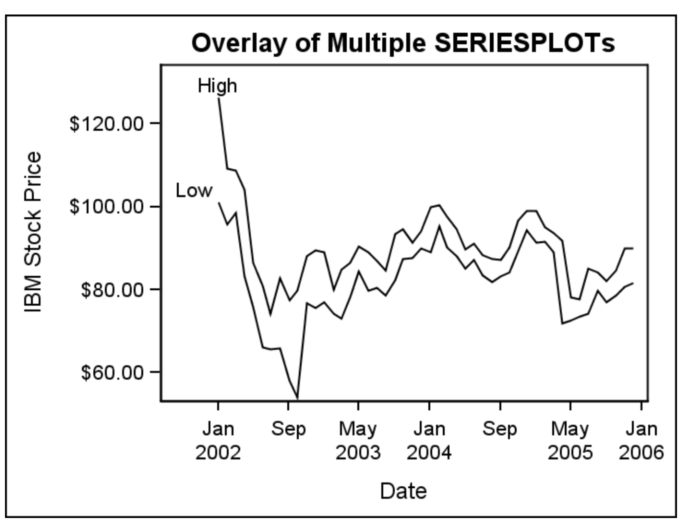
Changing text attributes is fully discussed in Chapter 7, "Adding and Changing Text in a Graph," on page 101.

Appearance of Non-grouped Data

When you use statements such as SERIESPLOT, BANDPLOT, NEEDLEPLOT, ELLIPSE, STEPPLOT, FRINGEPLOT, LINEPARM, and VECTORPLOT to draw plots containing lines, the same style element, GraphDataDefault, is used for all line and marker properties. You can think of these plots as "non-specialized," and they all have the same default appearance when used in overlays

In the graph that is produced by the following code, the series lines have the same default appearance.

```
proc template;
  define statgraph series;
    begingraph;
      entrytitle "Overlay of Multiple SERIESPLOTs";
      layout overlay / yaxisopts=(label="IBM Stock Price");
        seriesplot x=date y=high / curvelabel="High";
        seriesplot x=date y=low /  curvelabel="Low";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.stocks template=series;
   where date between "1jan2002"d and "31dec2005"d
         and stock="IBM";
run;
```

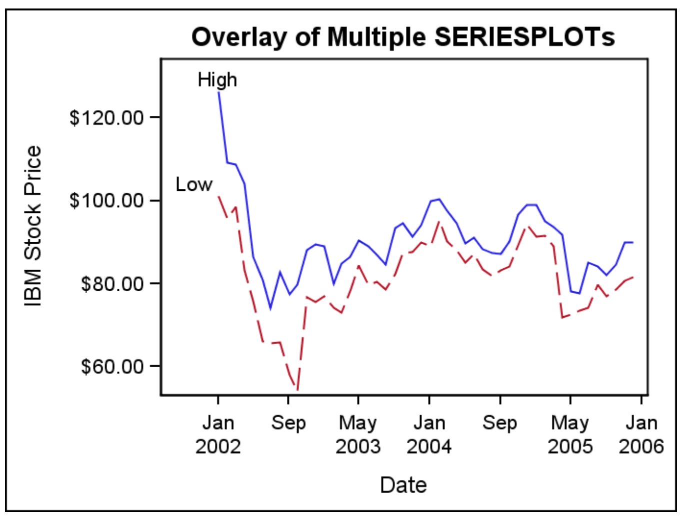


To ensure that the series lines differ in appearance, you can use any style element with line properties. A set of carefully constructed style elements named GraphData1 to GraphDataN (where N=12 for most styles, some styles might have fewer) are normally used for this purpose. These elements all use different marker symbols, line pattern, fill colors (COLOR=) and line/marker colors (CONTRASTCOLOR=). All line/marker colors are of different hues but with the same brightness, which means that all twelve colors can be distinguished but none stands out more than another. Fill colors are based on the same hue but have less saturation, making them similar but more muted than the corresponding contrast colors.

In the following template code, the style elements GraphData1 and GraphData2 are used to change the default appearance of the series lines in the graph.

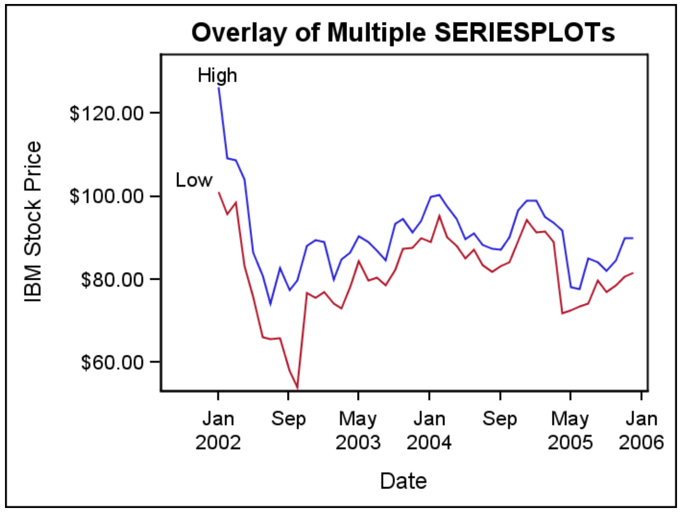
```
layout overlay / yaxisopts=(label="IBM Stock Price");
  seriesplot x=date y=high / curvelabel="High" lineattrs=GraphData1 ;
  seriesplot x=date y=low /  curvelabel="Low"  lineattrs=GraphData2 ;
endlayout;
```

Note: This same graph could also have been achieved by specifying CYCLEATTRS=TRUE on the LAYOUT OVERLAY statement and omitting the LINEATTRS= options on the plot statements.



By default, the GraphDataN style elements can be used interchangeably to achieve visual distinction. All of these elements vary color, line pattern, and marker symbols to gain maximum differentiation. Sometimes, you might not want to vary all properties at once. For example, to force only the color to change but not the line pattern, you can override one or more properties you want to hold constant.

```
layout overlay / yaxisopts=(label="IBM Stock Price");
  seriesplot x=date y=high / curvelabel="High"
    lineattrs=GraphData1(pattern=solid) ;
  seriesplot x=date y=low / curvelabel="Low"
    lineattrs=GraphData2(pattern=solid) ;
endlayout;
```

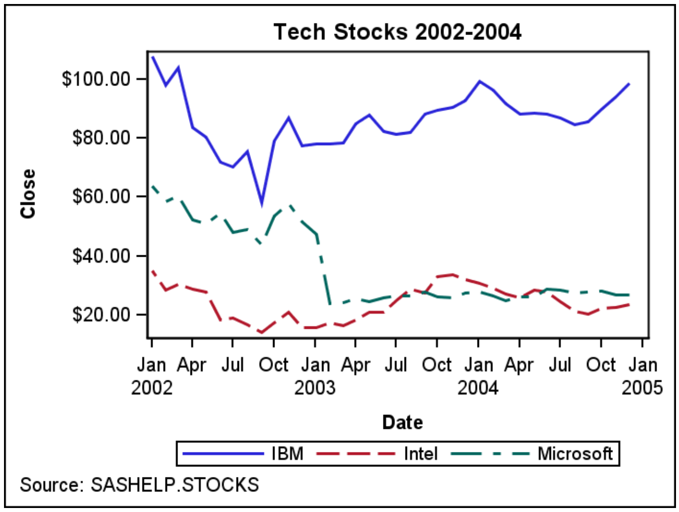


Other statements such as DENSITYPLOT, REGRESSIONPLOT, LOESSPLOT, PBSPLINEPLOT, MODELBAND, REFERENCELINE, and DROPLINE are "specialized" in the sense that their default line appearance is governed by other style elements such as GraphFit, GraphConfidence, GraphPrediction, GraphReference, or some other specialized style element. When these statements are used in conjunction with the "non-specialized" plot statements, there will automatically be differences in appearance.

Appearance of Grouped Data

The GROUP= *column* option is used to plot data when a classification or grouping variable is available. By default, this option automatically uses the style elements GraphData1 to GraphDataN for the presentation of each unique group value.

```
proc template;
  define statgraph group;
    begingraph;
      entrytitle "Tech Stocks 2002-2004";
      entryfootnote halign=left "Source: SASHELP.STOCKS";
      layout overlay;
        seriesplot x=date y=close / group=stock name="series"
          lineattrs=(thickness=2);
        discretelegend "series";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.stocks template=group;
  where date between "1jan02"d and "31dec04"d;
run;
```



In general, you cannot specify the line or marker properties for specific group values directly from GTL, as you can for non-grouped data. Consider the following specification for a line attribute:

```
lineattrs=(thickness=2)
```

When this line option is used, the line thickness is set as a style override. However, this override applies equally to all group values. The same is true for overrides of color or line pattern. This means that you can set one or more fixed appearance attributes from GTL for all group values, but you cannot directly assign unique visual properties to an individual group value in GTL.

Using Custom Styles to Control the Appearance of Grouped Data

Each style potentially can change the style attributes for GraphData1-GraphDataN. If you have certain preferences for grouped data items, you can create a modified style that will display your preferences. The following code creates a new style named STOCKS that is based on the supplied style STYLES.LISTING. This modification changes the properties for the GraphData1-GraphData3 style elements. All other style elements are inherited from LISTING.

```
proc template;
  define style stocks;
    parent=styles.listing;
    style GraphData1 /
      ContrastColor=blue
      Color=lightBlue
      MarkerSymbol="CircleFilled"
      Linestyle=1;
    style GraphData2 /
      ContrastColor=brown
      Color=lightBrown
      MarkerSymbol="TriangleFilled"
      Linestyle=1;
    style GraphData3 /
      ContrastColor=orange
```

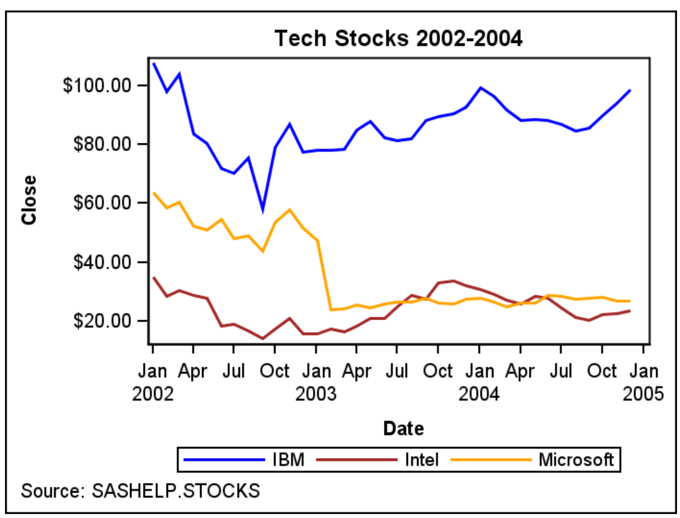
```
        Color=lightOrange
        MarkerSymbol="SquareFilled"
        Linestyle=1;
    end;
  run;
```

In this style definition, the LINESTYLE is set to 1 (solid) for the first three data values. Style syntax requires that line styles be set with their numeric value, not their keyword counterparts in GTL such as SOLID, DASH, or DOT. See Appendix A3, "Values for Marker Symbols and Line Patterns," on page 355 for the complete set of line styles.

CONTRASTCOLOR is the attribute applied to grouped lines and markers. COLOR is the attribute applied to grouped filled areas, such as grouped bar charts or grouped ellipses. MARKERSYMBOL defines the same values that can be specified with the MARKERATTRS=(SYMBOL=*keyword*) option in GTL. See Appendix A3, "Values for Marker Symbols and Line Patterns," on page 355 for the complete set of marker names.

After the STOCKS style is defined, it must be requested on the ODS destination statement. No modification of the compiled template is necessary:

```
ods listing style=stocks;

proc sgrender data=sashelp.stocks
              template=group;
 where date between
      "1jan02"d and "31dec04"d;
run;
```

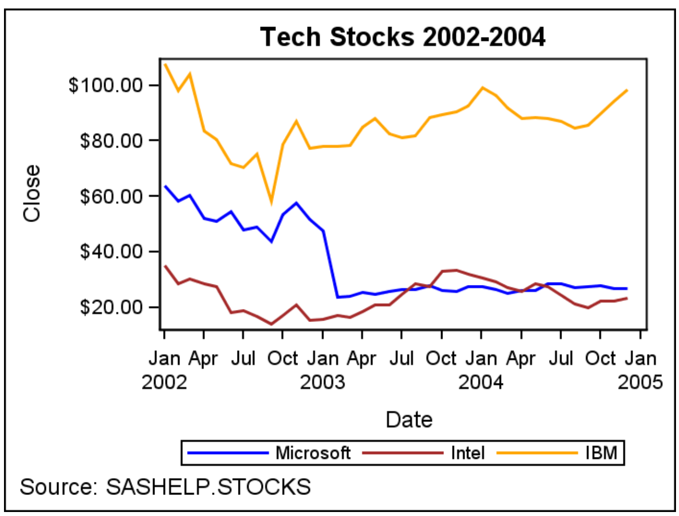


One issue you should be aware of is that the STOCKS style only customized the appearance of the first three group values. If there were more group values, other unaltered style elements will be used, starting with GraphData4. Most styles define (or inherit) GraphData1 to GraphData12 styles elements. If you need more elements, you can add as many as you desire, starting with one more than the highest existing element (for example, GraphData13) and numbering them sequentially thereafter.

Making the Appearance of Grouped Data Independent of Data Order

When unique group values are gathered, they are internally recorded in the order that they appear in the data. They are not subsequently sorted. This means that if an input data source is modified, sorted, or filtered, the order of the group values and their associations with GraphData1-GraphDataN might change.

```
proc sort data=sashelp.stocks out=stocks;
   by date descending stock;
run;

ods listing style=stocks;

proc sgrender data= stocks template=group;
 where date between
       "1jan02"d and "31dec04"d;
run;
```



In many cases, this might not be a problem because you really don't care which line pattern, marker symbols, or colors are associated with particular group values, but in some cases you might care. For example, if you create many plots grouped by GENDER, you might want a consistent set of visual properties for Females and Males across plots, regardless of the input data order.

In our example, you might want IBM to always have a blue color associated with it. To enforce this kind of association, you need to modify the input data by adding another numeric column that maps all group values to a positive integer that corresponds to one of the GraphData1-GraphDataN style elements. The name of the new variable can be anything.

```
data indexed;
  set sashelp.stocks;
  select (stock);
    when ("IBM")       ID=1;
    when ("Microsoft") ID=2;
```

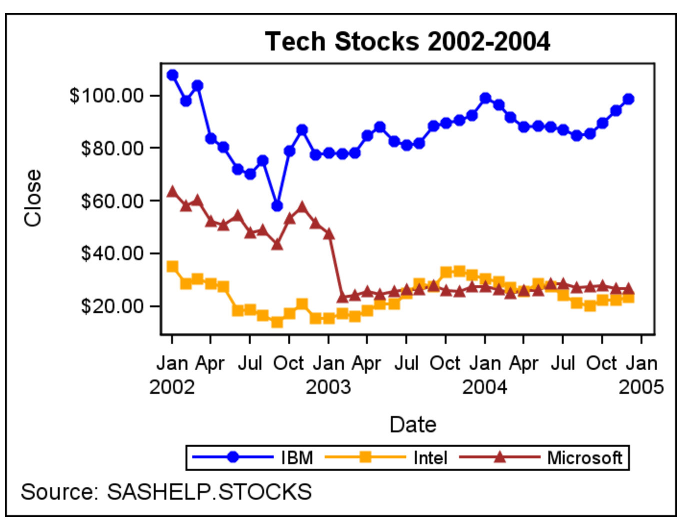
```
      when ("Intel")      ID=3;
      otherwise;
    end;
  run;
```

Now the template needs to be modified by adding an INDEX=ID option. The DISPLAY=(MARKERS) option was added to make sure that desired markers are also used.

```
proc template;
  define statgraph groupindex;
    begingraph;
      entrytitle "Tech Stocks 2002-2004";
      entryfootnote halign=left "Source: SASHELP.STOCKS";
      layout overlay;
        seriesplot x=date y=close / group=stock index=id
           name="series" lineattrs=(thickness=2) display=(markers) ;
        discretelegend "series";
      endlayout;
    endgraph;
  end;
run;
```

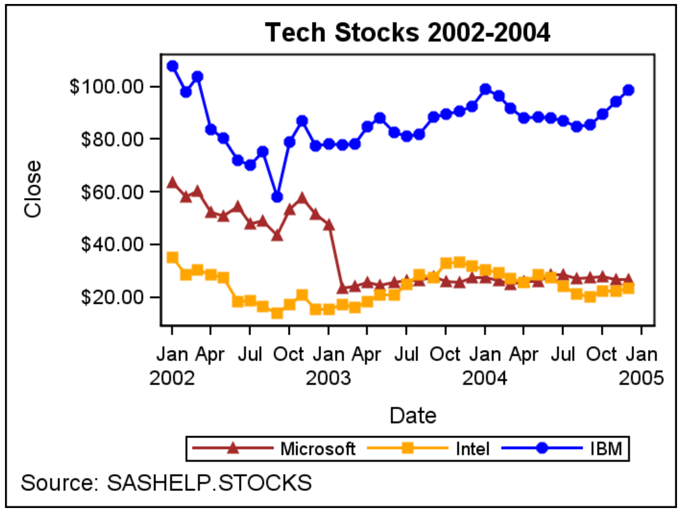
Now we can repeat our test to make sure that IBM is blue regardless of data order.

```
ods listing style=stocks;

proc sgrender data=indexed
              template=groupindex;
 where date between
      "1jan02"d and "31dec04"d;
run;
```




```
proc sort data=indexed out=stocks;
  by date descending stock;
run;

ods listing style=stocks;

proc sgrender data= stocks
              template=groupindex;
```

```
 where date between
      "1jan02"d and "31dec04"d;
run;
```



Recommendations

The issue of when to use hard-coded values versus style references for overriding appearance features is complex and basically boils down to what you are trying to achieve with GTL. Here are some recommendations that are based on common use cases:

- You are creating a graph for a specific purpose and probably will not use the code again.

  Recommendation: Develop your template code with one style in mind and use hard-coded overrides to make desired changes. One possibility is to use the JOURNAL style as a starting point. It has a gray-scale color scheme. If you want to introduce colors for certain parts the graph, there won't be much conflict with blacks and grays coming from the style. You really don't care what the graph looks like with another style.

- You are creating a reusable graph template (without hard-coded variable names) that can be used with different sets of data in different circumstances.

  Recommendation: If style overrides are needed, use style-reference overrides, not hard-coded overrides. This will allow your graph's appearance to change appropriately when you (or someone else) uses a different style.

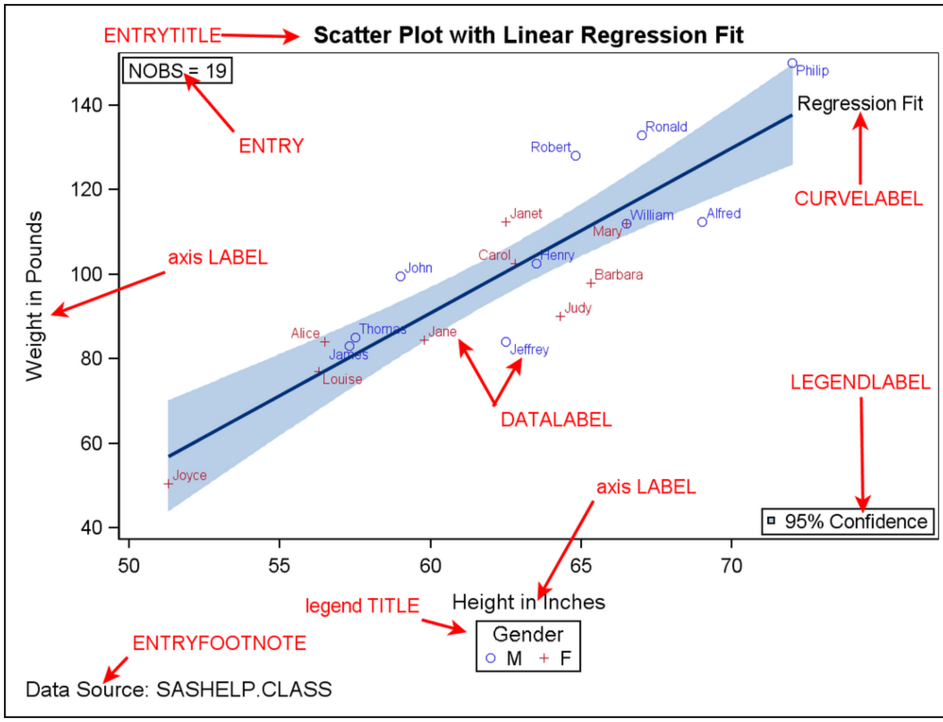
- You want all of your templates to produce output with the same look-and-feel, possibly a corporate theme.

  Recommendation: Spend time developing a new style that produces the desired "look-and-feel" rather than making a lot of similar appearance changes every time you create a new graph template to enforce consistency. Be sure to coordinate the colors and fonts for the graphical style elements with tabular style elements. See Chapter 17, “Managing the Graph Appearance with Styles,” on page 299 for more information.

Chapter 7
Adding and Changing Text in a Graph

Text Strings in Graphs

Using the GTL, you can add and control text that appears in your graph. The annotation in the following diagram indicates some of the options and statements that are used to set the text in a typical graph.



The following options, available on plot and legend statements, manage most of the text that you can add to a graph:

Task	Statement	Option
label data points	plot statements that display markers	DATALABEL=*column*
label a curve or a reference line	plot statements that display lines	CURVELABEL="*string*" | *column*
describe a plot in a legend	most plot statements	LEGENDLABEL="*string*"
add title to a legend	legend statements	TITLE="*string*"
label an axis	axis statement or layout axis option	LABEL="*string*"

The GTL also provides the following text statements that can be used to add custom information about the graph analysis or the graph display. This text is independent of the text that is managed by the options on plot and legend statements:

ENTRYTITLE "*string*"	Defines title text for the entire graph.
ENTRYFOOTNOTE "*string*"	Defines footnote text for the entire graph.
ENTRY "*string*"	Defines text that is displayed in the graphical area.

This chapter focuses primarily on how to set text properties for any text. Additional information on text-related features for axes, legends, insets, and multi-cell layouts is available in other chapters:

- For managing the text in axes, see Chapter 5, “Managing Axes in an OVERLAY Layout,” on page 53.
- For managing the text in legends, see Chapter 8, “Adding Legends to a Graph,” on page 117.
- For managing the text in insets, see Chapter 16, “Adding Insets to a Graph,” on page 271.
- For managing the text in multi-cell layouts, see Chapter 10, “Using an Advanced Multi-cell Layout,” on page 155 and Chapter 11, “Using Classification Panels,” on page 185.

Text Properties and Syntax Conventions

All options or statements that define text strings have supporting text options that enable you to set the color and font properties of the text. The following table shows some of the supporting text options that are available:

Statement Type	Option	Supporting Text Option
plot statements	DATALABEL=	DATALABELATTRS=
	CURVELABEL=	CURVELABELATTRS=
legend statements	TITLE=	TITLEATTRS=
layout or axis statements	LABEL=	LABELATTRS=
text statements		TEXTATTRS=

The supporting text options all have similar syntax:

supporting-text-option = *style-element* | *style-element* (*text-options*) | (*text-options*)

All *supporting-text-options* use a style element to determine their default characteristics. Thus, when a different ODS style is applied to a graph, you might see different fonts, font sizes, font weights, and font styles used for various pieces of text in the graph. See “Attributes as Collections of Related Options” on page 88 for a full discussion of how style elements and override options work.

Any text that you add to the graph can have the following properties for the *text-options*:

Text Option	Value	Examples
COLOR=	*color* | *style-reference*	(color=black) (color=GraphLabelText:color)

Text Option	Value	Examples
FAMILY=	" *font-name* " | *style-reference*	(family="Arial Narrow") (family=GraphLabelText:FontFamily)
SIZE=	*dimension* | *style-reference*	(size=10pt) (size=GraphLabelText:FontSize)
WEIGHT=	NORMAL | BOLD | *style-reference*	(weight=bold) (weight=GraphLabelText:FontWeight)
STYLE=	NORMAL | ITALIC | *style-reference*	(style=italic) (style=GraphLabelText:FontStyle)

Several style elements affect text in different parts of a graph. Each style element defines attributes for all of its available text options. The following table shows some of the style elements that are available for setting text attributes:

Style Element	Default Use
GraphTitleText	Used for all titles of the graph. Typically uses the largest font size among fonts in the graph.
GraphFootnoteText	Used for all footnotes. Typically uses a smaller font size than the titles. Sometime footnotes are italicized.
GraphLabelText	Used for axis labels and legend titles. Generally uses a smaller font size than titles.
GraphValueText	Used for axis tick values and legend entries. Generally uses a smaller font size than labels.
GraphDataText	Used for text where minimum size is necessary (such as point labels).
GraphUnicodeText	Used for adding special glyphs (for example α , ±, €) to text in the graph.
GraphAnnoText	Default font for text that is added as annotation (using the ODS Graphics Editor).

For example, to specify that axis labels should have the same text properties as axis tick values, you could specify the following:

```
layout overlay / xaxisopts=( labelattrs=GraphValueText )
                 yaxisopts=( labelattrs=GraphValueText );
```

Style elements can also be used to modify the display of grouped data. For example, by default, the text color of data labels for grouped markers in a scatter plot changes to match the marker color for each group value. To specify that grouped data labels should use the same color as non-grouped data labels, you could specify the following:

```
scatterplot x=height y=weight / group=age datalabel=name
  datalabelattrs=( color=GraphDataText:Color ) ;
```

To ensure that a footnote is displayed in bold italics, you could specify the following:

```
entryfootnote "Study conducted in 2007" /
  textattrs=( weight=bold style=italic ) ;
```

Because the other font properties are not overridden in this example, they are obtained from the GraphFootnoteText style element.

Text Statement Basics

The ENTRYTITLE, ENTRYFOOTNOTE, and ENTRY statements add text to predefined areas of the graph. The text that they add cannot be specified by the options that are available in plot, axis, legend, or layout statements.

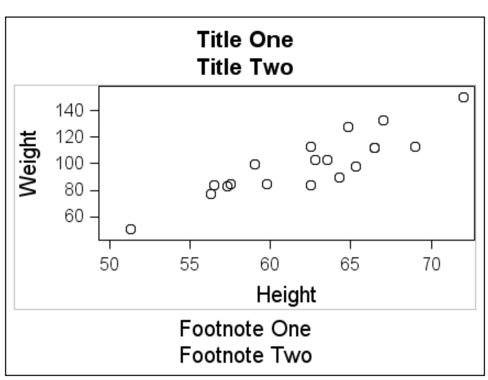
Using Titles and Footnotes

To add titles or footnotes to a graph, use one or more ENTRYTITLE or ENTRYFOOTNOTE statements. These statements must appear inside the BEGINGRAPH block, but outside any layout blocks. The following code shows the typical placement of these statements:

```
begingraph;
  entrytitle "Title One";
  entrytitle "Title Two";
  layout overlay;
    scatterplot x=height y=weight;
  endlayout;
  entryfootnote "Footnote One";
  entryfootnote "Footnote Two";
endgraph;
```

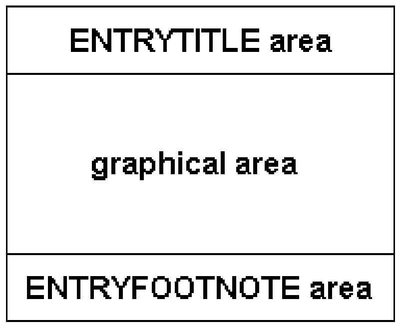
However, the following statement placement yields the same result:

```
begingraph;
  entryfootnote "Footnote One";
  entrytitle "Title One";
  layout overlay;
    scatterplot x=height y=weight;
  endlayout;
  entryfootnote "Footnote Two";
  entrytitle "Title Two";
endgraph;
```



Note: a light gray border was added to the graph area to indicate the boundaries between the separate areas.

Unlike SAS TITLE and FOOTNOTE statements, the GTL statements are not numbered. If you include multiple ENTRYTITLE or ENTRYFOOTNOTE statements, the titles or footnotes will be stacked in the specified order —all ENTRYTITLE statements are gathered and placed in the ENTRYTITLE area at the top of the graph, and all ENTRYFOOTNOTE statements are gathered and placed in the ENTRYFOOTNOTE area at the bottom of the graph.

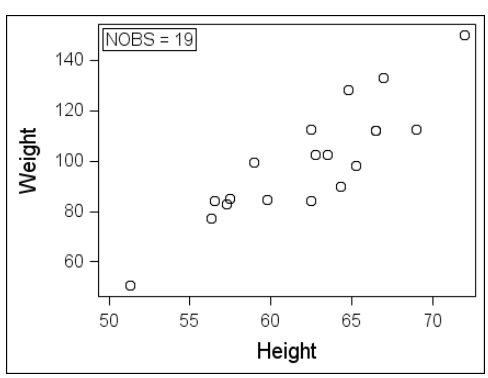


You can add as many titles and footnotes as you want. However, the space that is needed to accommodate the titles and footnotes always decreases the height of the graphical area. For graphs with extensive titles or footnotes, you should consider enlarging the graph size. For a discussion on sizing graphs, see "Controlling Graph Size" on page 326.

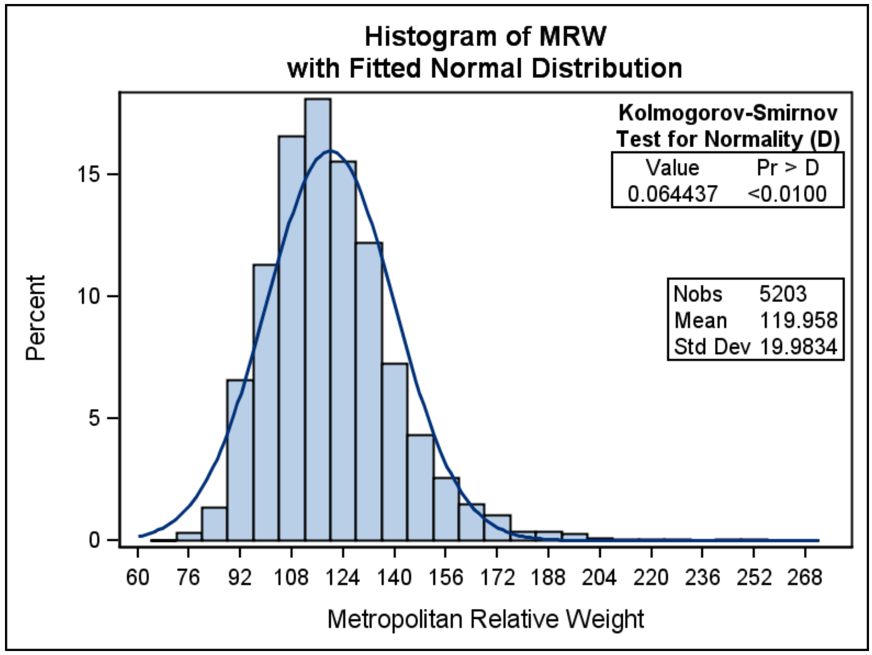
Using Text Entries in the Graphical Area

An ENTRY statement defines text within the graphical area. Here is a simple example that places text in the upper left corner of the plot wall area:

```
layout overlay;
  scatterplot x=height y=weight;
  entry halign=left "NOBS = 19" /
        valign=top border=true;
endlayout;
```



You can use multiple ENTRY statements in conjunction with GRIDDED layouts to create tables of text and complex insets.



This example is discussed in detail in Chapter 16, "Adding Insets to a Graph," on page 271

Managing the String on Text Statements

Text Statement Syntax

Options on the ENTRYTITLE, ENTRYFOOTNOTE, and ENTRY text statements enable you to create simple or complex text constructs. The following syntax shows the general form of these statements:

TEXT-STATEMENT *text-item* <…<*text-item*>> / <options>;

Any *text-item* is some combination of the following:

<*prefix-option* …<*prefix-option*>> "*string*" | *dynamic* | *character-expression* | {*text-command*}

What this means is that the final text that is to be created can be specified in a series of separate items, each with individual prefix options. Statement options can also affect the final text. These possibilities are explained by the examples in the following sections.

Using Rich Text

"Rich text" describes text in which each character can have different text properties. The following example creates rich text by separating the text into pieces and using prefix options to set different text properties for each piece. Properties that are set this way stay in effect for subsequent text items, unless changed by another TEXTATTRS= prefix option.

```
entrytitle textattrs=(size=12pt color=red) "Hello "
           textattrs=(size=10pt color=blue style=italic) "World";
```

Hello ***World***

For each horizontal alignment, the overall text for these statements is formed by the concatenation of the text items. Notice that there is no concatenation operator and that any spacing (such as word breaks) must be provided as needed within the strings ("Hello " "World"). The space that separates the text-item specifications is never included in the final text string.

Horizontally Aligning Text Items

Text items can have different horizontal alignments: LEFT, CENTER, or RIGHT. The default alignment is CENTER. Text items with the same alignment are gathered and concatenated.

```
entryfootnote halign=left textattrs=(weight=bold) "XYZ Corp."
              halign=right textattrs=(weight=normal) "30JUN08";
```

XYZ Corp. 30JUN08

Generating Text Items with Dynamics, Macro Variables, and Expressions

Text items are not limited to string literals. Text items can also be defined as dynamics, macro variables, or expressions. In the following example, SYSDATE is declared with an MVAR template statement. As a result, this automatic macro variable is resolved to today's date at runtime.

```
entryfootnote halign=left textattrs=(weight=bold) "XYZ Corp."
              halign=right textattrs=(weight=normal) SYSDATE ;
```

XYZ Corp. 30JUN08

This next example shows how the GTL EVAL function causes an expression to be evaluated at runtime. In this case, the PUT function (same as the PUT function in the DATA step) is used to convert a SAS date value into a string:

```
entryfootnote "Summary for "  eval(put(today(),mmddyyd.)) ;
```

Summary for 06-30-08

For more information on dynamics, macro variables, and EVAL expressions in GTLsee Chapter 14, "Using Dynamics and Macro Variables to Make Flexible Templates," on page 251 and Chapter 15, "Using Conditional Logic and Expressions," on page 259.

Adding Subscripts, Superscripts, and Unicode Rendering

You can build strings with subscripts or superscripts using the {SUB "*string*" } or {SUP "*string*" } text commands. You can also use dynamics or macro variables for the *string* portion of the text command.

```
entryfootnote "R" {sup "2"} "=.457";
entryfootnote "for the H" {sub "2"} "O Regression" ;
```

R^2=.457
for the H_2O Regression

Another way to form text is to use the {UNICODE "*hex-value*"x } text command. For fonts that support Unicode code points, you can use the following syntax to render the glyph (character) corresponding to any Unicode value:

```
entryfootnote  {unicode "03B1"x} "=.05" ;
```

In the code, the **"03B1"x** is the hexadecimal code point value for the lowercase Greek letter alpha. Because Greek letters and some other statistical symbols are so common in statistical graphics, keyword short cuts to produce them have been added to GTL syntax. So another way of indicating **"03B1"x** is

```
entryfootnote  {unicode alpha} "=.05" ;
```

α=.05

For a complete list of keywords that can be used with the **{unicode keyword}** notation, see Appendix A1, “SAS Keywords for Unicode Glyphs,” on page 341.

Additionally, any Unicode glyph for currency, punctuation, arrows, fractions and mathematical operators, symbols, and dingbats can be used. Fonts such as Arial (comparable to SAS-supplied Albany AMT) have many, but not all, Unicode code points available, and sometimes a more complete Unicode font such as Arial Unicode MS (or SAS-supplied Monotype Sans WT J) needs to be specified. ODS styles have a style element named GraphUnicodeText that can be safely used for rendering any unicode characters. The following example uses the GraphUnicodeText style element for rendering a bar over the X:

```
entry  "X'{unicode bar}"=6.78" / textattrs=GraphUnicodeText;
```

$\bar{X}$=6.78

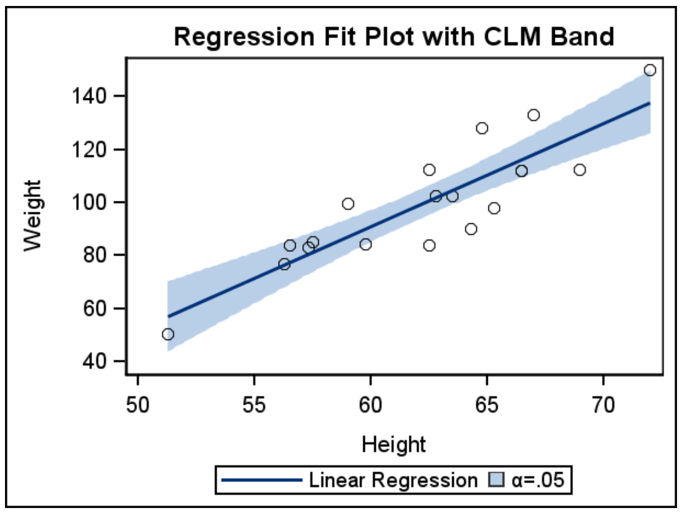
Using Unicode Values in Labels

The {UNICODE}, {SUB}, and {SUP} text commands apply only to the ENTRY, ENTRYTITLE, and ENTRYFOOTNOTE statements. However, strings that are assigned to axis labels, curve labels, legend labels, and so on, can present Unicode characters using what is called "in-line formatting." To use this special formatting, you embed within the string an ODS escape sequence followed by a text command. Specifically, whenever you include (*ESC*) in a quoted string, it signals that the next token represents a text command. Currently, only the {UNICODE} text command is recognized, not {SUB} or {SUP}.

In the following example, the alpha value for the upper and lower confidence limits is displayed using the Greek letter alpha:

```
proc template;
  define statgraph fit;
    begingraph;
      entrytitle "Regression Fit Plot with CLM Band";
      layout overlay;
        modelband "clm" / display=(fill)  name="band"
          legendlabel="(*ESC*){unicode alpha}=.05" ;
        scatterplot x=height y=weight / primary=true  ;
```

```
            regressionplot x=height y=weight / alpha=.05 clm="clm"
              legendlabel="Linear Regression" name="fit";
            discretelegend "fit" "band";
          endlayout;
        endgraph;
      end;
    run;

    proc sgrender data=sashelp.class template=fit;
    run;
```



Using Options on Text Statements

Options Available on All Text Statements

The ENTRYTITLE, ENTRYFOOTNOTE, and ENTRY text statements provide options that apply to all of the *text-items* that form the text string (unlike the prefix options, which can be applied to pieces of the text).

TEXT-STATEMENT *text-item* <…<*text-item*>> / <options>;

The following options are available on all of the text statements:

BACKGROUNDCOLOR= *style-reference* | *color*
Specifies the color of the text background.

BORDER= *boolean*
Specifies whether a border line is displayed around the text.

BORDERATTRS= *style-element* | *style-element* (*line-options*) | (*line-options*)
Specifies the properties of the border line.

OPAQUE= *boolean*
Specifies whether the entry background is opaque.

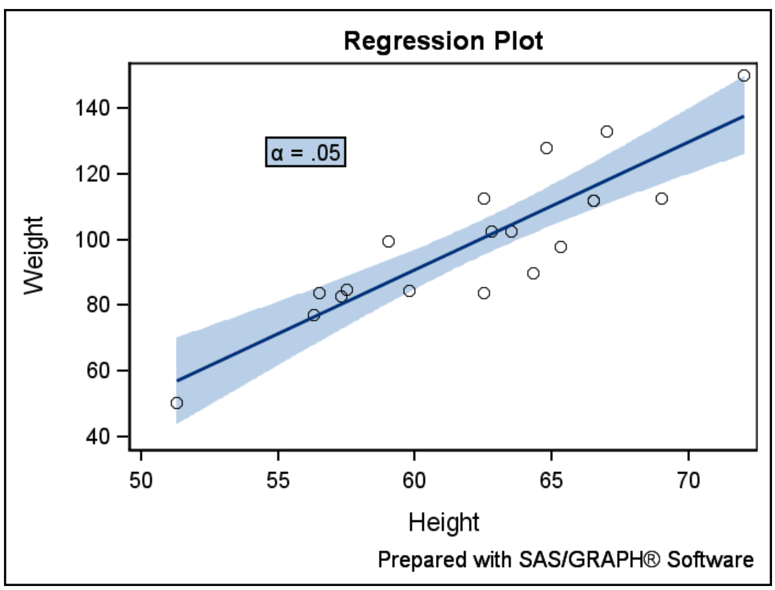
TEXTATTRS=*style-element* | *style-element* (*text-options*) | (*text-options*)
Specifies the font attributes of all text. If a TEXTATTRS= prefix option is also used, it takes precedence over this statement option.

Setting Text Background, Borders, and Padding

By default, the background of all text is transparent. To specify a background color, you must specify OPAQUE=TRUE to turn off transparency, which then enables you to specify a background color. In the following example, the fill color of the band is specified for the background of the entry text. A border is also added.

Note: Data points that are behind the entry text are obscured when OPAQUE=TRUE.

```
begingraph;
  entrytitle "Regression Plot";
  entryfootnote halign=right
   "Prepared with SAS/GRAPH" {unicode "00AE"x} " Software" /
    textattrs=(size=9pt);
  layout overlay;
    modelband "clm";
    scatterplot x=height y=weight;
    regressionplot x=height y=weight / clm="clm" alpha=.05;
    entry {unicode alpha} " = .05" / autoalign=auto border=true
      opaque=true backgroundcolor=GraphConfidence:color ;
  endlayout;
endgraph;
```



Notice that extra space appears between the entry border and the text. This space is called padding and can be set with the PAD= option. The default padding is

```
ENTRY "string" / PAD=(LEFT=3px RIGHT=3px TOP=0 BOTTOM=0) border=true;
```

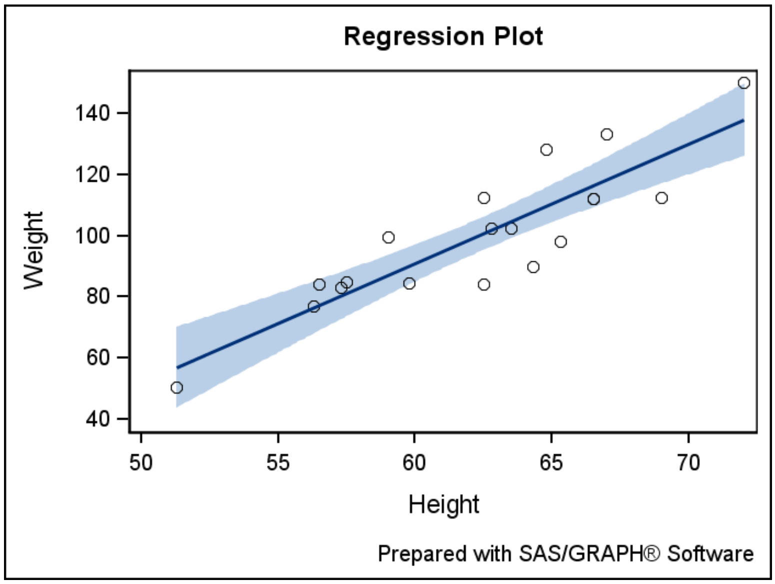
You can set the padding individually for the LEFT, RIGHT, TOP, and BOTTOM directions, or you can set the same padding in all directions as follows:

```
ENTRY "string" / PAD=5px border=true;
```

Padding is especially useful when you want to add extra space between titles, or add space between the last title (or first footnote) and the plot area in the graph:

```
begingraph;
  entrytitle "Regression Plot" / pad=(bottom=10px) ;
  entryfootnote halign=right
```

```
      "Prepared with SAS/GRAPH" {unicode "00AE"x} " Software" /
       textattrs=(size=9pt) pad=(top=10px) ;
     layout overlay;
       modelband "clm";
       scatterplot x=height y=weight;
       regressionplot x=height y=weight / clm="clm" alpha=.05;
     endlayout;
   endgraph;
```



Managing Long Text in Titles and Footnotes

When you change the size of a graph, the size of all fonts in the graph is scaled up or down by default. However, when the graph size is reduced, even font scaling has limits on what it can do with long text strings that are specified on ENTRYTITLE or ENTRYFOOTNOTE statements. The following statement options are available to deal with this situation:

TEXTFITPOLICY= WRAP | SHORT | TRUNCATE

SHORTTEXT= (*text-items*)

By default, TEXTFITPOLICY=WRAP, and no default is defined for the SHORTTEXT= option.

The text fitting policies take effect when the length of the text and/or its font properties cause the text line to exceed the space available for it. The font properties include the font family, font size, and font weight (BOLD or NORMAL). Thus, adjusting the length of the text and/or changing its font properties are adjustments you can make to fit text in the available space. You can also use the TEXTFITPOLICY= and/or SHORTTEXT= options.

The following long title uses the default fit policy, which is to wrap text that does not fit on a single line:

```
entrytitle "This is a lot of text to display on one line";
```

This is a lot of text to display on
one line

Notice that the current horizontal alignment (CENTER in this case) is used when text wraps. Text is wrapped only at word boundaries (a space). This next example sets the fit policy to TRUNCATE, and the ellipsis in the output text indicates where the truncation occurs.

This is a lot of text to display o...

Rather than truncating text, you can specify alternative "short" text to substitute whenever the primary text will not fit without wrapping in the available space. The short text is substituted whenever the primary text won't fit without wrapping.

```
entrytitle "This is a lot of text to display on one line" /
  textfitpolicy=short shorttext=("Short alternative text");
```

Short alternative text

ENTRY Statements: Additional Control

Features Available for ENTRY Text

ENTRY statements are more flexible than ENTRYTITLE or ENTRYFOOTNOTE statements and support additional features for automatically positioning text, aligning text vertically, and rotating text:

AUTOALIGN= NONE | AUTO | (*location-list*)
Specifies whether the entry is automatically aligned within its parent when nested within an overlay-type layout.

ROTATE= 0 | 90 | 180 | 270
Specifies the angle of text rotation.

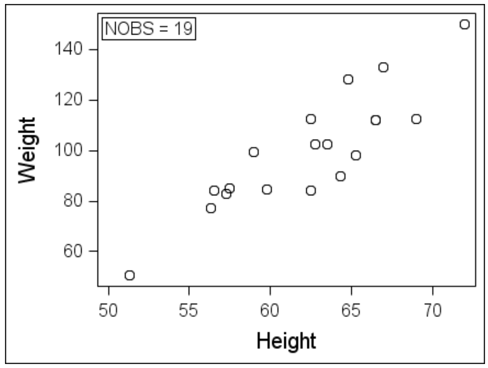
VALIGN= CENTER | TOP | BOTTOM
Specifies the vertical alignment of the text.

Positioning ENTRY Text

By default, any ENTRY statement that is defined within a 2D overlay-type layout and does not specify a location is placed in the center of the graph wall (HALIGN=CENTER VALIGN=CENTER).

If you know where you want to place the text, one way to position it is to use the HALIGN= and VALIGN= options, as shown in the following example:

```
layout overlay;
  scatterplot x=height y=weight;
  entry halign=left "NOBS = 19" /
        valign=top border=true;
endlayout;
```

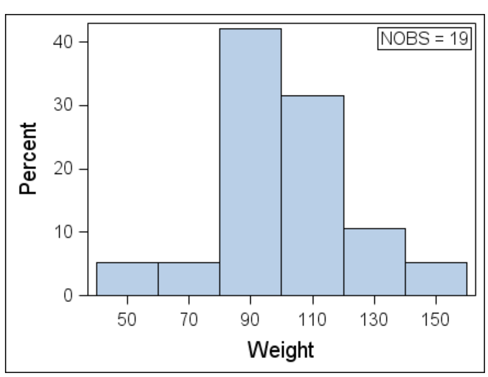


Whenever you add text within the graph wall, you have to consider the possibility that the text might appear on top of or behind data markers and plot lines. For this reason, you should consider using the AUTOALIGN= option rather than the HALIGN= and VALIGN= options for positioning the text.

The AUTOALIGN= option enables you to set a priority list that restricts the entry location to certain locations. The priority list can include any of the keywords TOPLEFT, TOP, TOPRIGHT, LEFT, CENTER, RIGHT, BOTTOMLEFT, BOTTOM, and BOTTOMRIGHT.

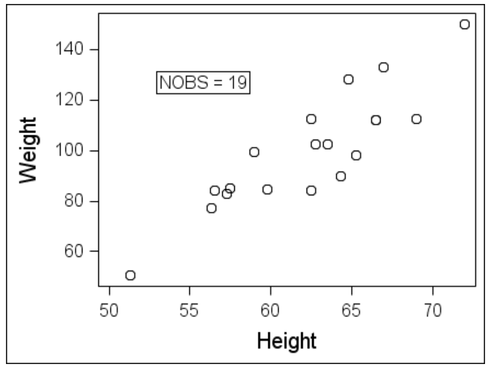
In the following histogram, we know that the best location for an entry is either TOPLEFT or TOPRIGHT, depending on the skewness of the data. With the following coding, if the data were skewed to the right so the entry text overlaps with the histogram, the text would automatically appear at TOPLEFT.

```
layout overlay;
  histogram weight;
  entry  "NOBS = 19" /
    autoalign=(topright topleft)
    border=true;
endlayout;
```



When the parent layout contains only scatter plots, the ENTRY statement can use the AUTOALIGN=AUTO setting to automatically position the text where it is the farthest away from any scatter points. In all cases, even one like the following example where many positions are available that might minimize data collision, the AUTO specification selects the position for you and you have no further control over the text position.

```
layout overlay;
  scatterplot x=height y=weight;
  entry halign=left "NOBS = 19" /
        autoalign=auto border=true;
endlayout;
```

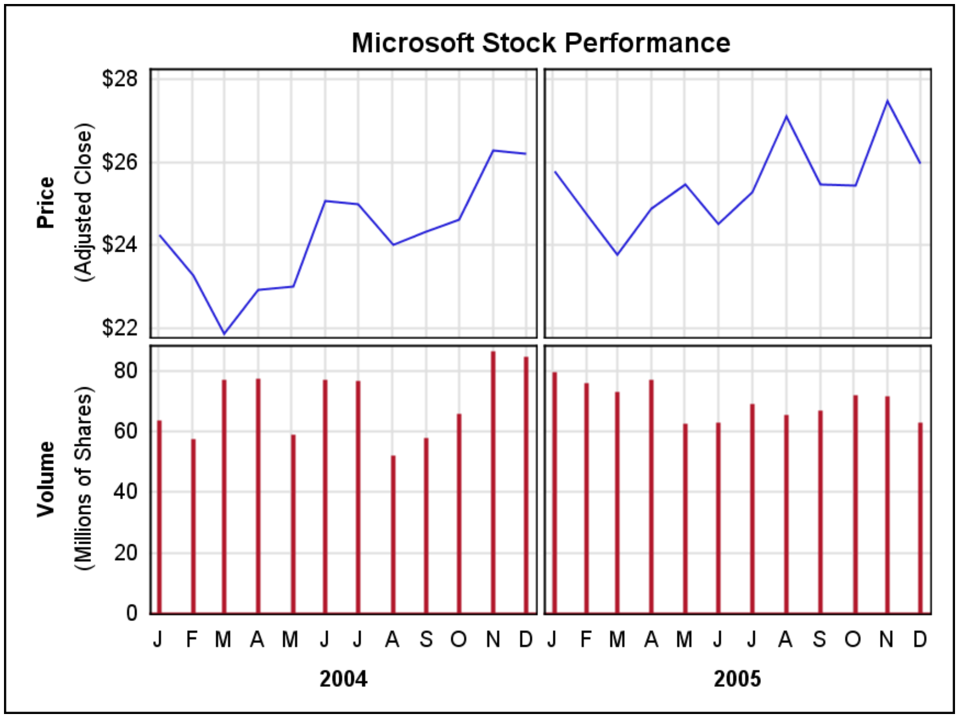


Rotating ENTRY Text

ENTRY statements can appear in most layout types. For example, ENTRY statements can be used to define the text that appears in a CELLHEADER block in a LATTICE layout. You can also use ENTRY statements in SIDEBAR , ROWHEADER, and COLUMNHEADER blocks.

In the following example, the ROWHEADERS block shows how to define rotated row headers for a lattice layout. The complete code for this example is shown in “Defining a Lattice with Additional Features” on page 169.

```
 rowheaders;
   layout gridded / columns=2;
     entry "Volume" / textattrs=GraphLabelText rotate=90 ;
     entry "(Millions of Shares)" / textattrs=GraphValueText rotate=90 ;
   endlayout;
   layout gridded / columns=2;
     entry "Price" / textattrs=GraphLabelText rotate=90 ;
     entry "(Adjusted Close)" / textattrs=GraphValueText rotate=90 ;
   endlayout;
endrowheaders;
```


Microsoft Stock Performance
Price
(Adjusted Close)
$28
$26
$24
$22
Volume
(Millions of Shares)
80
60
40
20
0
J F M A M J J A S O N D
J F M A M J J A S O N D
2004
2005

Chapter 8
Adding Legends to a Graph

Introduction to Legend Management

Some of the Uses for a Legend

A graphical legend provides a key to the marker symbols, lines, and other data elements that are displayed in a graph. Here are some of the situations where legends are useful:

- when a plot contains grouped markers (scatter plots, for example)
- when a plot contains lines that differ by color, marker symbol, or line pattern (series plots or step plots, for example)
- when a plot contains one or more lines or bands that require identification or explanation
- when series plots with different data are overlaid in the graph, or fit lines are displayed with confidence bands, or density plots with different distributions are generated
- when markers vary in color to show the values of a response variable
- when contour or surface plots use gradient fill colors to show the values of a response variable.

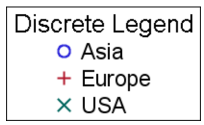
GTL does not automatically generate legends for the above situations. However, the mechanism for creating legends is simple and flexible.

Types of Legends in GTL

GTL supports two legend statements:

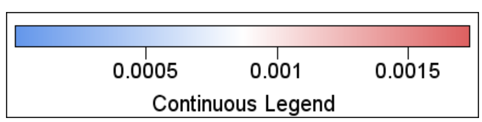
DISCRETELEGEND
: legend that contains one or more legend entries. Each entry consists of a graphical item (marker, line, ...) and corresponding text that explains the item. A discrete legend would be used for the first two situations listed in "Some of the Uses for a Legend" on page 117.

  For details, see "General Legend Features" on page 121 and "Features of Discrete Legends" on page 128.



CONTINUOUSLEGEND
: legend that maps a color gradient to response values. A continuous legend would be used for the last two situations listed in "Some of the Uses for a Legend" on page 117.

  For details, see "General Legend Features" on page 121 and "Features of Continuous Legends" on page 137.



General Syntax for Using Legends

Regardless of the situation, the basic strategy for creating legends is to "link" one or more plot statements to a legend statement by assigning a unique, case-sensitive name to the plot statement and then referencing that name on the legend statement:

```
plot-statement . . . / name="id-string1" ;

plot-statement . . . / name="id-string2" ;

legend-statement  "id-string1"  "id-string2"  < / options > ;
```

One way of thinking about this syntax is that you can identify any plot with a NAME= option, and you can then selectively include plot names on a legend statement. This enables the legend to query the identified plots so that it can get the information it needs to build the legend entries.

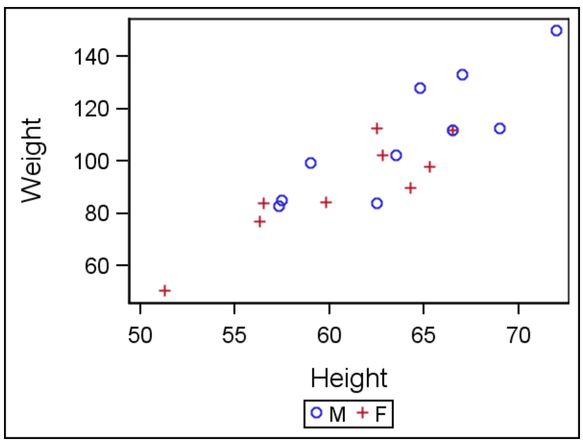
Note: When the legend statement includes the name of a plot, it does not always mean that the legend will include an entry for that plot. For example, a block plot with FILLTYPE=ALTERNATE will not show up in a legend.

Example Legend Coding for Common Situations

Show group values in a legend

The appearance of the markers is automatically determined by the current style. The order of the legend entries is controlled by the data order.

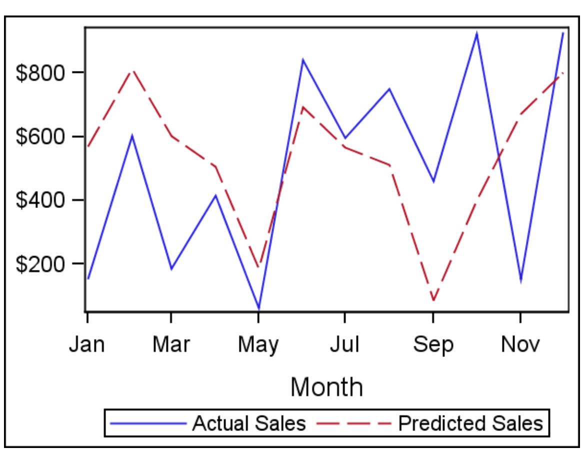
```
layout overlay;
  scatterplot x=height y=weight / group=sex name="scatter";
  discretelegend "scatter";
endlayout;
```



Identify overlaid plots in a legend

This example illustrates that more than one plot can contribute to a legend. The order of the names in the DISCRETELEGEND statement controls the order of the legend entries. For more information about the CYCLEATTRS= option, see "Ordering the Legend Entries for Non-grouped Plots" on page 130.

```
layout overlay / cycleattrs=true;
  seriesplot x=month y=actual    / name="sp1";
  seriesplot x=month y=predicted / name="sp2";
  discretelegend "sp1" "sp2";
endlayout;
```

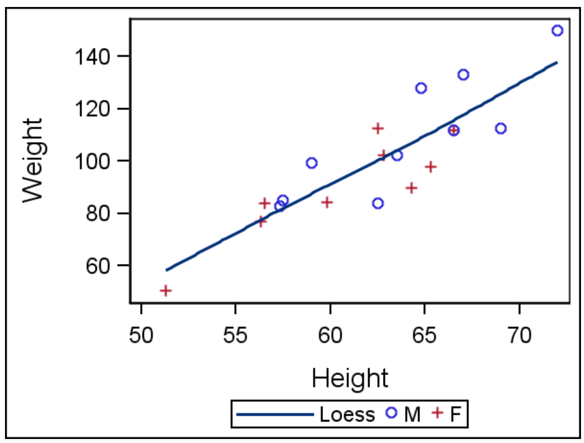


In this case, the default legend entry text was determined by the label for the Y= variable of each plot. You could set the legend entry text explicitly by specifying LEGENDLABEL="*string*" on each plot statement.

Show group values and identify plots in a legend

A legend can show group values for multiple groups and identify one or more plots.

```
layout overlay;
  scatterplot x=height y=weight / group=sex name="scatter";
  loessplot x=height y=weight / name="Loess";
  discretelegend "Loess" "scatter";
endlayout;
```

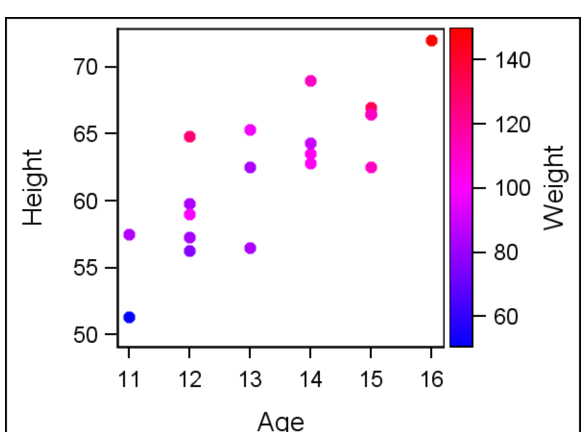


If a plot variable does not have a variable label, the case-sensitive plot name is used for the legend label. In this case, because the Y= variable of the LOESSPLOT statement does not have a variable label, the plot name "Loess" is used. You could also set the legend entry text explicitly by setting LEGENDLABEL="*string*" in the LOESSPLOT statement.

Show a legend for a continuous response variable (scatter plot)

This example shows how marker color in a scatter plot can represent the values of a response variable (WEIGHT in this case).

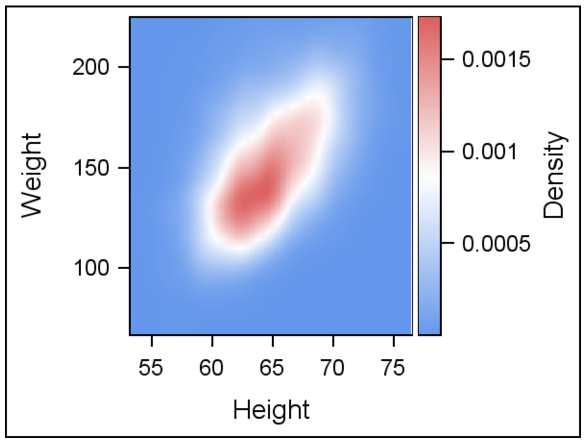
```
layout overlay;
  scatterplot x=age y=height / markercolorgradient=weight name="sc"
    markerattrs=(symbol=circlefilled);
  continuouslegend "sc" / title="Weight";
endlayout;
```



Show a legend for a continuous response variable (contour plot)

This example shows how a fill color gradient in a contour can represent values of a response variable (DENSITY in this case)

```
layout overlay;
  contourplotparm x=height y=weight z=density /
    contourtype=gradient name="con";
  continuouslegend "con" / title="Density";
endlayout;
```



General Legend Features

The following sections discuss several features that are common to both discrete legends and continuous legends.

Positioning Options

Overview

You can include a legend statement in most layout blocks. Most of the time you would simply like to ensure that the legend appears where you want in relation to the plot(s) of the graph. The issues differ, depending on whether you define a single-cell graph or a multi-cell graph. This section discusses single-cell graphs. The discussion of legend placement for multi-cell layouts such as GRIDDED, LATTICE, DATALATTICE, and DATAPANEL appears in the appropriate layout chapter:

- Chapter 9, "Using a Simple Multi-cell Layout," on page 143 (GRIDDED)
- Chapter 10, "Using an Advanced Multi-cell Layout," on page 155 (LATTICE)
- Chapter 11, "Using Classification Panels," on page 185 (DATAPANEL, DATALATTICE, PROTOTYPE)

The following positioning options control a legend's location within its parent layout. They are available only when the legend is nested within an overlay-type layout:

LOCATION= INSIDE | OUTSIDE
: determines whether the legend is drawn inside the plot wall of the cell, or outside the plot wall (and outside the axes). The default is OUTSIDE.

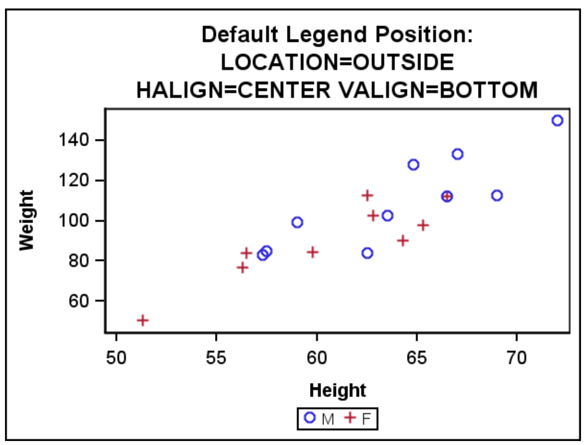
HALIGN = LEFT | CENTER | RIGHT
determines horizontal alignment. The default is CENTER.

VALIGN = TOP | CENTER | BOTTOM
determines vertical alignment. The default is BOTTOM.

Displaying Legends Outside of the Plot Wall

When you place a legend statement in a single-cell layout such as OVERLAY, OVERLAYEQUATED, or OVERLAY3D, the default legend appears outside the plot wall but inside the layout border:

```
layout overlay;
  scatterplot X=Height Y=Weight /
    name="sp" group=sex;
  discretelegend "sp" /
    location=outside
    halign=center valign=bottom ;
endlayout;
```

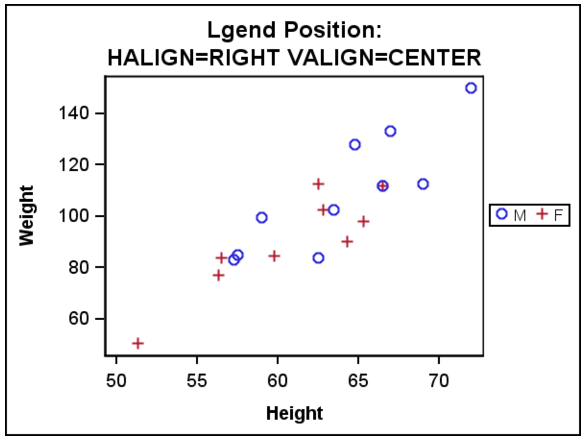


Using the HALIGN= and VALIGN= options, you can place a legend in eight positions outside the plot wall. The only combination that is not supported is HALIGN=CENTER and VALIGN=CENTER. To accommodate the legend, the size of the plot wall is adjusted so that the legend(s) can be displayed.

Note: Sometimes with large legends, this size adjustment causes problems. Sizing issues are discussed in "Arranging Legend Entries into Columns and Rows" on page 132 and "When Discrete Legends Get Too Large" on page 135.

The following example positions the legend in the outside center-right location.

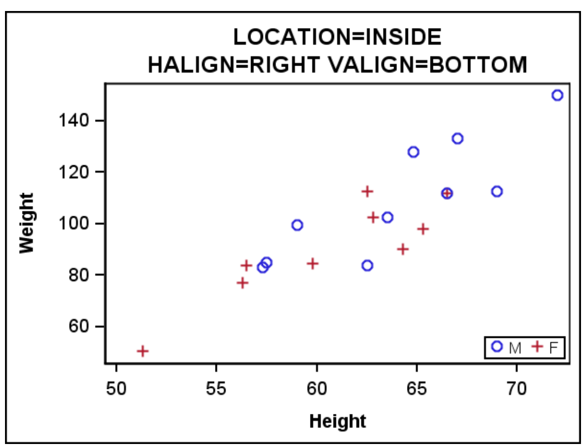
```
layout overlay;
  scatterplot X=Height Y=Weight /
    name="sp" group=sex;
  discretelegend "sp" /
    halign=right valign=center ;
endlayout;
```



Displaying Legends Inside the Plot Wall

A legend can be placed inside the plot wall (LOCATION=INSIDE) and positioned with the HALIGN= and VALIGN= options. Nine inside positions are possible. The defaults are HALIGN=CENTER and VALIGN=CENTER. The following example positions the legend in the inside bottom-right location.

```
layout overlay;
  scatterplot X=Height Y=Weight /
    name="sp" group=sex;
  discretelegend "sp" / location=inside
    halign=right valign=bottom ;
endlayout;
```



One of the advantages of inside legends is that the plot wall does not shrink.

One of the disadvantages of inside legends with HALIGN= and VALIGN= positions is that the legend might be placed on top of plot markers, lines, or filled areas (legends, entries, and nested layouts are always stacked on top of plots, regardless of the statement order in an overlay block).

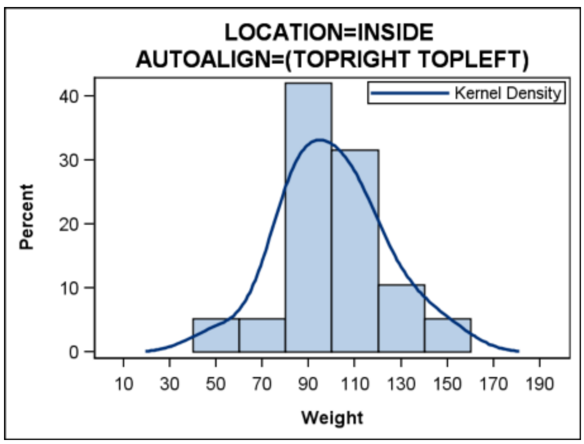
Automatically Aligning an Inside Legend

When the plot statements are specified in a 2D overlay-type layout, the AUTOALIGN= option can be used to automatically position an inside legend. AUTOALIGN= selects a position that avoids or minimizes collision with plot components.

The AUTOALIGN= option enables you to specify an ordered list of potential positions for the legend. The list contains one or more of the following keywords: TOPLEFT, TOP,

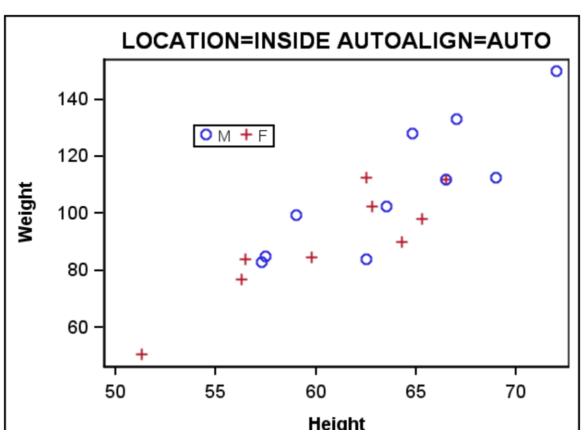
TOPRIGHT, LEFT, CENTER, RIGHT, BOTTOMLEFT, BOTTOM, and BOTTOMRIGHT. In the following example, we know that the best position for an inside legend is TOPRIGHT or TOPLEFT. Because the AUTOALIGN= option specifies a list of preferred positions, the first of the listed positions that does not involve data collision is used. Had the histogram been skewed to the right, the TOPLEFT position would be used.

```
layout overlay;
  histogram Weight / name="sp";
  densityplot Weight / kernel()
    legendlabel="Kernel Density"
    name="kde";
  discretelegend "kde" /
    location=inside
    autoalign=(topright topleft) ;
endlayout;
```



When the parent layout contains only scatter plots, you can fully automate the selection of an internal position by specifying AUTOALIGN=AUTO. This is a "smart" option that automatically selects a position where there is no (or minimal) collision with plot components. The AUTOALIGN=AUTO option selects a position for you. Note that positions that are not possible with HALIGN= and VALIGN= might be used.

```
layout overlay;
  scatterplot X=Height Y=Weight / name="sp" group=sex;
  discretelegend "sp" / location=inside autoalign=auto ;
endlayout;
```



General Appearance Options

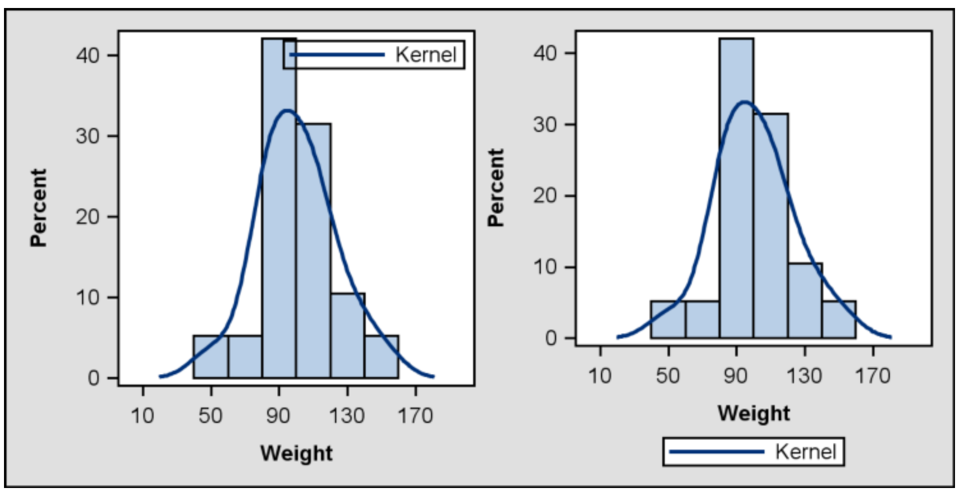
Using Background Transparency and Color

The following options control the appearance of the legend background:

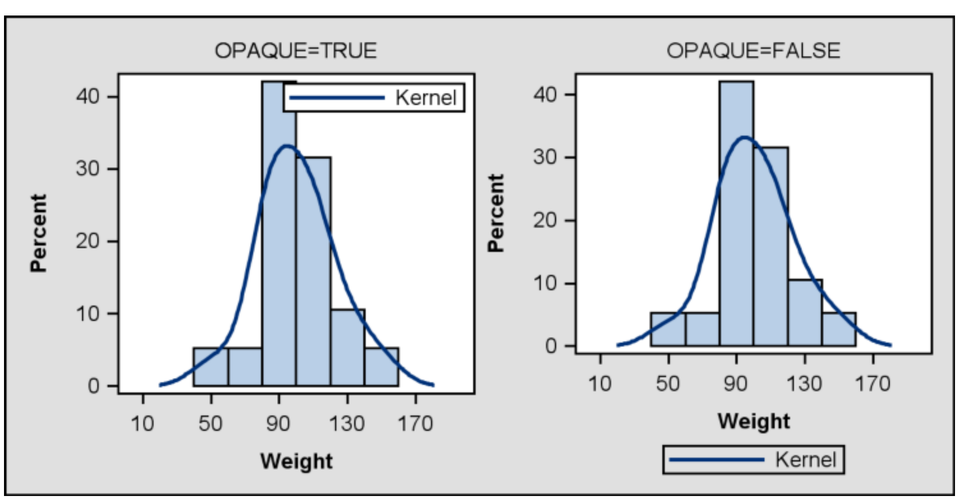
OPAQUE = TRUE | FALSE
determines whether the legend background is 100% transparent or 0% transparent.

BACKGROUNDCOLOR= *style-reference* | *color*
determines legend background color. OPAQUE=TRUE must be set for the background color to be seen. The GraphLegendBackground:Color style reference is the default.

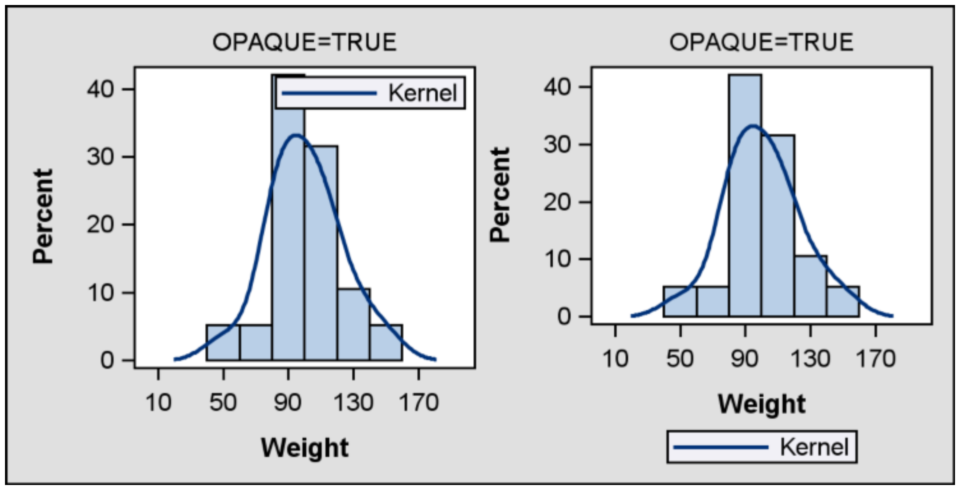
By default, OPAQUE=FALSE when LOCATION=INSIDE. This minimizes the potential for the legend to obscure the markers, lines, fills, and labels in the plot area. when When LOCATION=OUTSIDE, OPAQUE=TRUE by default. This enables the legend background color to appear. Typically, the default legend background color is the same as the plot wall background color. The following graph illustrates the default settings (the graph uses the DEFAULT style, which has a gray graph background):



The next graph illustrates how the graph looks when the default opacity is reversed. With reverse opacity, the default background color of an inside legend is the same as the fill color of the plot wall that is behind it. For outside legends, the default background color is 100% transparent, so the graph background color shows through the legend.



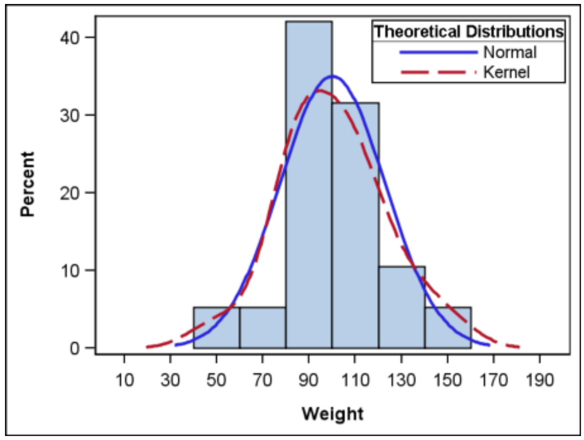
When the legend background is opaque, you can use the BACKGROUNDCOLOR= option to set its color. In the following example, BACKGROUNDCOLOR=GraphAltBlock:Color for both the inside and outside opaque legends. Other style references you could use include GraphHeaderBackground:Color, GraphBlock:Color, or any other style element with a COLOR= attribute. You can also specify a specific color, such as BACKGROUNDCOLOR=white.



Using a Legend Title and Title Border

By default, legends do not have titles. To add a title, you can use the TITLE= option. You can also add a dividing line between the legend title and the legend body with the TITLEBORDER=TRUE setting.

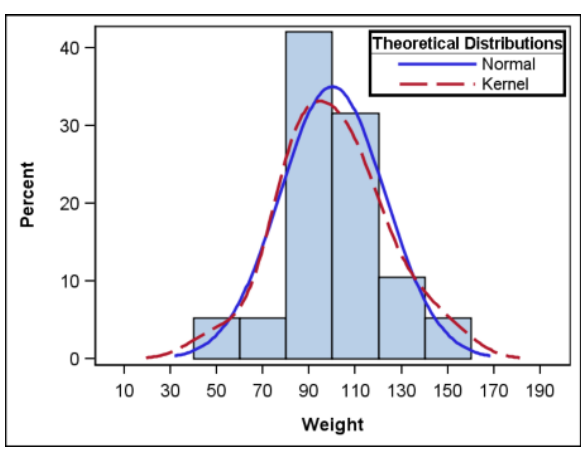
```
layout overlay;
  histogram Weight / name="sp";
    densityplot Weight / normal()
      legendlabel="Normal" name="norm"
      lineattrs=GraphData1;
    densityplot Weight / kernel()
      legendlabel="Kernel" name="kde"
      lineattrs=GraphData2;
    discretelegend "norm" "kde" /
      location=inside across=1
      autoalign=(topright topleft)
      title="Theoretical Distributions"
      titleborder=true ;
endlayout;
```



Legend Border

By default, a border is displayed around a legend. You can remove the border by specifying BORDER=FALSE (which also removes the title border). The line properties of a legend border can be set by the BORDERATTRS= option. The following example modifies the legend border so that it is thicker than the title border:

```
layout overlay;
  histogram Weight / name="sp";
    densityplot Weight / normal()
      legendlabel="Normal" name="norm"
      lineattrs=GraphData1;
    densityplot Weight / kernel()
      legendlabel="Kernel" name="kde"
      lineattrs=GraphData2;
    discretelegend "norm" "kde" /
      location=inside across=1
      autoalign=(topright topleft)
      title="Theoretical Distributions"
      titleborder=true
      borderattrs=(thickness=2) ;
endlayout;
```



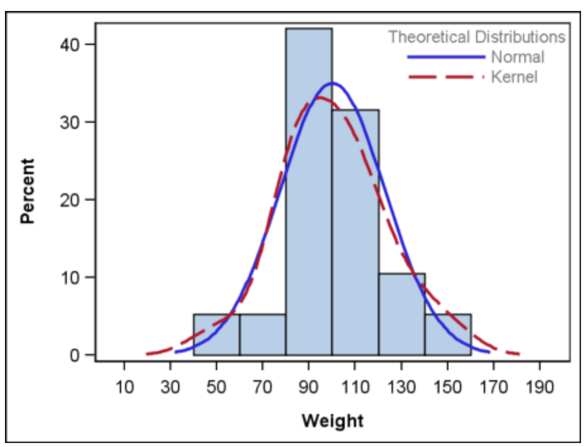
Legend Text Properties

The TITLEATTRS= and VALUEATTRS= options control the text properties of the legend. By default, the text properties come from the current style. The legend title uses **`TITLEATTRS = GraphLabelText`**, and legend entries use **`VALUEATTRS =`**

GraphValueText. For visual consistency in the graph, the GraphLabelText style element is also used for axis labels, and the GraphValueText style element is also used for axis tick values. In general, style elements are used as needed in a graph to maintain visual consistency.

The following example sets all legend text to gray. The font for the legend title is made the same as the default font for the legend values by setting TITLEATTRS=GraphValueText.

```
layout overlay;
  histogram Weight / name="sp";
    densityplot Weight / normal()
      legendlabel="Normal" name="norm"
      lineattrs=GraphData1;
    densityplot Weight / kernel()
      legendlabel="Kernel" name="kde"
      lineattrs=GraphData2;
    discretelegend "norm" "kde" /
      location=inside across=1
      autoalign=(topright topleft)
      title="Theoretical Distributions"
      border=false valueattrs=(color=gray)
      titleattrs=GraphValueText(color=gray) ;
endlayout;
```



Features of Discrete Legends

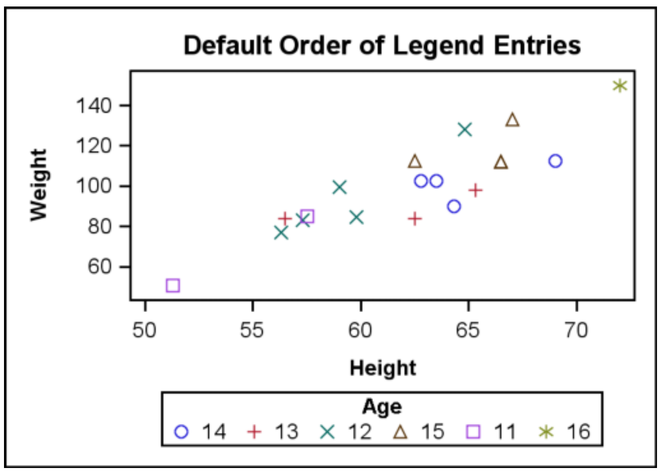
Ordering the Legend Entries for a Grouped Plot

Overivew

When the GROUP=*column* option is used with a plot, the unique values of *column* are presented in the legend in the order that they occur in the data.

```
proc template;
 define statgraph order;
  dynamic TITLE;
  begingraph;
   entrytitle TITLE;
   layout overlay;
```

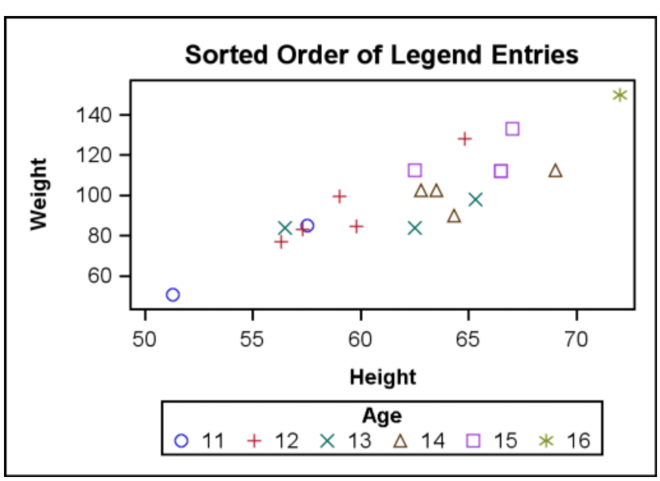
```
      scatterplot x=height y=weight / name="sp"
        group=age ;
      discretelegend "sp" / title="Age";
    endlayout;
  endgraph;
 end;
run;

proc sgrender data=sashelp.class template=order;
  dynamic
    title="Default Order of Legend Entries";
run;
```



Sorting the Data

To see the group values in ascending or descending order, you must sort the input data by *column* before executing the template.

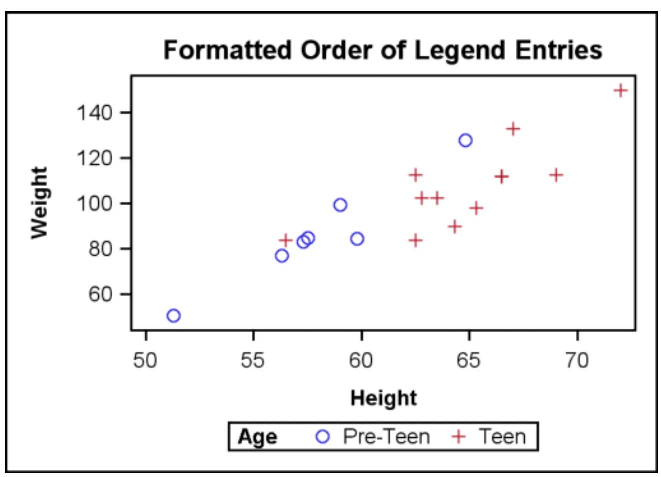
```
proc sort data=sashelp.class out=class;
 by age;
run;

proc sgrender data=class template=order;
  dynamic
    title="Sorted Order of Legend Entries";
run;
```



Formatting the Data

You can apply a format to a group column to change the legend entry labels or the number of classification levels. The ordering of the legend entries is based on the order of the pre-formatted group values. In the following example, the data is sorted in ascending order, so the legend entry order is "Pre-Teen" "Teen" "Adult" (there are no adults, so "Adult" does not appear in the graph). If the data were sorted in descending age order the legend entry order would be reversed.

```
proc format;
  value teenfmt
  low-12  = "Pre-Teen"
  13-19   = "Teen"
  20-high = "Adult";
run;

proc sort data=sashelp.class out=class;
  by age;
run;

proc sgrender data=class template=order;
  format age teenfmt.;
  dynamic
    title="Formatted Order of Legend Entries";
run;
```



In a GTL template, the plot statement, not the legend statement, defines the association of grouped data values with colors, symbols, and line patterns. The association is simply reflected in the legend entries. To change the mapping between grouped data values and the associated style elements, use the INDEX=*column* option on the plot statement. For a discussion of the INDEX= option, see Chapter 6, "Managing Graph Appearance: General Principles," on page 83.

Ordering the Legend Entries for Non-grouped Plots

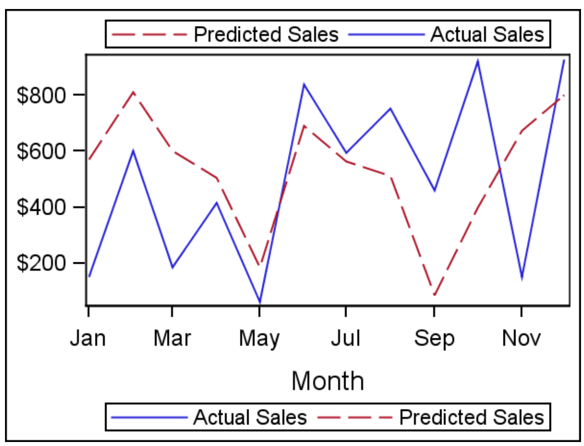
Overview

When plots are overlaid and you want to distinguish them in a legend, you must assign each plot a name and then reference the name in the legend statement. The order in which the plot names appear on the legend statement controls the ordering of the legend entries for the plots.

Varying Visual Properties

In the following examples, the CYCLEATTRS=TRUE setting is used as a quick way to change the visual properties of each plot without explicitly setting it. When CYCLEATTRS=TRUE, any plots that derive their default visual properties from one of the GraphData elements are cycled through those elements for deriving visual properties. So, the first plot gets its visual properties from the GraphData1 style element, the next plot gets its properties from the GraphData2 style element, and so on. When plot lines represent entities such as fit lines or confidence bands, it is recommended that you use options such as LINEATTRS= or OUTLINEATTRS= and specify appropriate style elements. For example, you might specify **`LINEATTRS=GraphFit`** or **`OUTLINEATTRS=GraphConfidence`**.

```
layout overlay / cycleattrs=true
    yaxisopts=(display=(ticks tickvalues));
  seriesplot X=month Y=actual /  name="a";
  seriesplot X=month Y=predict / name="p";

  discretelegend "a" "p" /valign=bottom;
  discretelegend "p" "a" /valign=top;
endlayout;
```

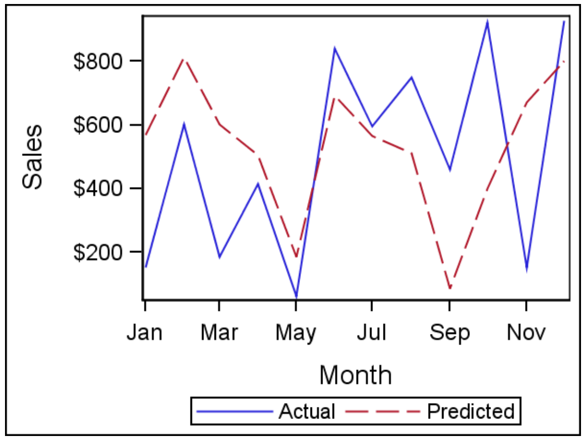


Assigning Legend Entry Labels

Every GTL plot type (except box plot) has a default legend entry label. For example, for some X-Y plots, the default entry legend label is the label of the Y= column (or the column name if no label is assigned).

To assign a legend entry label for a plot, you can use a LABEL statement with PROC SGRENDER, or use the LEGENDLABEL="*string*" option on the plot statement.

```
layout overlay / yaxisopts=(label="Sales")
                 cycleattrs=true;
  seriesplot x=month y=actual / name="a"
           legendlabel="Actual" ;
  seriesplot x=month y=predict / name="p"
           legendlabel="Predicted" ;

  discretelegend "a" "p"/  valign=bottom;
endlayout;
```



Note: Other techniques are available for labeling plots without using a legend. Plots that render one or more lines (SERIESPLOT, STEPPLOT, DENSITYPLOT, REGRESSIONPLOT, LOESSPLOT, PBSPLINEPLOT, MODELBAND, BANDPLOT, LINEPARM, REFERENCELINE, and DROPLINE) all support a CURVELABEL= option that places text inside or outside of the plot wall to label the line(s). Additional options are available to control curve label location, position, and text properties. For examples, see Chapter 6, “Managing Graph Appearance: General Principles,” on page 83 and Chapter 7, “Adding and Changing Text in a Graph,” on page 101.

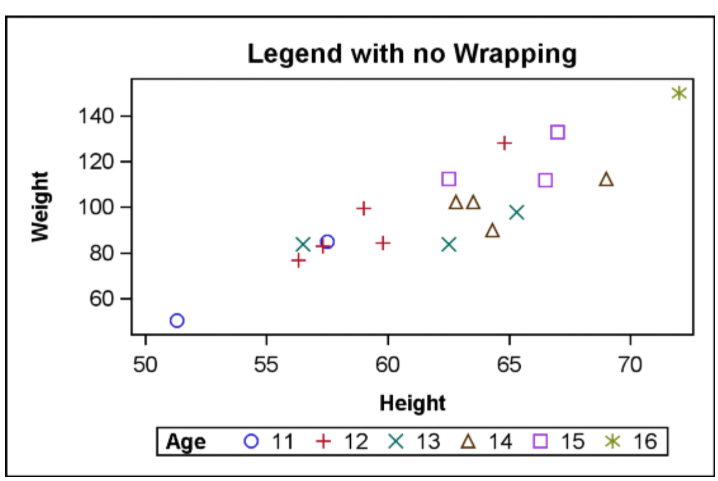
Arranging Legend Entries into Columns and Rows

Overview

When legends have many entries or the legend value labels are long, you might want to control how the legend entries are organized. The following examples show how the size of the graph can affect the default legend organization.

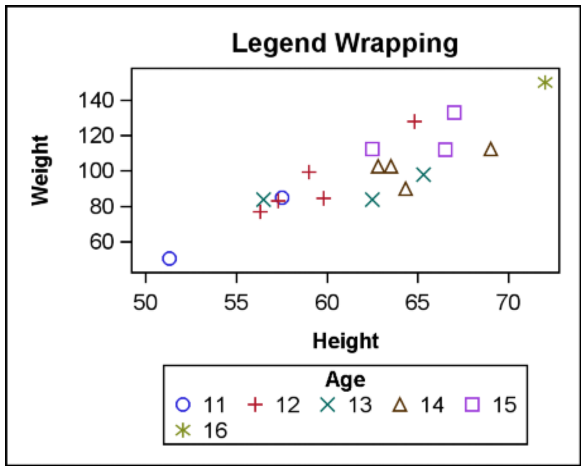
When the graph is wide enough, all legend information can fit in one row.

Note: When all the legend entries and the legend title will fit in one row, the legend title is drawn on the left as shown in the following graph. This is done to conserve the vertical space that is used by the legend.



Legend Wrapping

As the graph gets smaller, the area that is allotted to the legend is reduced. In the following graph, the width of the graph is reduced to the point where it causes the legend entries to wrap into an additional row. Because the legend needs this extra row, the height of the plot wall must be reduced, leaving less room for the data display. Also, because the legend entries and title do not fit in one row, the title is now drawn above the legend entries.



Options to Control Legend Wrapping

You can explicitly control the organization of legend entries with the following options on the legend statement:

ORDER = ROWMAJOR | COLUMMAJOR
: determines whether legend entries are wrapped on a column or row basis. Default is ROWMAJOR.

ACROSS = *number*
: determines the number of columns. Only used with ORDER=ROWMAJOR

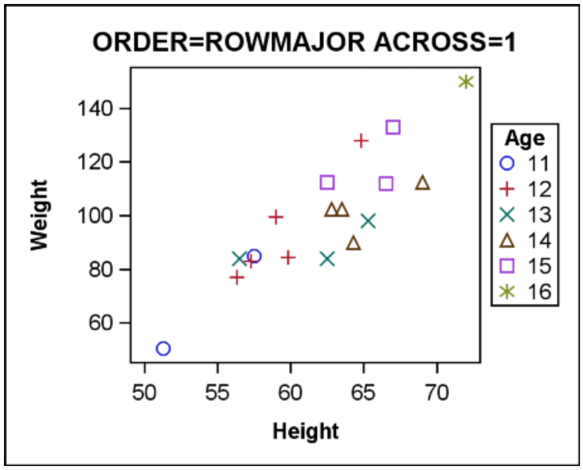
DOWN = *number*
: determines the number of rows. Use only with ORDER=COLUMNMAJOR

DISPLAYCLIPPED = TRUE | FALSE
: determines whether to show a legend when there are too many entries to fit in the available space

Organizing Legend Entries in a Fixed Number of Columns

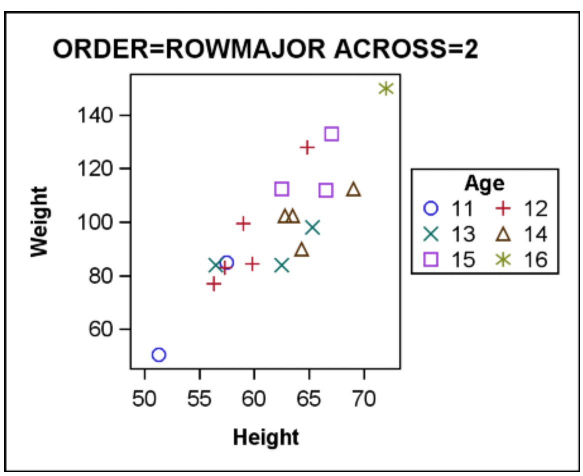
For legends with left or right horizontal alignment, a vertical orientation of legend entries works best because it allows the most space for the plot area. In such cases, you typically want to set a small fixed number of columns for the legend entries and let the entries wrap to a new row whenever necessary. This entails setting ORDER=ROWMAJOR and an ACROSS= value. In the following example, ACROSS=1 means "place all entries in one column, and start as many new rows as necessary."

```
layout overlay;
  scatterplot x=Height y=Weight / name="sp"
    group=age;
  discretelegend "sp" / title="Age"
    halign=right valign=center
    order=rowmajor across=1 ;
endlayout;
```



As you increase the number of columns, the plot area decreases. In the following example, ACROSS=2 means "place all entries in two columns left to right, and start as many new rows as necessary."

```
layout overlay;
  scatterplot x=Height y=Weight / name="sp"
    group=age;
  discretelegend "sp" / title="Age"
    halign=right valign=center
    order=rowmajor across=2 ;
endlayout;
```

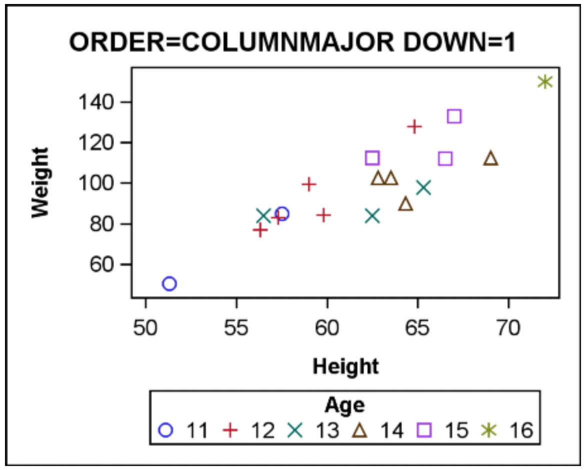


Organizing Legend Entries in a Fixed Number of Rows

For legends with a top and bottom alignment, a horizontal orientation of legend entries works best. In such cases, you typically want to set a small fixed number of rows for the legend entries and let the entries wrap to a new column whenever necessary. This entails setting ORDER=COLUMNMAJOR and a DOWN= value. In the following example, DOWN=1 means "place all entries in one row, and start as many new columns as necessary."

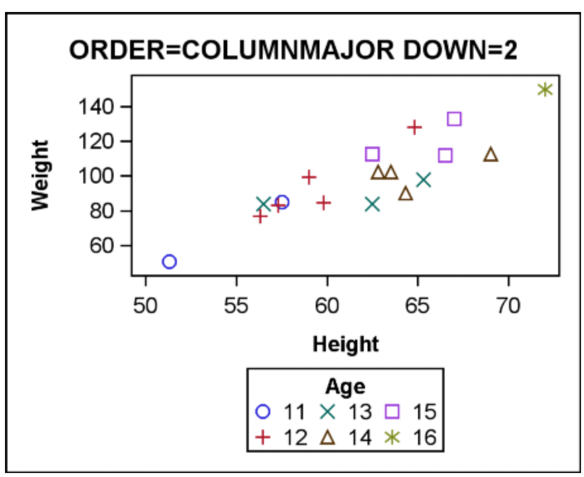
```
layout overlay;
  scatterplot x=Height y=Weight / name="sp"
    group=age;
  discretelegend "sp" / title="Age"
```

```
    order=columnmajor down=1 ;
endlayout;
```



As you increase the number of rows, the plot area decreases. In the following example, DOWN=2 means "place all entries in two rows top to bottom, and start as many new columns as necessary."

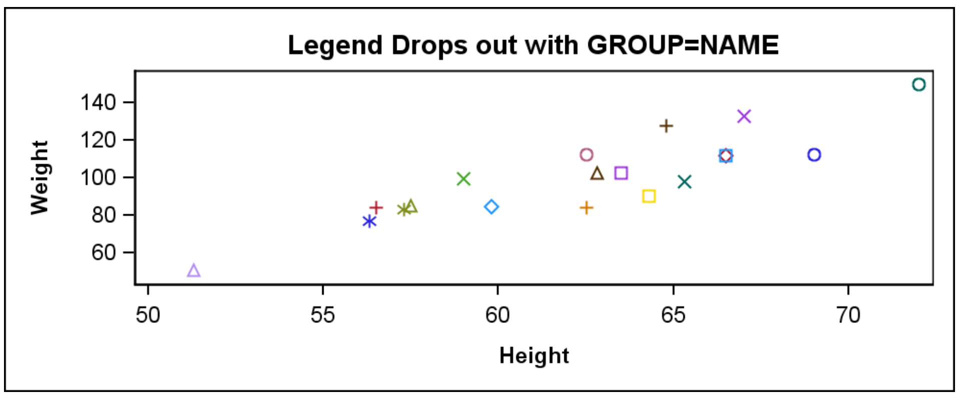
```
layout overlay;
  scatterplot x=Height y=Weight / name="sp"
    group=age;
  discretelegend "sp" / title="Age"
    order=columnmajor down=2 ;
endlayout;
```



When Discrete Legends Get Too Large

As a discrete legend gets more entries or as the legend entry text is lengthy, the legend grows and the plot wall shrinks to accommodate the legend's size. At some point, the plot wall becomes so small that it is useless. For that reason, whenever all the legends in a graph occupy more than 20% of the total area of the graph, the larger legends are dropped as needed from the graph to keep the legend area at 20% or less of the graph area. For example, the following code generates only one legend, but that legend would occupy more than 20% of the total area of the graph, so the legend is dropped and the plot is rendered as if no legend were specified.

```
proc template;
 define statgraph legendsize;
  begingraph;
    entrytitle "Legend Drops out with GROUP=NAME";
    layout overlay;
      scatterplot x=Height y=Weight / name="sp" group=name;
      discretelegend "sp" / title="Name" across=2 halign=right;
    endlayout;
  endgraph;
 end;
run;

proc sort data=sashelp.class out=class; by name; run;

proc sgrender data=class template=legendsize;
run;
```



When the legend is dropped from the graph, you see the following log note:

```
NOTE: Some graph legends have been dropped due to size constraints. Try adjusting
      the MAXLEGENDAREA=, WIDTH= and HEIGHT= options in the ODS GRAPHICS
      statement.
```

In such cases, you can use the WIDTH= and HEIGHT= options in the ODS GRAPHICS statement to increase the graph area so that at some point the legend is displayed.

Another alternative is to use the MAXLEGENDAREA= option to change the threshold area for when legends drop out. The following specification allows all legends to occupy up to 40% of the graph area:

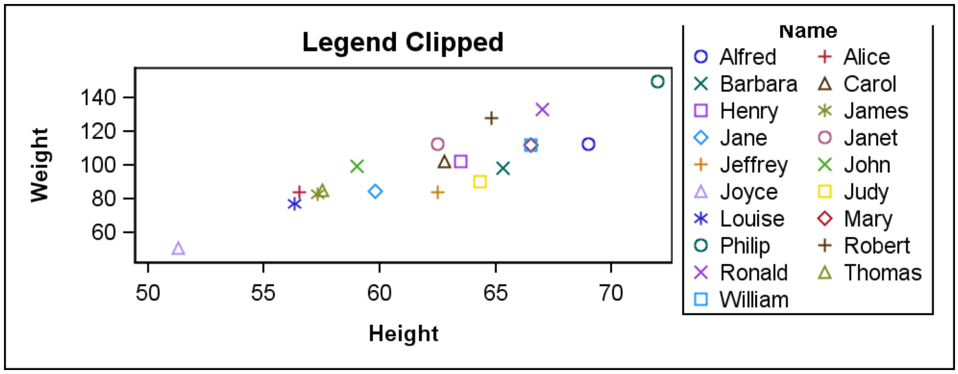
```
ods graphics / maxlegendarea=40;
proc sgrender data=class template=legendsize;
run;
```

However, changing the total area that is allotted to legends might not resolve the problem if the specified legend organization does not fit in the existing size. In these cases, the legend might not be displayed and you would see the following log message:

```
WARNING: DISCRETELEGEND statement with DISPLAYCLIPPED=FALSE is getting clipped.
         The legend will not be drawn.
```

To investigate this problem, you can specify DISPLAYCLIPPED=TRUE in the DISCRETELEGEND statement, which forces the legend to display so that you can visually inspect it.

```
discretelegend "sp" / title="Name" across=2 halign=right displayclipped=true ;
```

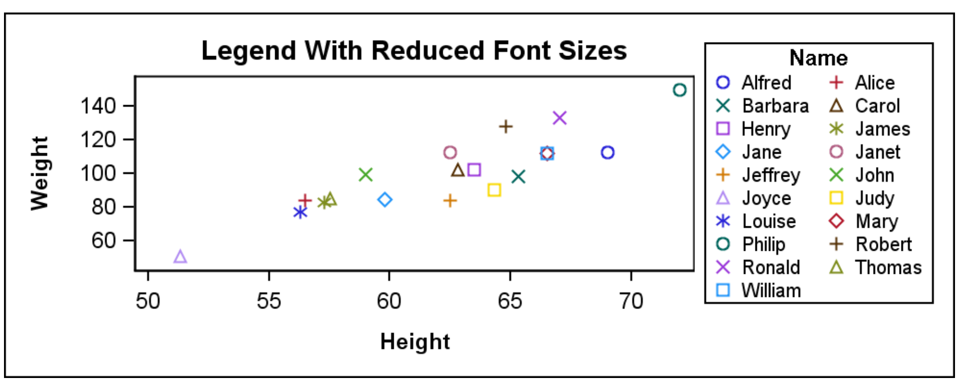


In the current example, it is apparent that the height chosen for the output is not large enough to display the title and all legend entries in two columns. The problem can be fixed in any of the following ways:

- increasing the graph height (HEIGHT= on ODS GRAPHICS statement or DESIGNHEIGHT= on the BEGINGRAPH statement)
- relocating the legend and/or reorganizing it with the ACROSS= or DOWN= options
- setting DISPLAYCLIPPED=TRUE if you are willing to see only a portion of the legend
- reducing the font size for the legend entries (and possibly the title).

To change the font sizes of the legend entries, use the VALUEATTRS= option on the legend statement. To change the font size of the legend title, use the TITLEATTRS= option. Normally, the legend entries are displayed in 9pt font, and the legend title is displayed in 10pt font. The following example reduces the size of legend text:

```
discretelegend "sp" / title="Name" across=2 halign=right
                      valueattrs=(size= 7pt) titleattrs=(size= 8pt);
```



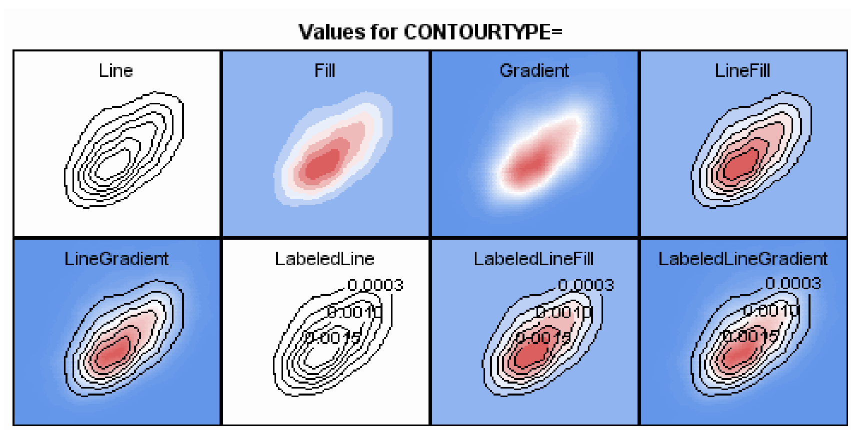
Features of Continuous Legends

Plots That Can Use Continuous Legends

A continuous legend maps the data range of a response variable to a range of colors. Continuous legends can be used with the following plot statements when the enabling plot option is also specified.

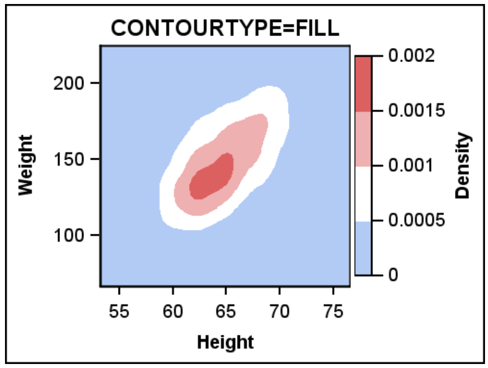
Plot Statement	Enabling Plot Option	Related Plot Options
CONTOURPLOTPARM	CONTOURTYPE=	COLORMODEL= REVERSECOLORMODEL= NLEVELS= NHINT=
SCATTERPLOT	MARKERCOLORGRADIENT=	COLORMODEL= REVERSECOLORMODEL=
SURFACEPLOTPARM	SURFACECOLORGRADENT=	COLORMODEL= REVERSECOLORMODEL=

A contour plot provides the CONTOURTYPE= option, which you can use to manage the contour display. The following graph illustrates the values that are available for the CONTOURTYPE= option.

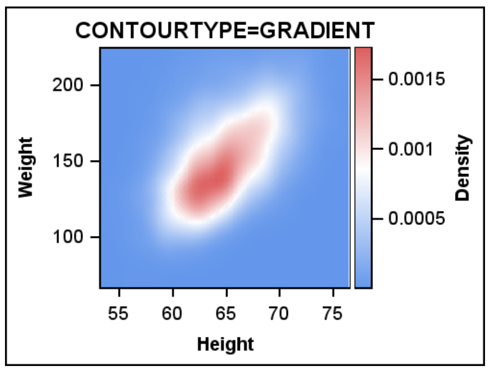


All of the variations that support color, except for LINE and LABELEDLINE, can have a legend that shows the value of the required Z= column. For example, the following code generates a contour plot with CONTOURTYPE=FILL:

```
proc template;
 define statgraph contour;
  begingraph;
   entrytitle "CONTOURTYPE=FILL";
   layout overlay / xaxisopts=(offsetmin=0 offsetmax=0)
                    yaxisopts=(offsetmin=0 offsetmax=0);
     contourplotparm x=Height y=Weight z=Density / name="cont"
       contourtype=fill ;
     continuouslegend "cont" / title="Density";
   endlayout;
  endgraph;
 end;
run;

proc sgrender data=sashelp.gridded template=contour;
 where height>=53 and weight<=225;
run;
```

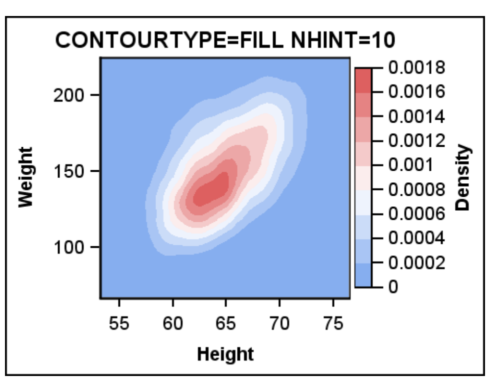


If you change to CONTOURTYPE=GRADIENT you get the following output:



For a FILL contour, the Z variable is split into equal-sized value ranges, and each range is assigned a different color. The continuous legend shows the value range boundaries and the associated colors as a long strip of color swatches with an axis on it. The contour options NHINT= and NLEVELS= are used to change the number of levels (ranges) of the contour. NHINT=10 requests that a number near ten be used that results in "good" intervals for displaying in the legend. NLEVELS=10 forces ten levels to be used.

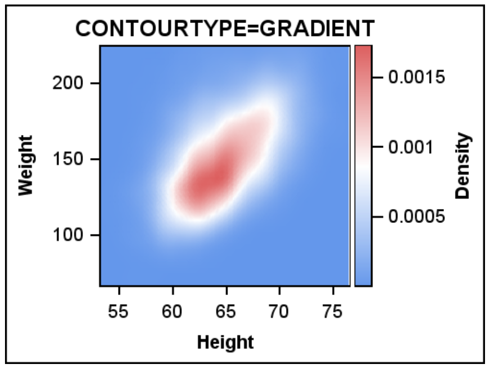
```
contourplotparm x=Height y=Weight
  z=Density / name="cont"
  contourtype=fill nhint=10 ;
continuouslegend "cont" /
  title="Density";
```



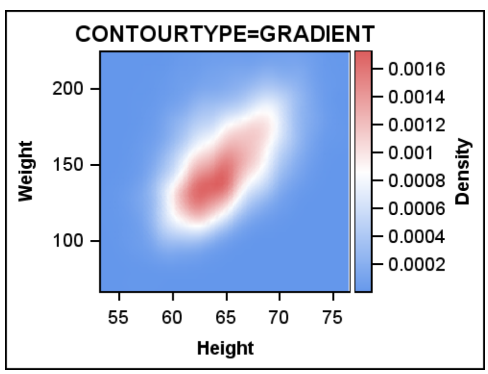
You can think of a GRADIENT contour as a FILL contour with a very large number of levels. A color ramp is displayed with an axis that shows reference points that are within the data range. The number of reference points is determined by default.

When a CONTINUOUS legend is used with a plot that uses gradient color, the VALUESCOUNT= and VALUESCOUNTHINT= options can be used to manage the legend's gradient axis. These options are similar to the NLEVELS= and NHINT= plot options.

```
continuouslegend "cont" /
  title="Density"
  valuecounthint=5 ;
```




```
continuouslegend "cont" /
  title="Density"
  valuecounthint=10 ;
```



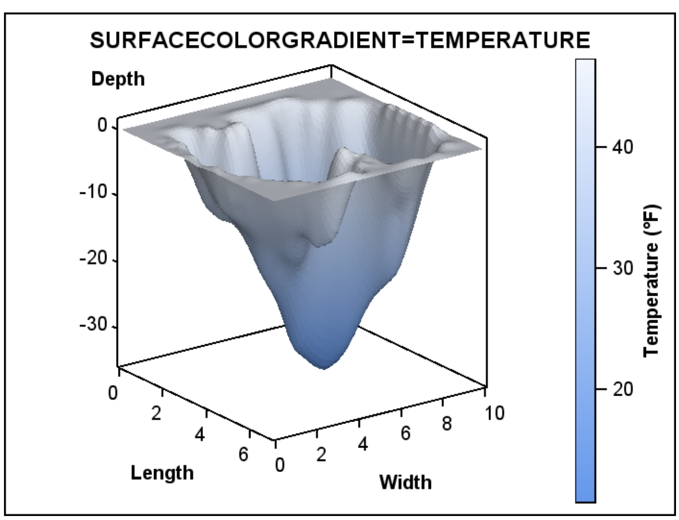
Positioning a Continuous Legend

The ACROSS= , DOWN= and ORDER= options are not supported by the CONTINUOUSLEGEND statement. However, you can position a continuous legend with the LOCATION=, HALIGN=, VALIGN=, and ORIENT= options. By default, LOCATION=OUTSIDE and ORIENT=VERTICAL when HALIGN=RIGHT or HALIGN=LEFT.

Using Color Gradients to Represent Response Values

Contour plots and surface plots support the use of color gradients to represent response values. For example, the SURFACEPLOTPARM statement provides the SURFACECOLORGRADIENT=*numeric-column* setting to map surface colors to a continuous gradient and enable the use of a continuous legend. All surface types (FILL, FILLGRID, and WIREFRAME) can be used. The COLORMODEL= and REVERSECOLORMODEL= options also apply. For more information on surface plots, see Chapter 13, “Using 3D Graphics,” on page 233.

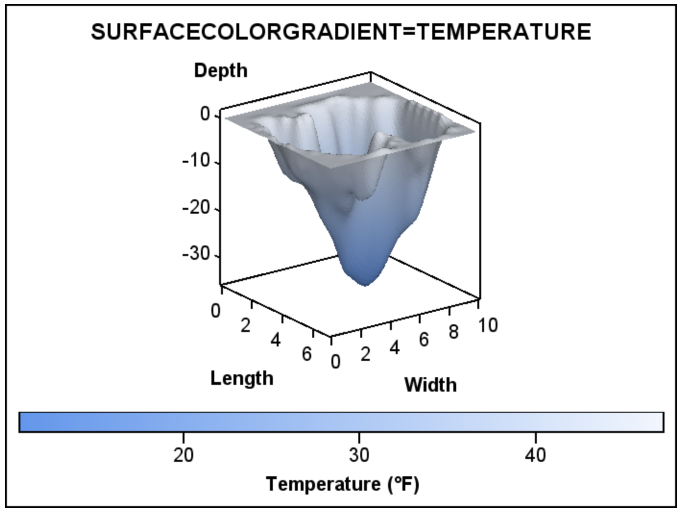
```
proc template;
  define statgraph surfaceplot;
   begingraph;
    entrytitle "SURFACECOLORGRADIENT=TEMPERATURE";
    layout overlay3d / cube=false;
      surfaceplotparm x=length y=width z=depth / name="surf"
        surfacetype=fill
        surfacecolorgradient=temperature
        reversecolormodel=true
        colormodel=twocoloraltramp ;
      continuouslegend "surf" /
        title="Temperature ((*ESC*){unicode '00B0'x}F)"
        halign=right ;
    endlayout;
   endgraph;
  end;
run;

data lake;
  set sashelp.lake;
  if depth = 0 then Temperature=46;
  else Temperature=46+depth;
run;

/* create smoothed interpolated spline data for surface */
proc g3grid data=lake out=spline;
   grid width*length = depth temperature / naxis1=75 naxis2=75 spline;
run;

proc sgrender data=spline template=surfaceplot;
run;
```



When you use VALIGN=BOTTOM or VALIGN=TOP instead of the HALIGN= option , then the default orientation of the legend automatically becomes ORIENT=HORIZONTAL:

```
continuouslegend "surf" /
  title="Temperature ((*ESC*){unicode '00B0'x}F)"
  valign=bottom ;
```

Notice the coding that is used to embed a degree symbol into the legend title. For more information on using symbols in text, see Chapter 7, “Adding and Changing Text in a Graph,” on page 101.



Chapter 9

Using a Simple Multi-cell Layout

The LAYOUT GRIDDED Statement

The GTL provides several layout types to organize your graph into smaller regions (cells). The GRIDDED and LATTICE layouts support a regular grid of cells with a fixed number of rows and columns. The DATALATTICE and DATAPANEL layouts generate classification panels, which are graphs where the number of cells and the cell content are determined by the values of one or more classification variables.

The GRIDDED layout differs from the classification panel layouts in that the number of cells must be predefined and that you must define of the content of each cell separately. GRIDDED is superficially similar to a LATTICE layout because it can create a grid of heterogeneous plots. However, the LATTICE layout can automatically align plot areas across columns and rows and has much more functionality. For more information on the LATTICE layout, see Chapter 10, “Using an Advanced Multi-cell Layout,” on page 155.

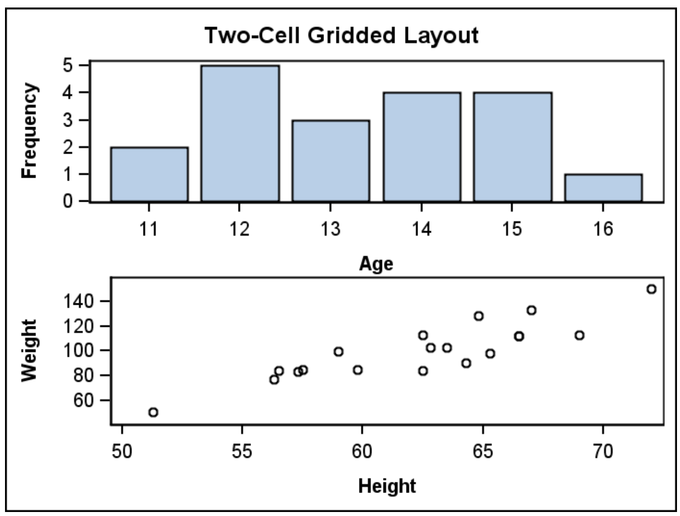
Typical applications of GRIDDED layouts are to create

- a table of text, such as an inset (discussed in detail in Chapter 16, “Adding Insets to a Graph,” on page 271)
- a simple grid of plots (discussed in this chapter).

In a GRIDDED layout, each cell is independent. Contents of the cell can be specified by a stand-alone plot statement or a nested layout. The following example shows a very simple GRIDDED layout:

```
proc template;
  define statgraph intro;
```

```
    begingraph;
      entrytitle "Two-Cell Gridded Layout";
      layout gridded;
        barchart x=age;
        scatterplot x=height y=weight;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.class template=intro;
run;
```



In this case, each plot statement is considered independent and is placed in a separate cell. When no grid size is provided, the default layout creates a graph with one column of cells, and it allots each cell the same amount of space. The number of rows in the grid is determined by the number and arrangement of stand-alone plot statements and nested layouts in the GRIDDED layout block.

Defining a Basic Grid

Although you can generate a nice looking graph in a default GRIDDED layout, in most cases you will want more control over the grid, how it is populated, and the complexity of the cell contents.

Setting Grid Dimensions

Assume you want a grid of five plots. Before starting to write code, you must first decide what grid dimensions you want to set (how many columns and rows) and whether you want to permit an empty cell in the grid. If do not want an empty cell, you must limit the grid to five cells, which gives you two choices for the grid dimensions: five columns by one row (5x1), or one column by five rows (1x5).

To specify the grid size, you use the COLUMNS= or ROWS= option in the LAYOUT GRIDDED statement. To use ROWS=, you must also specify ORDER=COLUMNMAJOR.

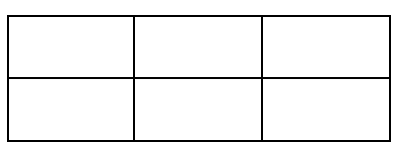
Two explicit specifications could be used to create the following grid, which contains one row and five columns:

`layout gridded / columns=5;`
`  /* plot defintions */`
`endlayout;`
	When the number of columns is specified, you place a limit on how many columns can be displayed across a row. The COLUMNS= option is honored only if ORDER=ROWMAJOR (the default).
	In the example code to the left, if you were to include more than five plot definitions, additional rows (with five columns) would be added automatically to accommodate all of the cells that are needed to display all specified plot definitions.

`layout gridded / order=columnmajor`
`                             rows=1;`
`  /* plot definitions */`
`endlayout;`
	When the number of rows is specified, you place a limit on how many rows can be displayed down a column. The ROWS= option is honored only if ORDER=COLUMNMAJOR.
	In the example code to the left, if you were to include more than five plot definitions, additional columns would be added automatically, but the grid would not wrap to a second row because the ROWS= setting limits the grid to a single row.

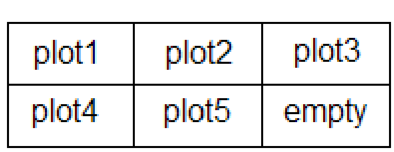
If you are willing to have an empty cell in the grid, you could use a 2x3 or a 3x2 grid:

```
layout gridded / columns=3 ;
endlayout;
```



By default, the layout uses the ORDER=ROWMAJOR setting to populate grid cells. This specification essentially means "fill in all cells in the top row (starting at the top left) and then continue to the next row below." COLUMNS=1 by default when ORDER=ROWMAJOR, so you must specify an alternative setting to increase the number of columns in the grid:

```
layout gridded / columns=3 ;
  /* plot1 definition */
  /* plot2 definition */
  /* plot3 definition */
  /* plot4 definition */
  /* plot5 definition */
endlayout;
```



Alternatively, you can specify ORDER=COLUMNMAJOR, which means "fill in all cells in the left column and then continue to the next column to the right." ROWS=1 by default when ORDER=COLUMNMAJOR, so you must specify an alternative setting to increase the number of rows in the grid:

```
layout lattice / rows=2 order=columnmajor ;
  /* plot1 definition */
  /* plot2 definition */
  /* plot3 definition */
  /* plot4 definition */
  /* plot5 definition */
endlayout;
```

plot1	plot3	plot5
plot2	plot4	empty

Setting Gutters

To conserve space, the default GRIDDED layout does not include a gap between cell boundaries. In some cases, this might cause the cell contents to appear too congested. You can add a vertical gap between all cells with the COLUMNGUTTER= option, and you can add a horizontal gap between all rows with the ROWGUTTER= option. If no units are specified, pixels (PX) are assumed.

```
layout gridded / columns=3 columngutter=5 rowgutter=5 ;
  /* plot1 definition */
  /* plot2 definition */
  /* plot3 definition */
  /* plot4 definition */
  /* plot5 definition */
endlayout;
```

plot1	plot2	plot3
plot4	plot5	empty

Note that by adding gutters, you do not increase the size of the graph. Instead, the cells shrink to accommodate the gutters. Depending on the number of cells in the grid and the size of the gutters, you will frequently want to adjust the size of the graph to obtain optimal results, especially if the cells contain complex graphs. For more information, see “Sizing Issues” on page 150

Defining Cells

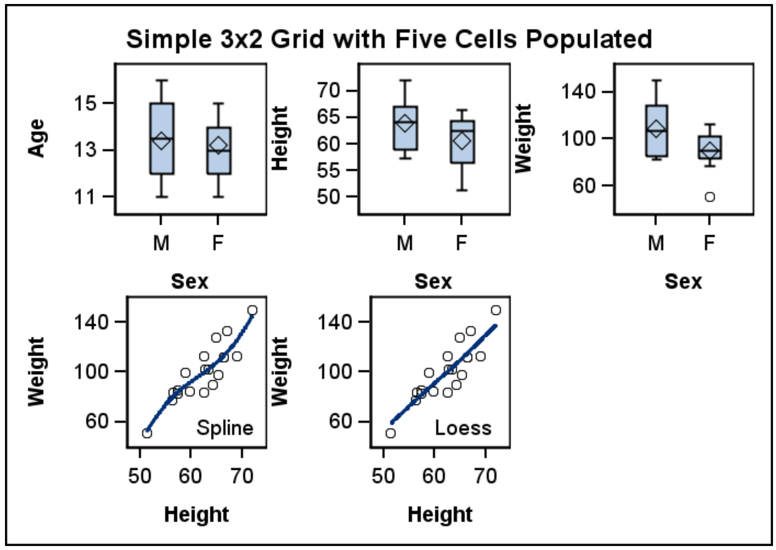
Two valid techniques are available for indicating the contents of a cell:

Technique	Example	Advantages	Disadvantages
stand-alone plot statement or text statement	`scatterplot x= y=;`	simplicity	can't have overlays can't adjust axes, borders, or backgrounds (these are layout options)

layout block	`layout overlay;`
	`scatterplot x= y=;`
	`seriesplot x= y=;`
	`endlayout;`
		cell can contain a complex plot
		axes can be adjusted
		other layout types can be used
			more complexity

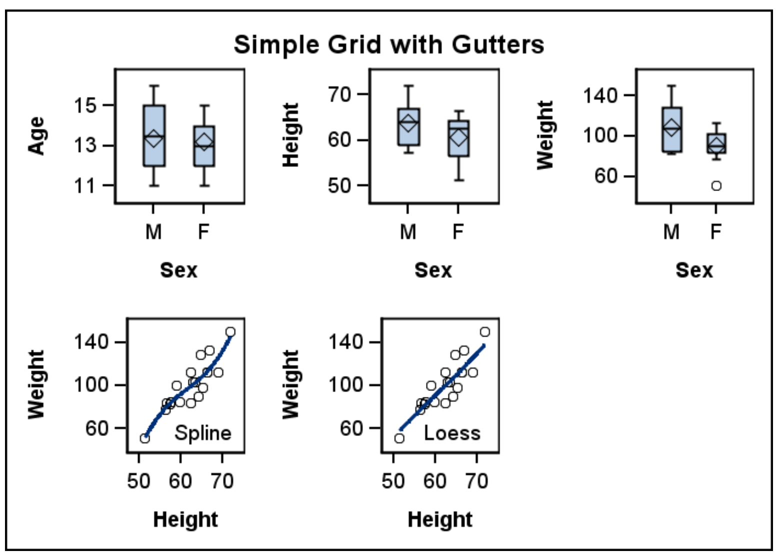
The following definition for a GRIDDED layout shows a simple example:

```
entrytitle "Simple 3x2 Lattice with Five Cells Populated";

layout gridded / columns=3;

  /* stand-alone plot statements define cells 1-3 */
  boxplot x=sex y=age;
  boxplot x=sex y=height;
  boxplot x=sex y=weight;

  /* overlay blocks define cells 4-5 */
  layout overlay;
    scatterplot y=weight x=height;
    pbsplineplot y=weight x=height;
    entry halign=right "Spline" / valign=bottom;
  endlayout;

  layout overlay;
    scatterplot y=weight x=height;
    loessplot y=weight x=height;
    entry halign=right "Loess " / valign=bottom;
  endlayout;

endlayout;
```



Notice that some Y-axis labels are too close to their neighboring plots. You can use the COLUMNGUTTER= and ROWGUTTER= options to add gutters between all columns and rows. The following layout statement defines a grid with 30-pixel gutters:

```
layout gridded / columns=3 columngutter=30 rowgutter=30 ;
```



Notice that adding gutters visually separates graphs, but it does not increase the overall graph size. To compensate for the gutters, the cells become smaller. This same behavior is observed by other multi-cell layouts, as well.

Building a Table of Text

Using a Single Layout

One of the most common applications of the GRIDDED layout is to build a table of text or statistics using nested ENTRY statements.

```
layout gridded / columns=2 order=rowmajor
                 border=true columngutter=5px;
  /* row 1 */
  entry halign=left "N";
  entry halign=left "5203";
  /* row 2 */
  entry halign=left "Mean";
  entry halign=left "119.96";
  /* row 3 */
  entry halign=left "Std Dev";
  entry halign=left "19.98";
endlayout;
```

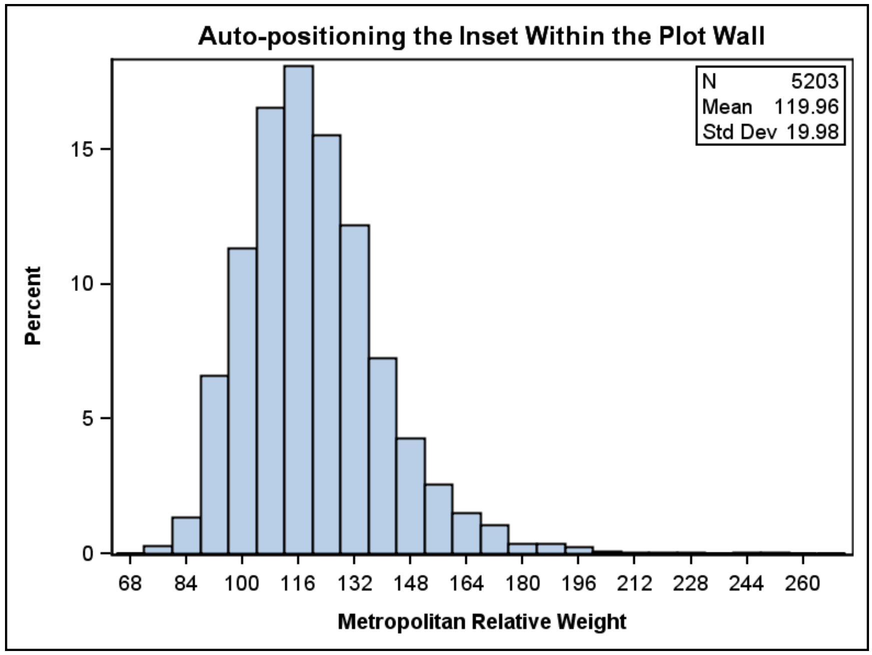
N	5203
Mean	119.96
Std Dev	19.98

Tables like this can be organized many different ways. For more information on these techniques, see Chapter 7, “Adding and Changing Text in a Graph,” on page 101 for details about ENTRY statements, and see Chapter 16, “Adding Insets to a Graph,” on page 271 for details about defining the tables.

Using Nested Layouts

When GRIDDED layouts are used to create tables of text, the tables often appear within another layout. For example, the table might be used within the plot wall of an OVERLAY layout, or within a SIDEBAR block of a LATTICE layout. When the table is used within a LAYOUT OVERLAY, it is often necessary to position the table so that it avoids collision with the plot. In the following example, the AUTOALIGN=(*position-list*) option of the GRIDDED layout is used to dynamically position the table in the TOPRIGHT or TOPLEFT position. TOPRIGHT is tried first, but TOPLEFT is used if the TOPRIGHT position would cause the histogram to collide with the table.

```
proc template;
define statgraph inset2;
  begingraph;
    entrytitle "Auto-positioning the Inset Within the Plot Wall";
    layout overlay;
      histogram mrw;
      layout gridded / columns=1 border=true
                       columngutter=5px
                       autoalign=(topright topleft);
        entry  halign=left "N"       halign=right "5203";
        entry  halign=left "Mean"    halign=right "119.96";
        entry  halign=left "Std Dev" halign=right "19.98";
      endlayout;
    endlayout;
  endgraph;
end;
run;

proc sgrender data=sashelp.heart template=inset2;
run;
```



In this example, the values for the statistics in the table are hard-coded. Obviously, you would prefer that the statistics values be calculated in the template. Chapter 16, “Adding Insets to a Graph,” on page 271 shows how these values can be computed in the template or passed to the template using dynamic or macro variables.

Sizing Issues

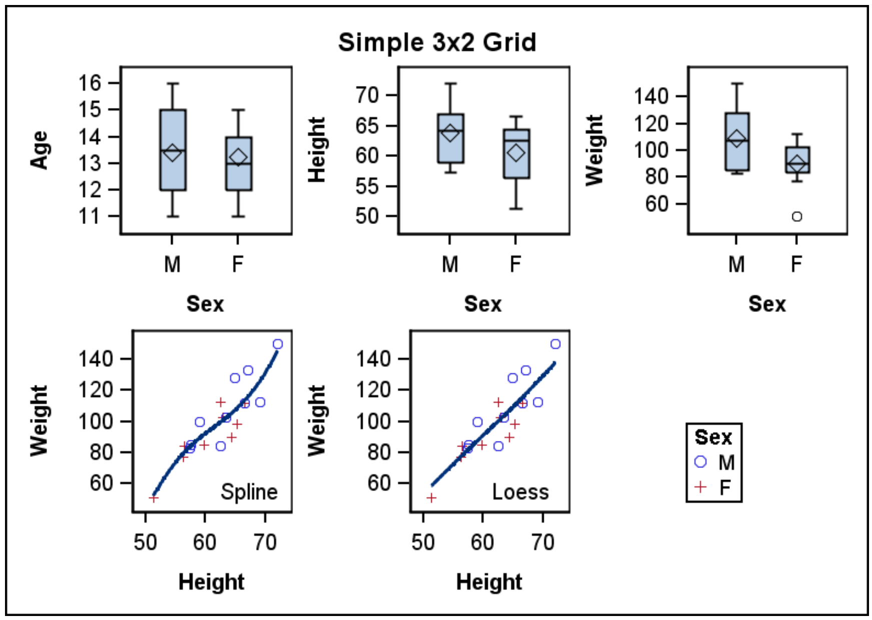
Row and Column Sizes

Unlike the LATTICE layout, the GRIDDED layout offers no way to control column sizes or row sizes. These sizes are determined by the contents of the cells. If only plots are used in the cells, the grid is partitioned equally based on the graph size. However, any individual cell in the grid might contain a legend or text. Consider the next two examples, in which the sixth cell of the grid is populated with a legend.

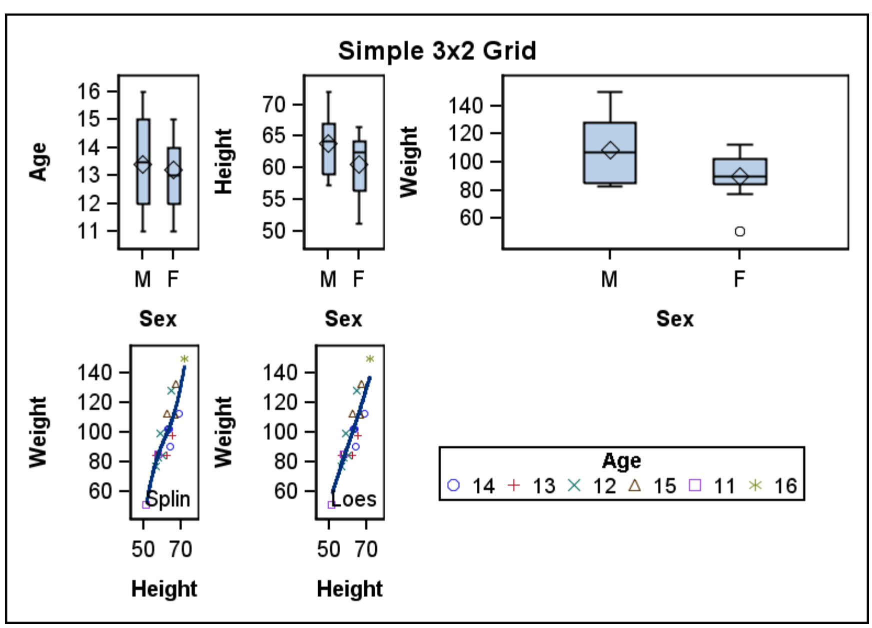
```
layout gridded /  columns=3 rows=2 columngutter=10 rowgutter=10;
  /* standalone plot statements define cells 1-3 */
     boxplot x=sex y=age;
     boxplot x=sex y=height;
     boxplot x=sex y=weight;

  /* overlay blocks define cells 4-5 */
     layout overlay;
       scatterplot y=weight x=height / group=sex name="scatter" ;
       pbsplineplot y=weight x=height;
       entry halign=right "Spline" / valign=bottom;
         endlayout;

     layout overlay;
       scatterplot y=weight x=height / group=sex;
       loessplot y=weight x=height;
       entry halign=right "Loess " / valign=bottom;
```

```
        endlayout;

      /* legend defines cell 6 */
        discretelegend "scatter" / title="Sex";
    endlayout;
```

In this first case, the legend height and width are smaller than the default column and rows sizes, so the legend fits nicely into the empty cell.

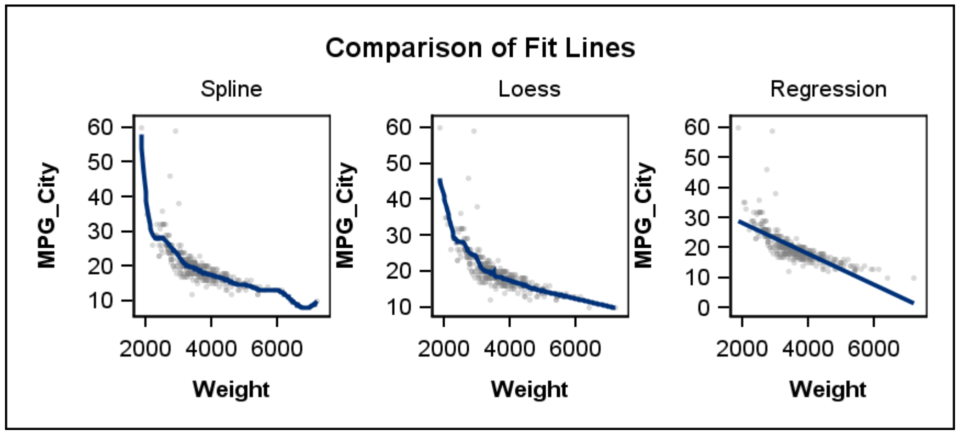


However, this second case demonstrates that if the legend is larger than the default column width or row height, the legend size has precedence and the cell size is adjusted to fit the legend. The same thing might happen when ENTRY statements with lengthy strings are used in cells.

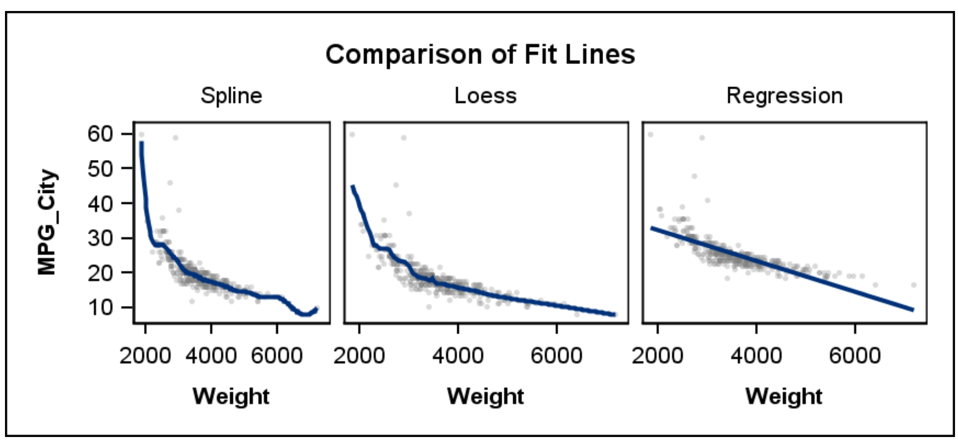


Because of this behavior, you should consider using a LATTICE layout whenever you want to enforce uniform or user-defined column widths and row heights for the grid, regardless of cell contents. If this layout were changed to a LATTICE, the legend would be either omitted or clipped, depending on the setting of the DISPLAYCLIPPED= option of the DISCRETELEGEND statement.

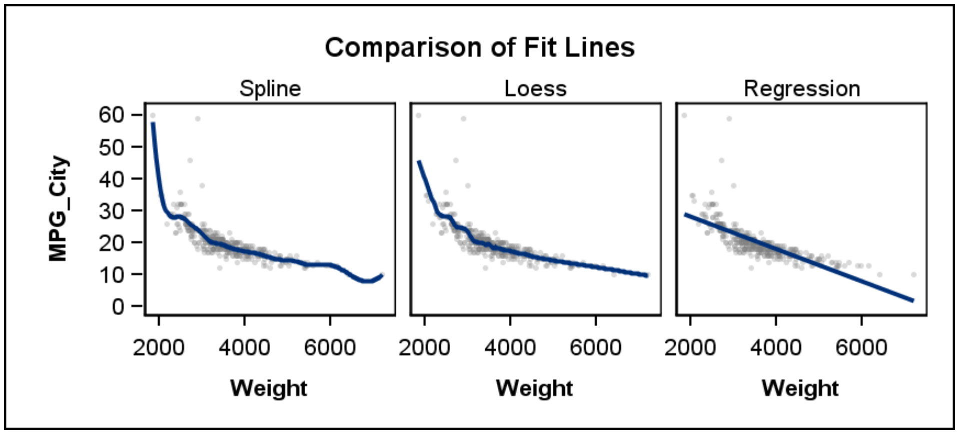
Even when the GRIDDED layout does not contain legend or text statements, the plot-area size in a row or column in the grid might be changed by cell contents. Consider this three-cell GRIDDED layout with OVERLAY layouts defining each cell.



Because the Y axes are duplicated across cells, you might try to conserve space by turning off the Y axes for the second and third cells. You can do this with the YAXISOPTS=(DISPLAY=NONE) option of the OVERLAY layout. Here is the result:



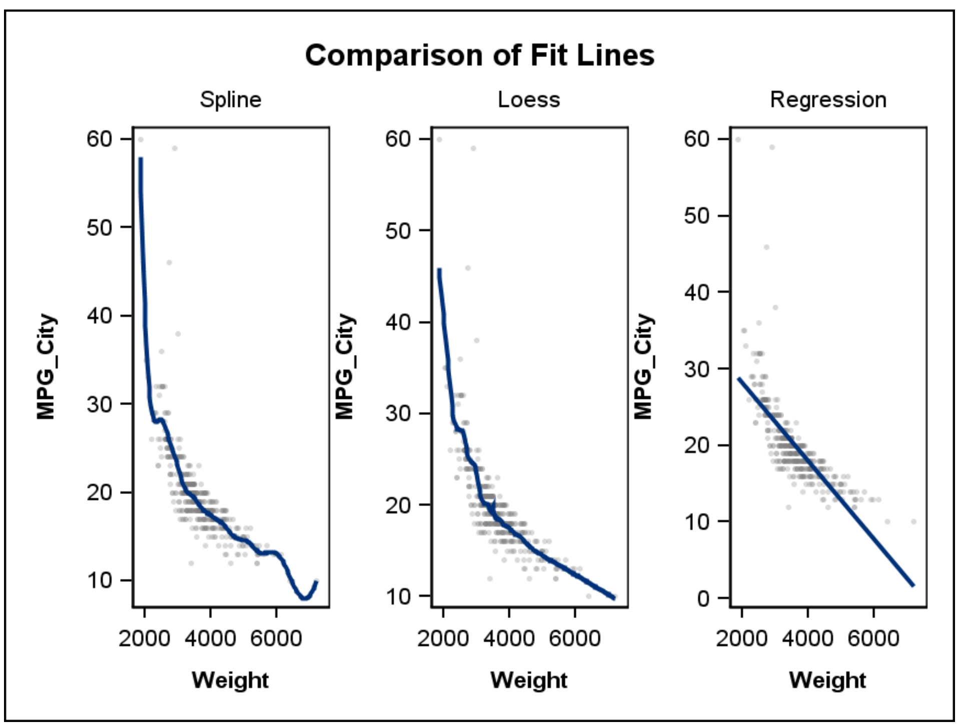
Once again, the three cells have the same size, but the plot areas do not because the cells that no longer display the Y axis have extended the plot areas into the space that formerly displayed the axes. Rather than using the GRIDDED layout, you can use the LATTICE layout to ensure that the three plot areas have the same size:



This graph was produced with LATTICE layout with an external axis. See Chapter 10, "Using an Advanced Multi-cell Layout," on page 155 for details.

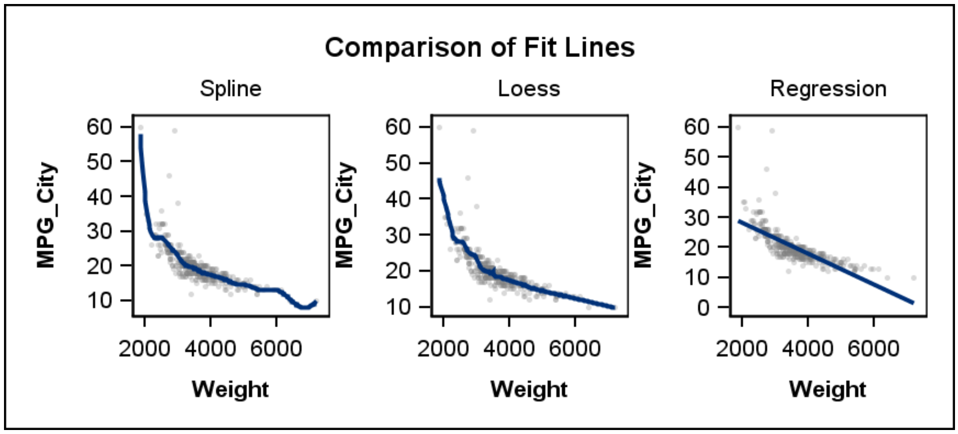
Adjusting Graph Size

When defining the grid size, you will generally have some idea of a good overall aspect ratio for the graph. For example, if you are creating a one row by three column grid, the graph has a default aspect ratio of 4:3 and looks as follows:



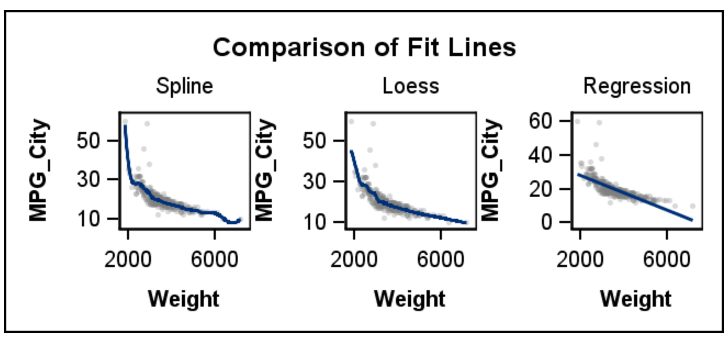
The graph would look better if the graph height were smaller in relation to the width. You can establish a good default graph size in the template definition by setting the DESIGNWIDTH= and DESIGNHEIGHT= options in the BEGINGRAPH statement. After some experimentation, you might decide that something closer to a 2:1 aspect ratio looks good:

```
begingraph / designwidth=400px designheight=180px;
```



The DESIGNWIDTH= and DESIGNHEIGHT= options set the graph size as part of the template definition so that if you later want a larger or smaller version of this graph, you can use the ODS GRAPHICS statement rather than resetting the design size and recompiling the template. You need only specify either a WIDTH= or a HEIGHT= option in the ODS GRAPHICS statement. The other dimension is automatically computed for you, based on the aspect ratio that is specified in the compiled template by the DESIGNWIDTH= and DESIGNHEIGHT= options.

```
ods graphics / reset width=375px;
proc sgrender data=sashelp.cars template=fitcompare;
run;
```



If you provide both the HEIGHT= and WIDTH= options in the ODS GRAPHICS statement, you completely override the design aspect ratio. If the WIDTH= or HEIGHT= options are not specified, the design size is in effect.

Setting the DESIGNHEIGHT= and DESIGNWIDTH= options is highly recommended for all multi-cell layouts that contain plots. This recommendation applies to the GRIDDED, LATTICE, DATAPANEL, and DATALATTICE layouts.

Chapter 10
Using an Advanced Multi-cell Layout

The LAYOUT LATTICE Statement

The LAYOUT LATTICE statement defines a multi-cell grid of graphs that can automatically align plot areas and tick display areas across grid cells to facilitate data comparisons among plots. The LATTICE layout differs from the classification panel layouts in that the number of cells must be predefined and that you must define the content of each cell separately. LATTICE is superficially similar to a GRIDDED layout because it can create a grid of heterogeneous plots. However, the LATTICE has much more functionality and supports the following:

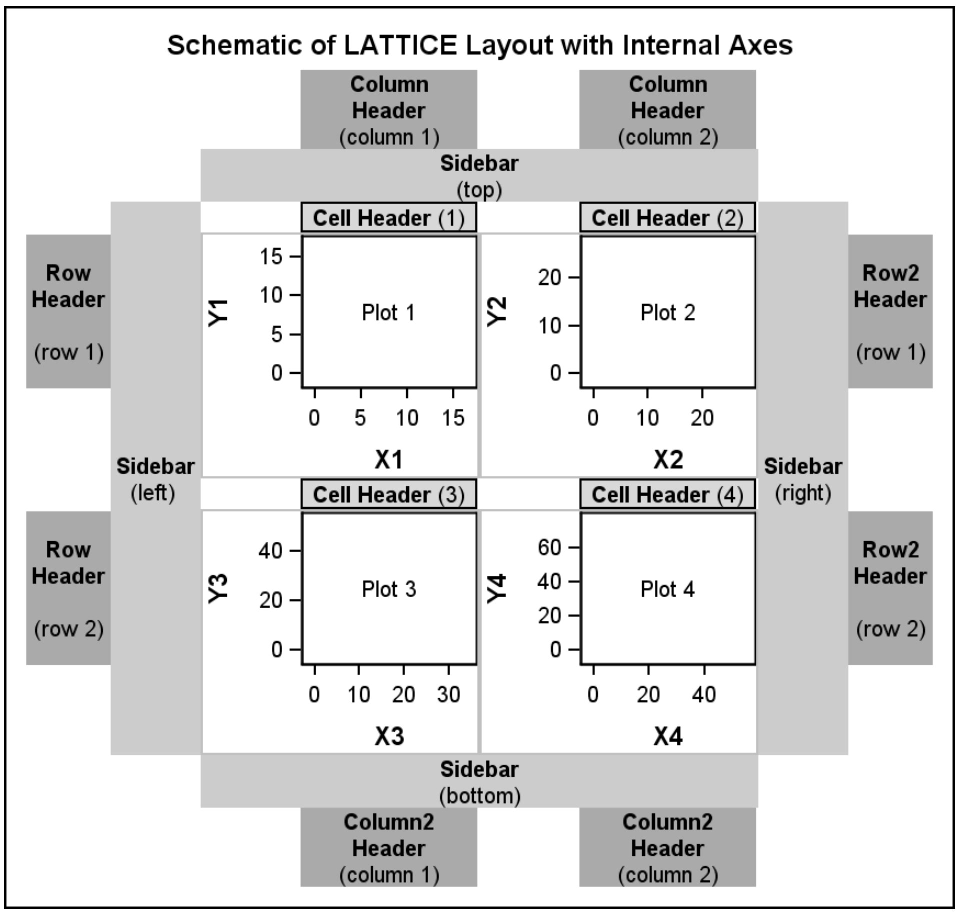
- adjustable column and row sizes
- axis equalization on a row or column basis to facilitate comparisons
- internal axes on a per-cell basis, or external axes for rows or columns of cells
- internal labeling of cell contents (cell header)
- external labeling of rows and columns (column and row headers)

- external sidebars that span all columns (top and bottom) or rows (left and right).

Figure 10.1 on page 156 shows a four-cell grid (two rows and two columns). It was produced with a LATTICE layout to illustrate the features of this layout type. The figure contains definitions of four plots, which by default are treated independently.

A mixture of plot types or nested layouts could be used in the cells of the lattice. By default, each plot manages its own axes internal to the lattice boundaries. In the figure, a light gray border has been added to each plot to show its boundaries within the lattice. The shaded areas represent the optional features that you can add to the lattice definition. By default, these shaded areas are not used in the lattice and space is not reserved for them. Thus, in the default case, the plot areas would expand to replace the shaded areas in the cells.

Figure 10.1 *LATTICE Layout with Internal Axes*



The shaded areas that are shown in the figure are typically used as follows:

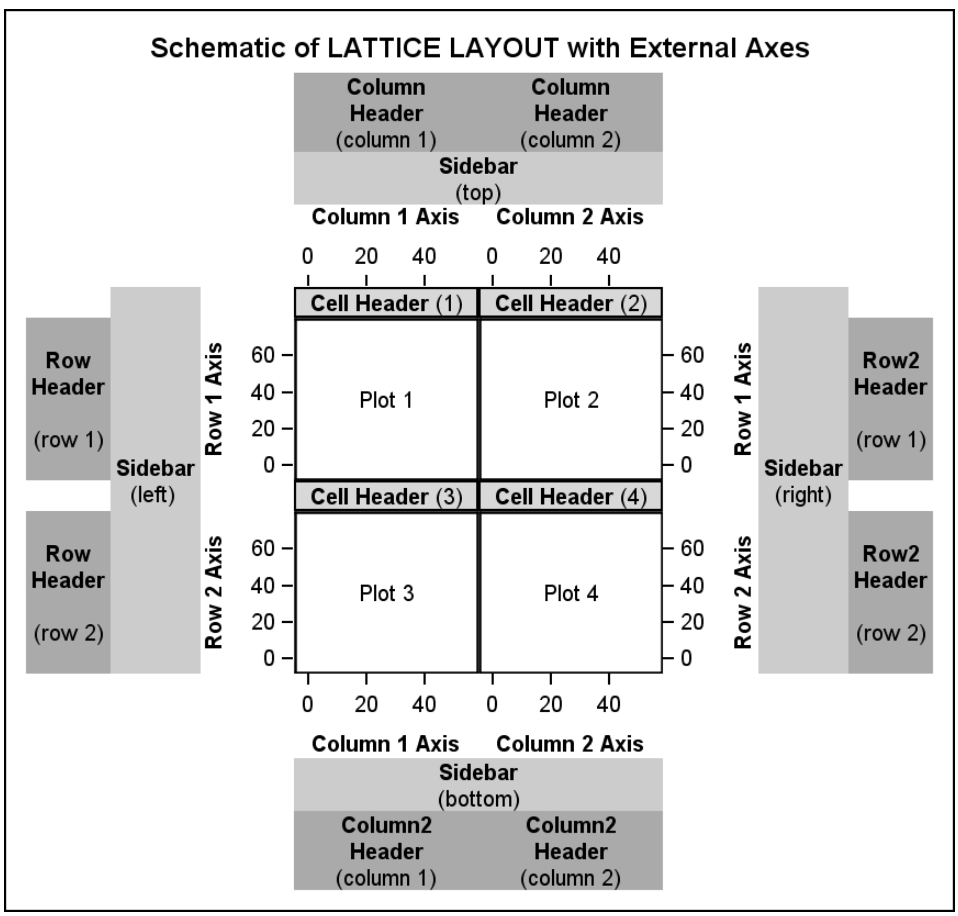
- Cell Headers are commonly used to describe the contents of a cell. Notice that the cell header, when present, has a separate space above the plot wall area. The cell header can contain more than one line of text, but it is not restricted to displaying text. For example, you could use this area to display a legend.
- Sidebars are often used to present text or a legend that pertains to all rows or all columns in the grid. Again, the sidebar is not limited to text or a legend. You could place another plot in a sidebar.
- Column Headers and Row Headers present text that pertains to individual columns and rows. These header areas can also be used to display other components, like legends and plots.

Figure 10.2 on page 157 shows how the lattice would look if you used additional options to externalize the axes. The figure externalizes both the row and column axes, but you could externalize the axes only for the rows, or only for the columns. When axes are external to the cells, the scale of the data ranges that are displayed for the plots are always unified in some form. Unifying the scale of the data ranges means taking the minimum of all data minima and the maximum of all data maxima from a set of plots. The following variations are available for unifying the axes:

- the scale of the data ranges of all X-axes in a column can be unified on a per-column basis, or unified across all columns (see "Column 1 Axis" and "Column 2 Axis" in Figure 10.2 on page 157).
- the scale of the data ranges of all Y-axes in a row can be unified on a per-row basis, or unified across all rows (see "Row 1 Axis" and "Row 2 Axis" in Figure 10.2 on page 157).

By default, external axes are displayed only on the primary axes (bottom and left). They are not displayed on the secondary axes (top and right) unless requested. Notice that external axes use less space and result in larger plot areas than internal axes. (Compare Figure 10.2 on page 157 with Figure 10.1 on page 156, which is the same size.)

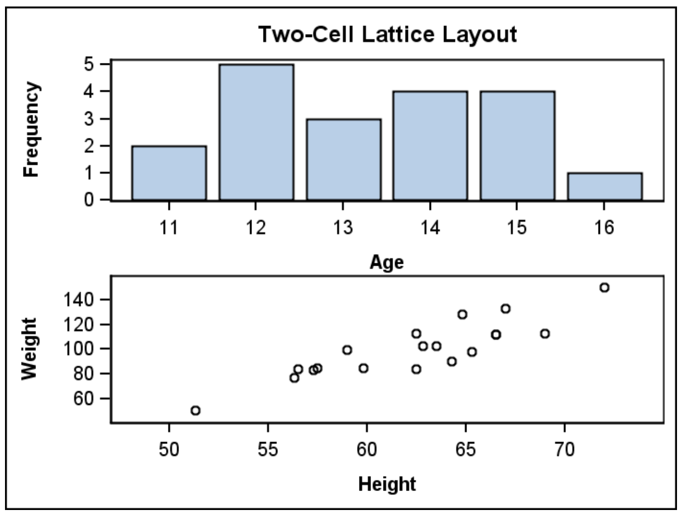
Figure 10.2 *LATTICE Layout with External Axes*



The following example shows a very simple LATTICE layout:

```
proc template;
  define statgraph intro;
    begingraph;
      entrytitle "Two-Cell Lattice Layout";
```

```
        layout lattice;
          barchart x=age;
          scatterplot x=height y=weight;
        endlayout;
      endgraph;
    end;
  run;

  proc sgrender data=sashelp.class template=intro;
  run;
```



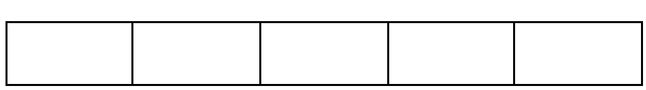
In a LATTICE layout, each plot statement is considered independent and is placed in a separate cell. When no grid size is provided, the default layout creates a graph with one column of cells, and it allots each cell the same amount of space. The number of rows in the grid is determined by the number of stand-alone plot statements in the layout block.

Defining a Basic Lattice

Setting Grid Dimensions

Assume you want a grid of five plots. Before starting to write code, you must first decide what grid dimensions you want to set (how many columns and rows) and whether you want to permit an empty cell in the grid. If you do not want an empty cell, you must limit the grid to five cells, which gives you two choices for the grid dimensions: five columns by one row (5x1), or one column by five rows (1x5).

To specify grid size, you use the ROWS= and COLUMNS= options in the LAYOUT LATTICE statement. These options can be used in three ways to create the following grid, which contains one row and five columns:



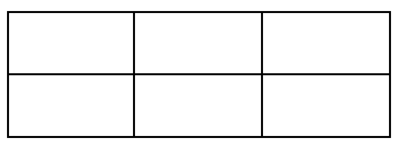
`layout lattice / columns=5 rows=1;`
`  /* plot definitions */`
`endlayout;`
	This makes the grid size explicit.

`layout lattice / order=columnmajor`
`                 rows=1;`
`  /* plot definitions */`
`endlayout;`
	To specify only one grid row, also specify ORDER=COLUMNMAJOR. In this case, there will be as many grid columns as there are plot definitions. This is the recommended way to create a row of plots.

`layout lattice / columns=5;`
`  /* plot defintions */`
`endlayout;`
	When only the number of columns is specified, you place a limit on how many plots can appear in one row. If you were to include more than five plot definitions, additional rows (with five columns) would be added automatically because ORDER=ROWMAJOR by default.

If you are willing to have an empty cell in the grid, you could use a 2x3 or a 3x2 grid:

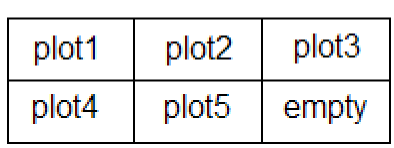
```
layout lattice / columns=3 rows=2 ;
endlayout;
```



Note: The LAYOUT LATTICE statement honors the full specification of columns and rows, unlike the LAYOUT GRIDDED statement, which honors only COLUMNS= or ROWS=, depending on the ORDER= setting.

By default, the layout uses the ORDER=ROWMAJOR setting to populate grid cells. This specification essentially means "fill in all cells in the top row (starting at the top left) and then continue to the next row below":

```
layout lattice / columns=3 rows=2 ;
  /* plot1 definition */
  /* plot2 definition */
  /* plot3 definition */
  /* plot4 definition */
  /* plot5 definition */
endlayout;
```



Alternatively, you can specify ORDER=COLUMNMAJOR, which means "fill in all cells in the left column and then continue to the next column to the right":

```
layout lattice / columns=3 rows=2 order=columnmajor ;
  /* plot1 definition */
  /* plot2 definition */
  /* plot3 definition */
  /* plot4 definition */
```

```
  /* plot5 definition */
endlayout;
```

plot1	plot3	plot5
plot2	plot4	empty

Setting Gutters

To conserve space, the default LATTICE layout does not include a gap between cell boundaries. In some cases, this might cause the cell contents to appear too congested. You can add a vertical gap between all cells with the COLUMNGUTTER= option, and you can add a horizontal gap between all rows with the ROWGUTTER= option. If no units are specified, pixels (PX) are assumed.

```
layout lattice / columns=3 rows=2 columngutter=5 rowgutter=5 ;
  /* plot1 definition */
  /* plot2 definition */
  /* plot3 definition */
  /* plot4 definition */
  /* plot5 definition */
endlayout;
```

plot1	plot2	plot3
plot4	plot5	empty

Note that by adding gutters, you do not increase the size of the graph. Instead, the cells shrink to accommodate the gutters. Depending on the number of cells in the grid and the size of the gutters, you will frequently want to adjust the size of the graph to obtain optimal results, especially if the cells contain complex graphs. For more information, see "Adjusting the Graph Size" on page 182.

Defining Cells

Several valid techniques are available for indicating the contents of a cell:

Technique	Example	Advantages	Disadvantages
stand-alone plot statement	`scatterplot x= y=;`	simplicity	can't adjust axes can't have overlays can't have cell headers

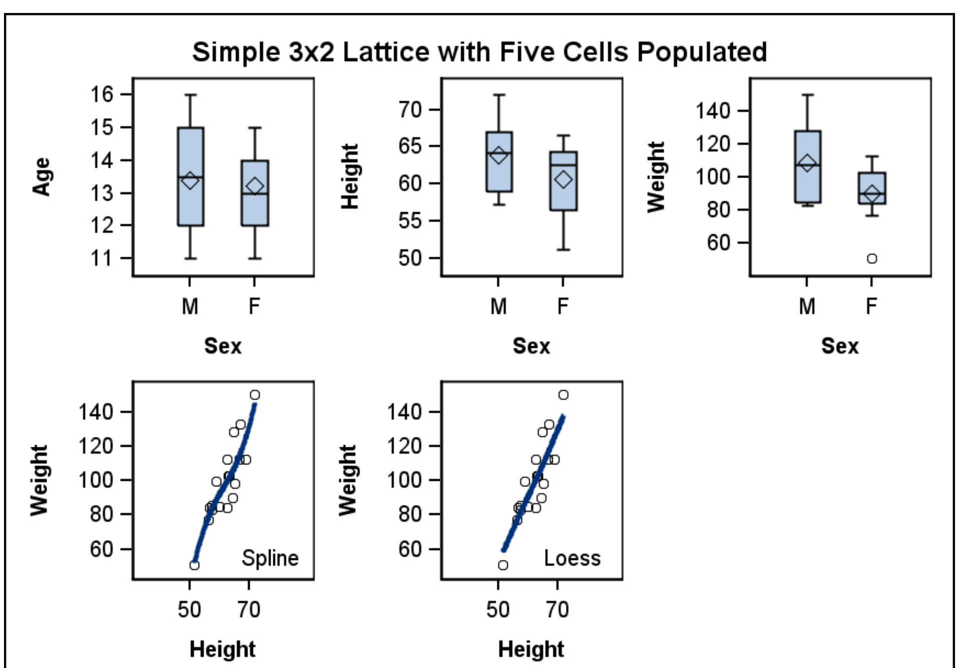
layout block	`layout overlay;`
	`scatterplot x= y=;`
	`seriesplot x= y=;`
	`endlayout;`
		cell can contain a complex plot
		axes can be adjusted
		other layout types can be used
			can't have cell headers

Technique	Example	Advantages	Disadvantages

cell block	`cell;`
	` layout overlay;`
	`  scatterplot x= y=;`
	`  seriesplot x= y=;`
	` endlayout;`
	`endcell;`
		makes it easy to see cell boundary in code
		
		required if a cell header is desired
			adds to program length when no cell header is desired

The following code fragment for a LATTICE layout shows a simple example:

```
entrytitle "Simple 3x2 Lattice with Five Cells Populated";
layout lattice / columns=3 rows=2 columngutter=10 rowgutter=10;

  /* stand-alone plot statements define cells 1-3 */
  boxplot x=sex y=age;
  boxplot x=sex y=height;
  boxplot x=sex y=weight;

  /* overlay blocks define cells 4-5 */
  layout overlay;
    scatterplot y=weight x=height;
    pbsplineplot y=weight x=height;
    entry halign=right "Spline" / valign=bottom;
  endlayout;

  layout overlay;
    scatterplot y=weight x=height;
    loessplot y=weight x=height;
    entry halign=right "Loess " / valign=bottom;
  endlayout;

endlayout;
```

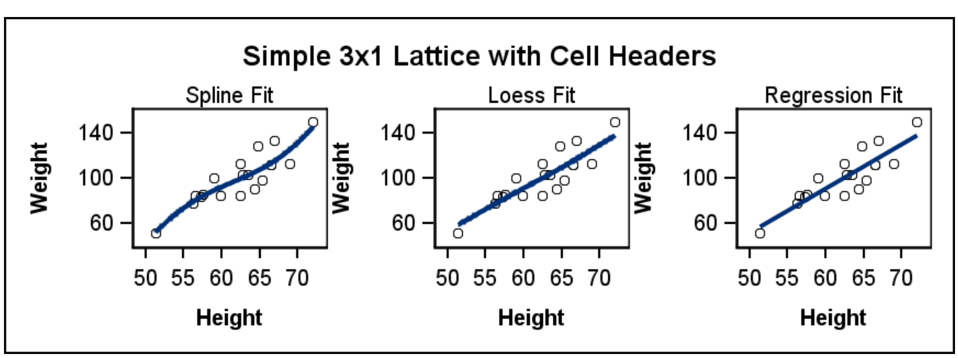


In the examples shown to this point, a LATTICE layout produces the same result as a GRIDDED layout. We can now look at features that are not available with the GRIDDED layout.

Adding Cell Headers

To add cell headers to the grid, you must specify a CELL block that contains a nested CELLHEADER block. The CELLHEADER block can contain one or more ENTRY statements, or it can contain other statements (DISCRETELEGEND, for example).

```
entrytitle "Simple 3x1 Lattice with Cell Headers";
layout lattice / columns=3 rows=1;

  /* cell blocks cells 1-3 */
    cell;
      cellheader;
        entry "Spline Fit";
      endcellheader;
      layout overlay;
        scatterplot y=weight x=height;
        pbsplineplot y=weight x=height;
      endlayout;
    endcell;

    cell;
      cellheader;
        entry "Loess Fit";
      endcellheader;
      layout overlay;
        scatterplot y=weight x=height;
        loessplot y=weight x=height;
      endlayout;
    endcell;

    cell;
      cellheader;
        entry "Regression Fit";
      endcellheader;
      layout overlay;
        scatterplot y=weight x=height;
        regressionplot y=weight x=height;
      endlayout;
    endcell;
endlayout;
```



You can enhance any cell header in the following way:

- nest a GRIDDED layout in the CELLHEADER block
- set BORDER=TRUE on the LAYOUT GRIDDED statement
- add the ENTRY statement(s) to the GRIDDED layout.

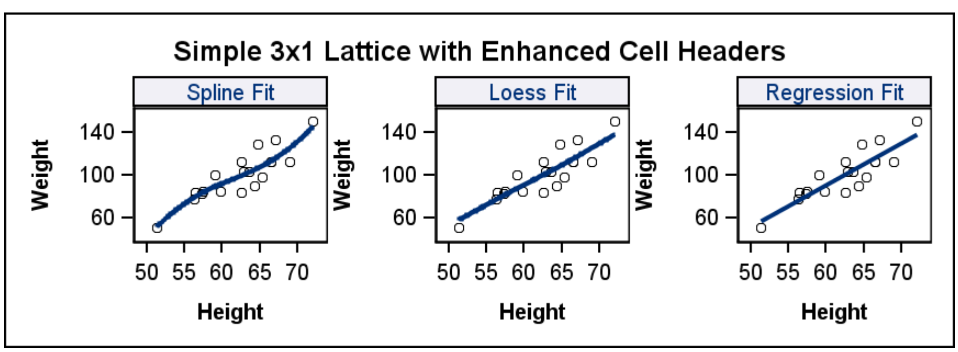
Because the GRIDDED layout fills the cell header space above the plot wall, its border aligns nicely with the plot.

You can further enhance the cell header by making the GRIDDED layout's background opaque and setting a background color for it. To ensure that the color remains coordinated with the current style, you could choose any of several style elements that define light background colors, such as GraphHeaderBackground, GraphBlock, or GraphAltBlock. Note that several style definitions set the GraphHeaderBackground color to be the same as the GraphBackground color. For styles like LISTING and JOURNAL, the background is white.

As a final enhancement, you could coordinate the text color for the cell headers with a visual attribute in the plot. For example, if you are displaying a fit plot in the cell, you could set the text color to match the color of the fit line. The TEXTATTRS= option in the ENTRY statement can be used to set the text properties. The default settings for TEXTATTRS= are derived from the GraphValueText style element. For more information on ENTRY statements, see Chapter 7, “Adding and Changing Text in a Graph,” on page 101.

The following code enhances the cell header block of the first cell. Similar code would be used to enhance the header blocks of the other two cells:

```
cellheader;
  layout gridded / border=true opaque=true
                   backgroundcolor=GraphAltBlock:color;
    entry "Spline Fit" / textattrs=(color=GraphFit:contrastColor);
  endlayout;
endcellheader;
```



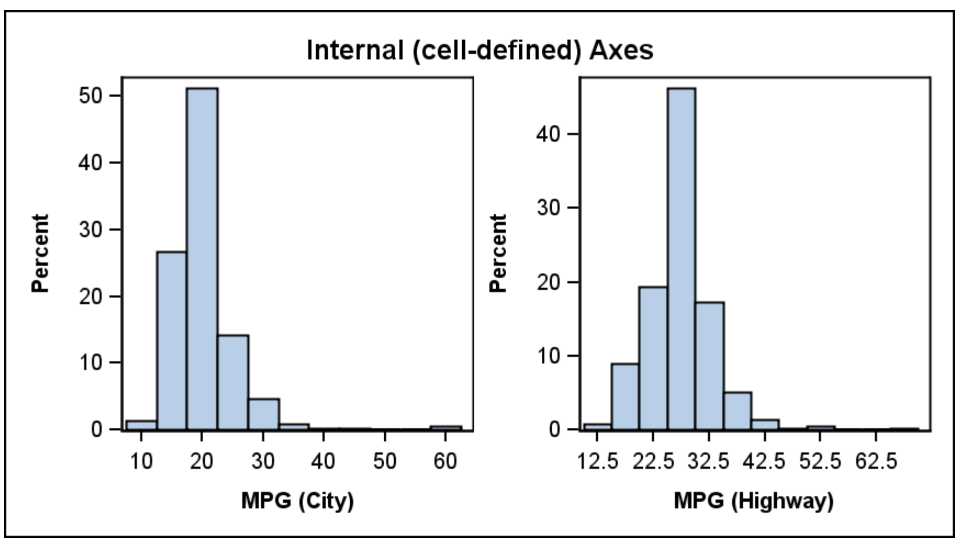
If you have a lengthy text description to add to a cell header, you should use multiple ENTRY statements to break the text into small segments; otherwise, the text might be truncated. Also, for a given row, if the number of lines of text in the cell headers varies, a uniform cell height is maintained across the row by setting all the row headers to the height needed by the largest cell header.

Creating Uniform Axes Across Rows or Columns

Internal Axes

By default, the plots in the cells of the LATTICE layout manage their own axes, as demonstrated by the following example:

```
proc template;
 define statgraph internalaxes;
  begingraph;
   entrytitle "Internal (cell-defined) Axes";
   layout lattice / columns=2 columngutter=5px;
     histogram mpg_city;
     histogram mpg_highway;
   endlayout;
 endgraph;
end;
run;

proc sgrender data=sashelp.cars template=internalaxes;
run;
```



In this example, notice that the X and Y axes have different data ranges for each cell. In cases where you want to facilitate comparisons of the cell contents, you can set uniform axis scales across the rows in the grid, or across the columns, or across both.

Uniform Axis Ranges

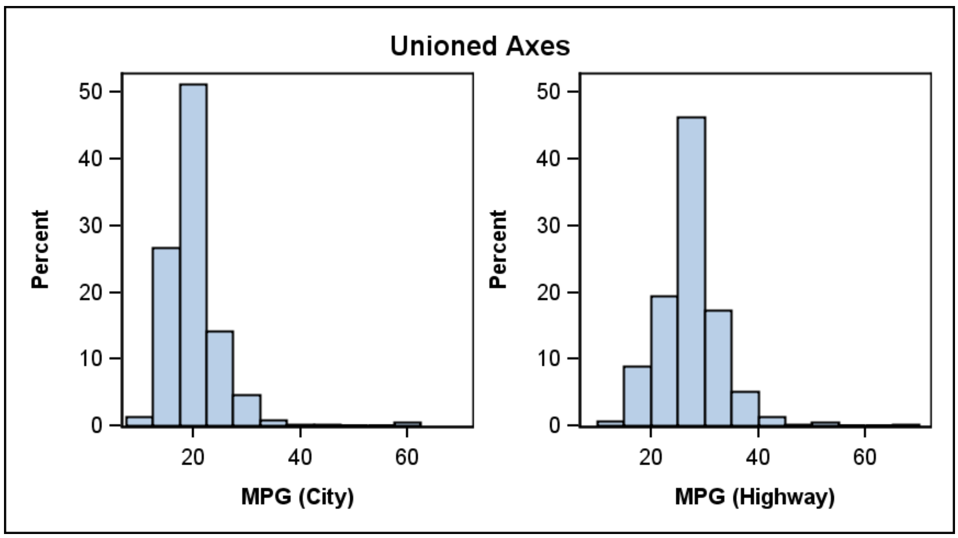
To set a uniform scale on the X axes in each row of a lattice, use the COLUMNDATARANGE= option on the LAYOUT LATTICE statement. Likewise, to set a uniform scale on the Y axes in each row of the lattice, use the ROWDATARANGE= option. Both options accept one of the following values:

DATA
: scales the axes independently for each cell. This is the default.

UNION
: finds the minimum of the data minima and the maximum of the data maxima, on a per-row or per-column basis, and sets this range on the appropriate axis for each cell in a row or column.

UNIONALL
: finds the minimum of the data minima and the maximum of the data maxima over all rows or all columns, and sets this range on the appropriate axis for each cell.

```
layout lattice / columns=2 columngutter=5px
                 columdatarange=unionall
                 rowdatarange=union ;
  histogram mpg_city;
  histogram mpg_highway;
endlayout;
```



Note: The default X-axis for a histogram shows ticks at bin midpoints or bin start/end points. If the histograms happen to have the same bin width, it is possible to create uniformly scaled X axes. However, when the bin widths are different, there might not be any common midpoints. To handle this situation, the LATTICE layout automatically switches to a LINEAR type axis so that the axis tick values can be uniform, even though they might not be at bin midpoints or boundaries for all histograms.

Some restrictions apply to the UNION and UNIONALL settings on any row or column of the lattice:

- all plots must have the same axis type: LINEAR, LOG, TIME, or DISCRETE
- overlaid plots should not specify both a Y and Y2 axis, or both an X and X2 axis.
- if a cell contains a LAYOUT OVERLAY3D or LAYOUT OVERLAYEQUATED statement, the uniform axis ranges and external axes are not supported for that row or column.

External Axes

Specifying External Axes

Whenever axis scales have been unified for a row or a column, you can replace the individual cell axes in that row or column with a single axis that is external to the cells.

To externalize X axes, use the following syntax:

COLUMNAXES;
  COLUMNAXIS / *options* ;
  <COLUMNAXIS / *options* ;>
ENDCOLUMNAXES;

To externalize Y axes, use the following syntax:

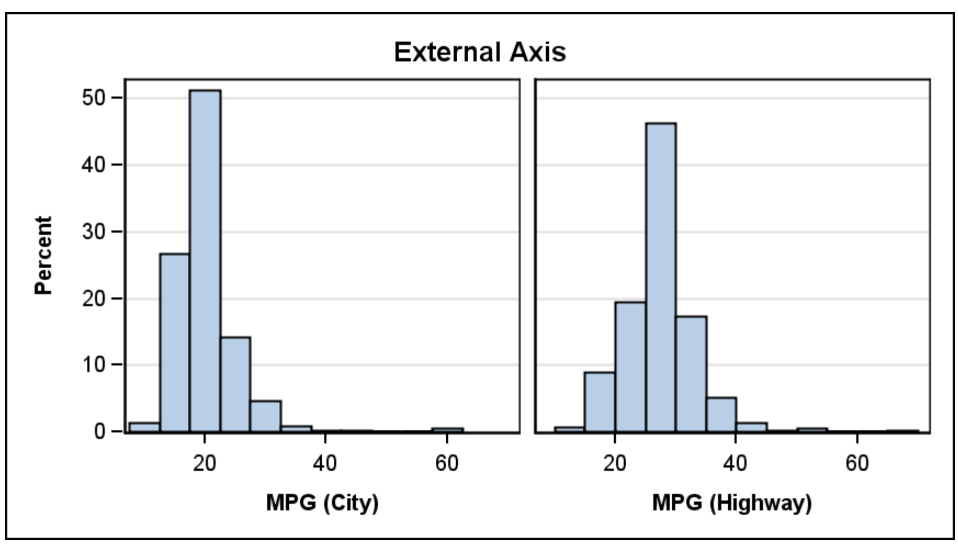
ROWAXES;
  ROWAXIS / *options* ;
  <ROWAXIS / *options* ;>
ENDROWAXES;

Within the axes blocks, you should specify as many COLUMNAXIS or ROWAXIS statements as there are columns or rows in the grid. The *options* that are available to each statement are similar to those that are available for the XAXISOPTS= and YAXISOPTS= options of a LAYOUT OVERLAY statement. The *options* that you specify can differ from statement to statement.

Note: When a row or column external axis is used, all axis options on the internal axes in that same dimension will be ignored.

The following code fragment externalizes the Y axes:

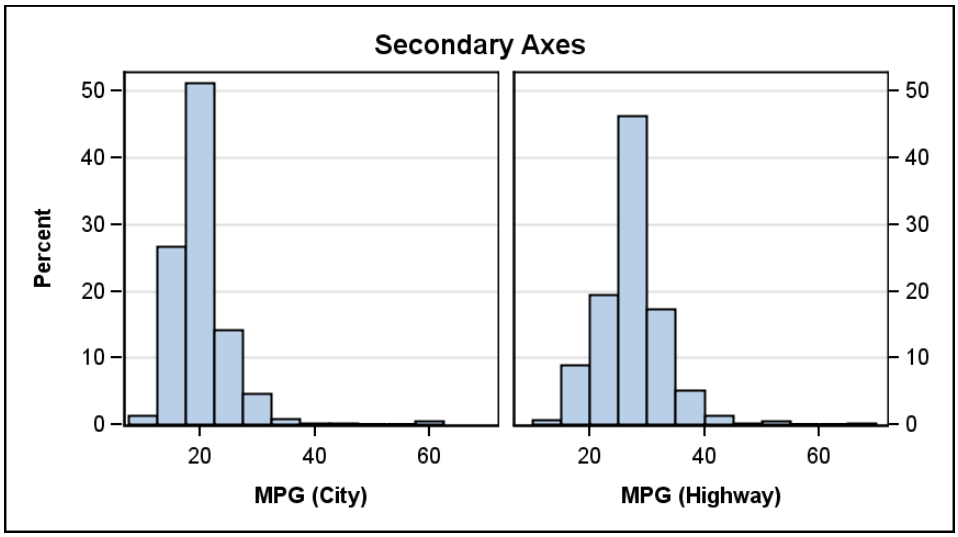
```
layout lattice / columns=2 columngutter=5px
                 columndatarange=unionall
                 rowdatarange=union;
  histogram mpg_city;
  histogram mpg_highway;
  rowaxes;
    rowaxis / griddisplay=on;
  endrowaxes;
endlayout;
```



Displaying External Secondary Axes

The DISPLAYSECONDARY= option can be used on a ROWAXIS statement to display a row axis at the right of the lattice. It can be used on a COLUMNAXIS statement to display a column axis at the top of the lattice. An external secondary axis is a duplicate of the external primary axis, not a truly independent axis. However, you can change the features that are displayed on the secondary axis. In the following example, the ticks and tick values are repeated on the right side of the lattice, but the axis label is suppressed by not listing it among the features that are requested on the DISPLAYSECONDARY= option.

```
layout lattice / columns=2 columngutter=5px
                 columndatarange=unionall
                 rowdatarange=union;
  histogram mpg_city;
  histogram mpg_highway;
  rowaxes;
    rowaxis / griddisplay=on displaysecondary=(ticks tickvalues);
  endrowaxes;
endlayout;
```

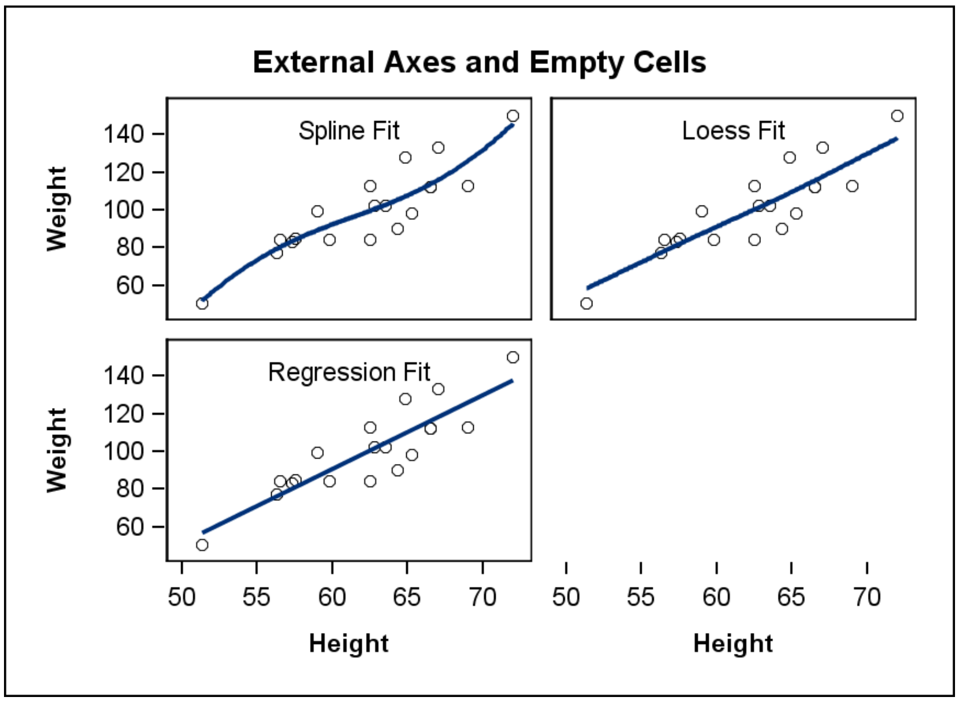


External Axes and Empty Cells

If a LATTICE layout generates empty cells and there are external axes, a row or column axis might be displayed near one or more of those empty cells. The following example shows the default case:

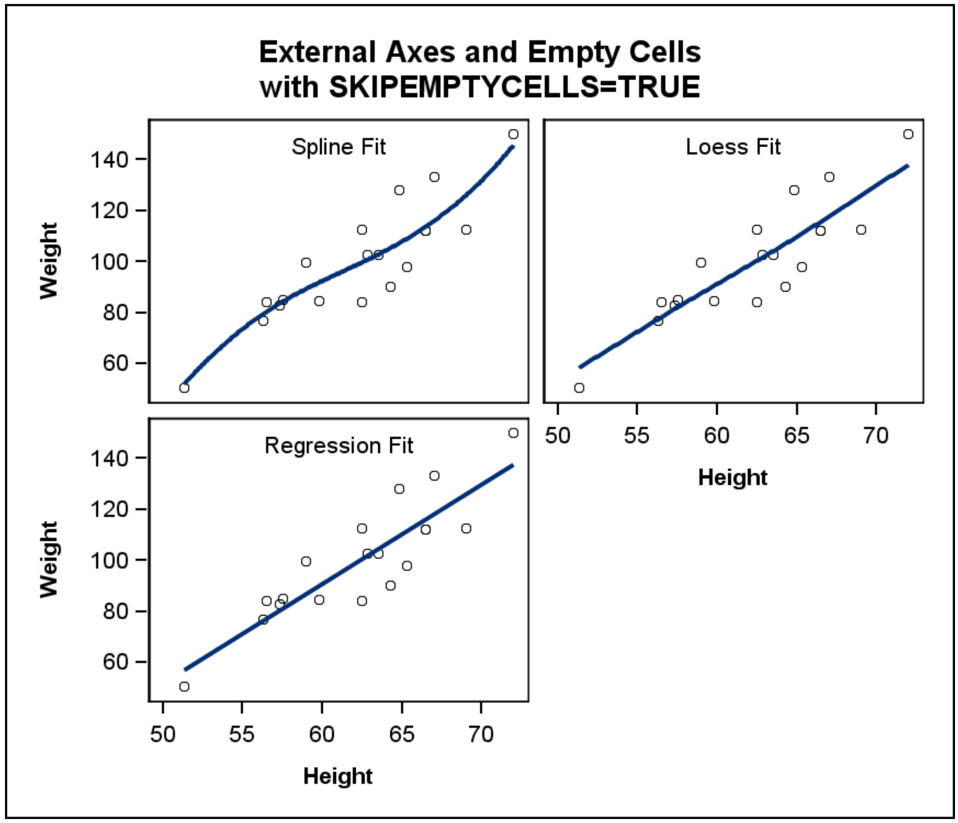
```
layout lattice / columns=2 rows=2
  rowgutter=5px columngutter=5px
  rowdatarange=unionall columndatarange=unionall;

  /* overlay blocks define cells 1-3 */
  layout overlay;
    entry "Spline Fit" / valign=top;
    scatterplot y=weight x=height;
    pbsplineplot y=weight x=height;
  endlayout;
  layout overlay;
    entry "Loess Fit" / valign=top;
    scatterplot y=weight x=height;
    loessplot y=weight x=height;
  endlayout;
  layout overlay;
    entry "Regression Fit" / valign=top;
    scatterplot y=weight x=height;
    regressionplot y=weight x=height;
  endlayout;
```

```
    rowaxes;
      rowaxis;
      rowaxis;
    endrowaxes;
    columnaxes;
      columnaxis;
      columnaxis;
    endcolumnaxes;
  endlayout;
```



Adding the SKIPEMPTYCELLS=TRUE setting to the LAYOUT LATTICE statement eliminates the space that is normally reserved for the empty cells. In that case, an external axis that might have been displayed near an empty cell will be displayed near a populated cell instead:

```
layout lattice / columns=2 rows=2
  rowgutter=5px columngutter=5px
  rowdatarange=unionall columndatarange=unionall
  skipemptycells=true ;
```



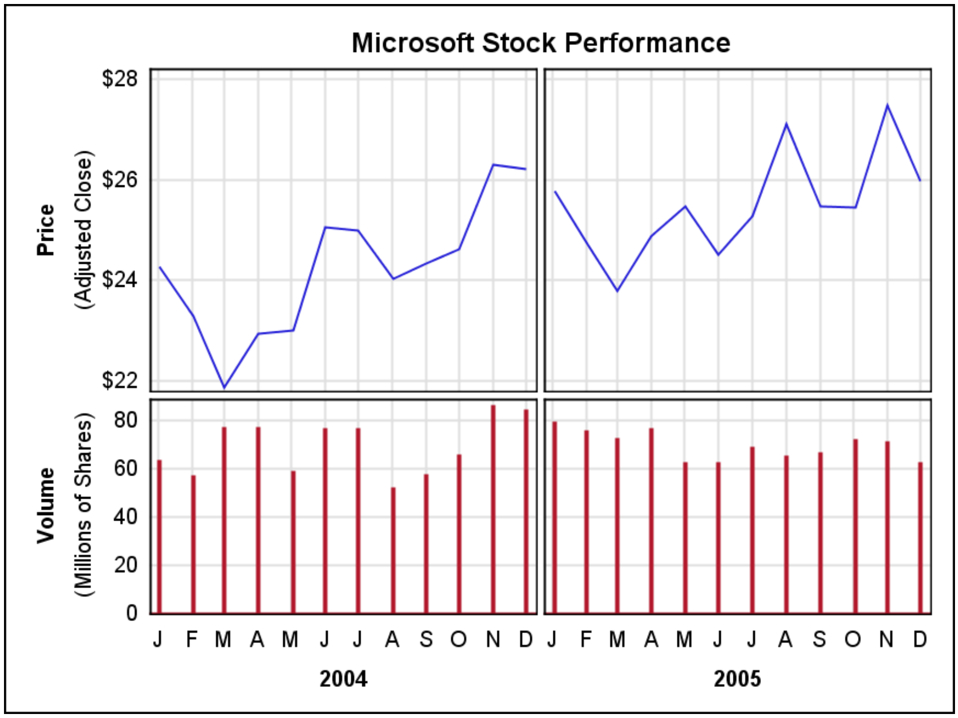
Defining a Lattice with Additional Features

Overview

The following sections explain how to generate Figure 10.3 on page 170, which requires the following tasks:

- transforming the input data
- using external axes instead of internal cell axes
- adding sidebars that display descriptive text
- using column headers
- sizing rows.

Figure 10.3 *Stock Plot*



Transforming the Input Data

A common use for a lattice is to create a graph that shows different subsets of the same input data. In some cases, those subsets are already defined in the input data. However, you will frequently have to transform the input data to make it suitable for the graph you are trying to create. This might require any or all of the following:

- summarizing the data
- transposing the data
- scaling the data values
- creating new variables that represent subsets of the data.

The graph that is shown in Figure 10.3 on page 170 is based on data from SASHELP.STOCKS, which contains several years of monthly stock information for three companies. The data set contains columns for STOCK, DATE , VOLUME, and ADJCLOSE (Adjusted Closing Price). However, it does not have the volume and price information in the form that is needed for the graph. The LATTICE layout does not support subsets of the input data on a per-cell basis. So, in order to make the cell content different, unique variables must be created for each cell to provide the appropriate date, volume, and price information. The following DATA step performs the necessary input data transformations:

```
data stock;
  set sashelp.stocks;
  where stock eq "Microsoft" and year(date) in (2004 2005);
  format Date2004 Date2005 date.
         Price2004 Price2005 dollar6.;
  label Date2004="2004" Date2005="2005";
  if year(date) = 2004 then do;
```

```
      Date2004=date;
      Vol2004=volume*10**-6;
      Price2004=adjclose;
   end;
   else if year(date)=2005 then do;
      Date2005=date;
      Vol2005=volume*10**-6;
      Price2005=adjclose;
   end;
   keep Date2004 Date2005 Vol2004
        Vol2005 Price2004 Price2005;
run;
```

The data is filtered for Microsoft and for the years 2004 and 2005. Next, new variables are created for each year and the Volume and Stock Price within each year. Because the volumes are large, they are scaled to millions. This scaling will be noted in the graph. This coding results in a "sparse" data set, but it is the correct organization for the lattice because observations with missing X or Y values are not plotted.

Obs	Date2004	Date2005	Price2004	Price2005	Vol2004	Vol2005
1	.	01DEC05	.	$26	.	62.8924
2	.	01NOV05	.	$27	.	71.4692
3	.	03OCT05	.	$25	.	72.1325
4	.	01SEP05	.	$25	.	66.9765
5	.	01AUG05	.	$27	.	65.5300
6	.	01JUL05	.	$25	.	69.0466
7	.	01JUN05	.	$25	.	62.9567
8	.	02MAY05	.	$25	.	62.6998
9	.	01APR05	.	$25	.	77.0902
10	.	01MAR05	.	$24	.	72.8997
11	.	01FEB05	.	$25	.	75.9923
12	.	03JAN05	.	$26	.	79.6428
13	01DEC04	.	$26	.	84.4881	.
14	01NOV04	.	$26	.	86.4461	.
15	01OCT04	.	$25	.	65.7429	.
16	01SEP04	.	$24	.	57.7253	.
17	02AUG04	.	$24	.	52.1046	.
18	01JUL04	.	$25	.	76.6667	.
19	01JUN04	.	$25	.	77.0683	.
20	03MAY04	.	$23	.	58.9425	.
21	01APR04	.	$23	.	77.3867	.
22	01MAR04	.	$22	.	77.1119	.
23	02FEB04	.	$23	.	57.3859	.
24	02JAN04	.	$24	.	63.6359	.

The key point to be aware of is that every plot in every cell must use variables that contain just the information appropriate for that cell. You cannot use WHERE clauses within the template definition to form subsets of the data.

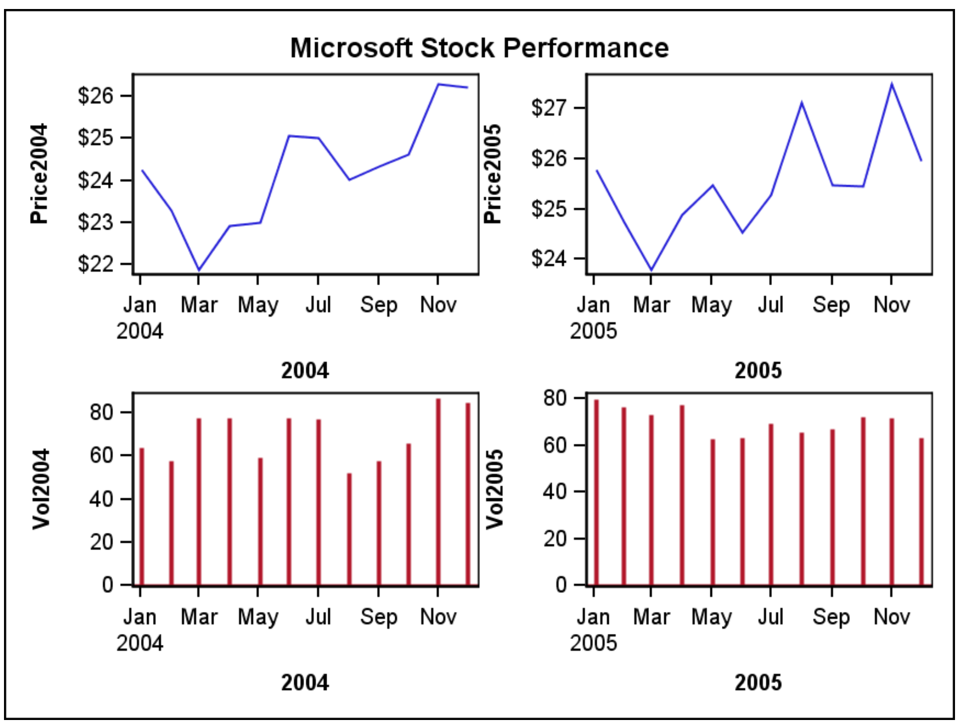
The following initial template defines the lattice:

```
proc template;
define statgraph lattice1;
 begingraph;
   entrytitle "Microsoft Stock Performance";
   layout lattice / columns=2 rows=2;
     /* define row 1 */
```

```
        seriesplot y=price2004 x=date2004 / lineattrs=GraphData1;
        seriesplot y=price2005 x=date2005 / lineattrs=GraphData1;

        /* define row 2 */
        needleplot  y=vol2004 x=date2004 /
          lineattrs=GraphData2(thickness=2px pattern=solid);
        needleplot  y=vol2005 x=date2005 /
          lineattrs= GraphData2(thickness=2px pattern=solid);

      endlayout;
    endgraph;
  end;
  run;

  proc sgrender data=stock template=lattice1;
  run;
```

Note that because Date2004 and Date2005 have an associated SAS date format that a TIME axis is used and the variable labels are used for X-axis labels.

Figure 10.4 *Initial Lattice for the Graph*

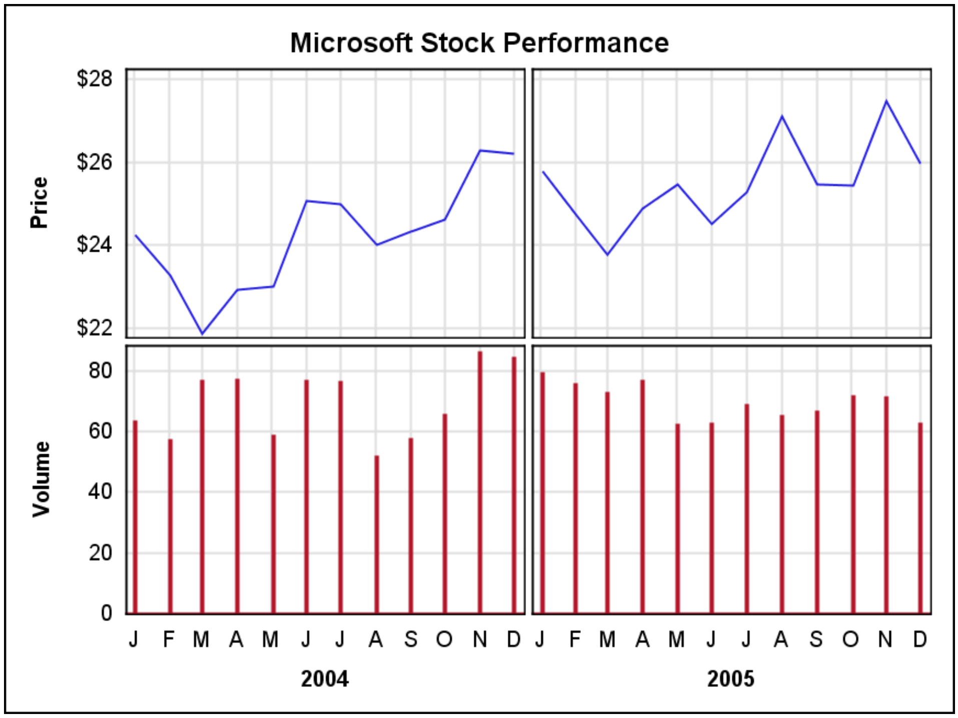


Using External Axes

Figure 10.4 on page 172 would benefit from externalizing the X and Y axes because the external axes will reduce the redundant X axis information and unify the data ranges in the Y axes. We would also like to add grid lines to all axes. To conserve space along the X axes, the automatic formatting of each TIME axis is turned off in the following template code. The TICKVALUEFORMAT=MONNAME1. setting indicates how to format the time axis tick values.

```
proc template;
define statgraph lattice2;
 begingraph / designwidth=495px designheight=370px;
   entrytitle "Microsoft Stock Performance";
   layout lattice / columns=2 rows=2
                    rowdatarange=union columndatarange=union
                    rowgutter=3px columngutter=3px ;
     /* define row 1 */
     seriesplot x=date2004  y=price2004 / lineattrs=GraphData1;
     seriesplot x=date2005  y=price2005 / lineattrs=GraphData1;

     /* define row 2 */
     needleplot  x=date2004  y=vol2004 /
         lineattrs=GraphData2(thickness=2px pattern=solid);

     needleplot   x=date2005 y=vol2005 /
         lineattrs= GraphData2(thickness=2px pattern=solid);

     rowaxes;
       rowaxis / griddisplay=on display=(label tickvalues)
                 label="Price"  labelattrs=(weight=bold);
       rowaxis / griddisplay=on display=(label tickvalues)
                 label="Volume" labelattrs=(weight=bold);
     endrowaxes;

     columnaxes;
       columnaxis / griddisplay=on display=(label tickvalues)
                    labelattrs=(weight=bold)
                    timeopts=(tickvalueformat=monname1.);
       columnaxis / griddisplay=on display=(label tickvalues)
                    labelattrs=(weight=bold)
                    timeopts=(tickvalueformat=monname1.);
     endcolumnaxes;
   endlayout;
 endgraph;
end;
run;

proc sgrender data=stock template=lattice2;
run;
```

Figure 10.5 *Lattice with External Axes*



Using Cell Axes

In most cases externalizing axes improves graph appearance and streamlines coding. However, if there are some axis options that do not apply uniformly to all axes in a column or row, you need to use the standard axis options on a cell basis instead of external axes.

For example, if you wanted X-axis grid lines to appear on the top row of plots but not on the second row of plots, you could not use external axes. Instead, you would enclose the cell contents in an overlay-type layout block and add XAXISOPTS= options on the layout statements. as shown in the following layout blocks:

```
/* overlay blocks define X-axis options for row 1 */
layout overlay / xaxisopts=(display=none  griddisplay=on);
  seriesplot x=date2004  y=price2004 / lineattrs=GraphData1;
endlayout;

layout overlay /  xaxisopts=(display=none griddisplay=on);
  seriesplot x=date2005  y=price2005 / lineattrs=GraphData1;
endlayout;

/* overlay blocks define X-axis options for row 2 */
layout overlay / xaxisopts=(display=(label tickvalues)
                           timeopts=(tickvalueformat=monname1.));
  needleplot  x=date2004  y=vol2004 /
    lineattrs=GraphData2(thickness=2px pattern=solid);
endlayout;

layout overlay / xaxisopts=(display=(label tickvalues)
                 timeopts=(tickvalueformat=monname1.));
  needleplot   x=date2005 y=vol2005 /
```

```
    lineattrs= GraphData2(thickness=2px pattern=solid);
endlayout;
```

Adding Sidebars

The graph in Figure 10.5 on page 174 is progressing well, but the ENTRYTITLE is centered on the entire graph. It would look better if it were centered on the grid area. This can be accomplished by removing the ENTRYTITLE statement and replacing it with a SIDEBAR block. Four sidebar areas are available: two that span all columns (one on the TOP and one on the BOTTOM), and two that span all rows (one on the RIGHT and one on the LEFT).

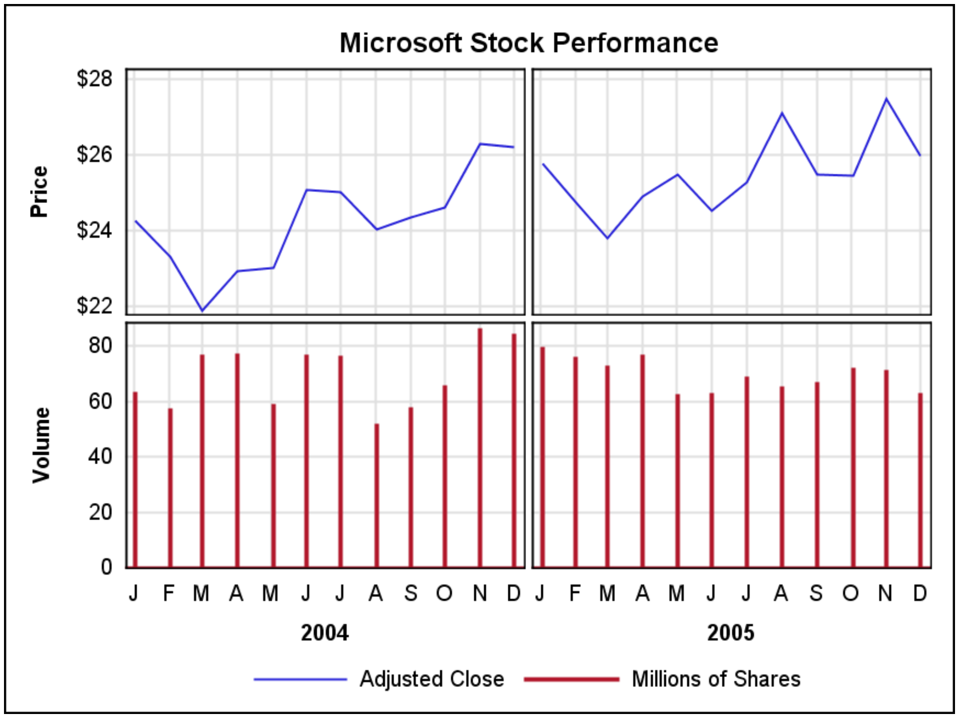
```
sidebar / align=top;
  entry "Microsoft Stock Performance" /
    textattrs=GraphTitleText pad=(bottom=5px);
endsidebar;
```

Finally, we need a way of explaining that the prices in the first row represent an adjusted close value. We also need to explain that the axis scaling for the second row is in millions of shares. Two strategies are available for providing this information.

The first strategy is to create an external legend. For this strategy, we must define legend text on two of the plot statements, and add a DISCRETELEGEND statement to the BOTTOM sidebar.

```
seriesplot x=date2004  y=price2004 /
  lineattrs=GraphData2(thickness=2px pattern=solid)
  name="series" legendlabel="Adjusted Close";

needleplot  x=date2004  y=vol2004 /
  lineattrs=GraphData2(thickness=2px pattern=solid)
  name="needle" legendlabel="Millions of Shares";

sidebar / align=bottom;
  discretelegend "series" "needle" / border=off pad=(top=10px);
endsidebar;
```

The following graph shows what this modification looks like:



The other strategy is to add to the row information. At first glance it would seem that you could do this very simply by extending the axis label text:

```
rowaxes;
  rowaxis / griddisplay=on display=(tickvalues)
            label="Volume (Millions of Shares)" ;
  rowaxis / griddisplay=on display=(tickvalues)
            label="Price (Adjusted Close)" ;
endrowaxes;
```

The problem here is that the extra axis label text might not fit; depending on the text size and the graph size, the text might be truncated. The axis option SHORTLABEL="*string*" is available to handle truncation, but we want more text, not alternate text, and there is no way to wrap the axis label to two lines. The solution is use row headers instead of specifying axis labels.

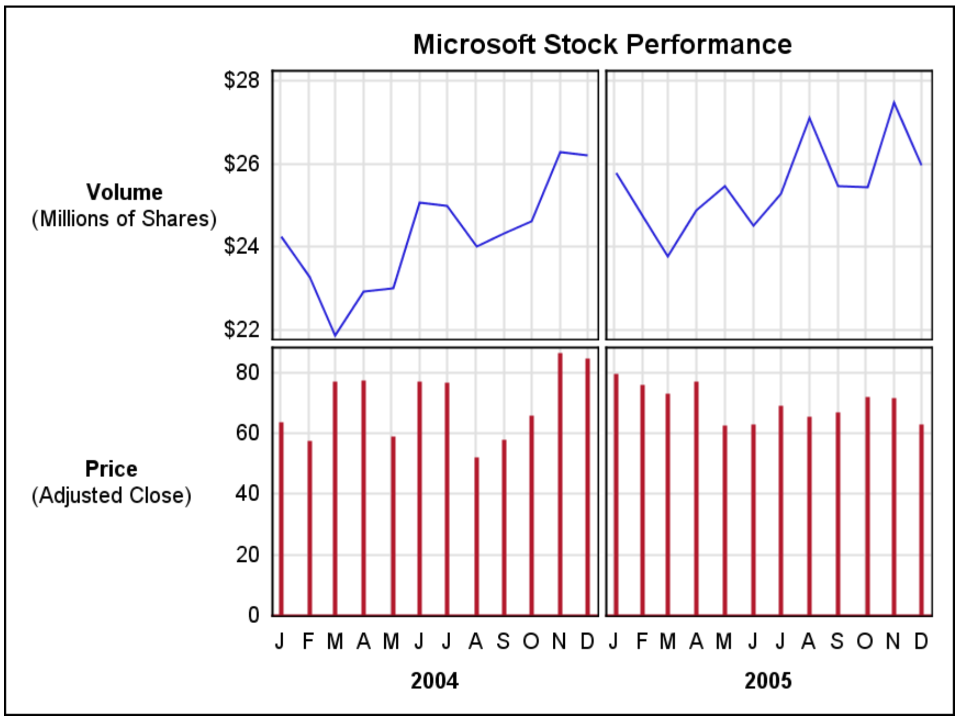
Using Column or Row Headers

For the graph that is shown in Figure 10.5 on page 174, we want to explain that the axis scaling in the first row is in millions of shares, and that the prices in the second row represent an adjusted close value. The strategy that we used in "Adding Sidebars" on page 175 was to create an external legend that displays that information. Another strategy we can use is to remove the label information from the row axes and introduce a ROWHEADERS block, as shown in the following code:

```
rowaxes;
  rowaxis / griddisplay=on display=(tickvalues);
  rowaxis / griddisplay=on display=(tickvalues);
endrowaxes;

rowheaders;
  layout gridded / columns=1;
```

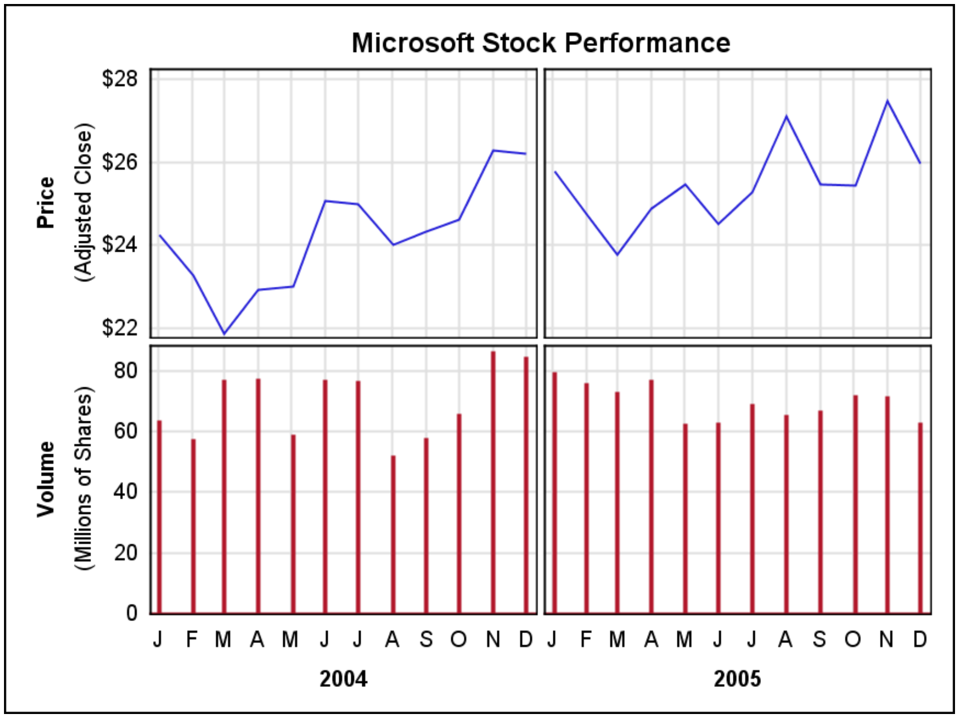
```
    entry "Volume" / textattrs=GraphLabelText;
    entry "(Millions of Shares)" / textattrs=GraphValueText;
  endlayout;
  layout gridded / columns=1;
    entry "Price" / textattrs=GraphLabelText;
    entry "(Adjusted Close)" / textattrs=GraphValueText;
  endlayout;
endrowheaders;
```

By nesting the ENTRY statements in the GRIDDED layouts, we can have multiple lines of text split exactly where we want and in any text style we desire. Without the GRIDDED layouts, only one ENTRY statement could be used per row.



To allow more space for the plots, we can rotate the row header text to make it appear to be a row axis label. Notice that we must specify COLUMNS=2 for the GRIDDED layouts.

```
rowaxes;
  rowaxis / griddisplay=on display=(tickvalues);
  rowaxis / griddisplay=on display=(tickvalues);
endrowaxes;

rowheaders;
  layout gridded / columns=2 ;
    entry "Price" / textattrs=GraphLabelText rotate=90 ;
    entry "(Adjusted Close)" / textattrs=GraphValueText rotate=90 ;
  endlayout;
  layout gridded / columns=2 ;
    entry "Volume" / textattrs=GraphLabelText rotate=90 ;
    entry "(Millions of Shares)" / textattrs=GraphValueText rotate=90 ;
  endlayout;
endrowheaders;
```



The clean look of the graph is achieved by removing redundant cell axis information and moving it to external column and row locations. In this example, the use of row headers provided the desired flexibility over row axis labels.

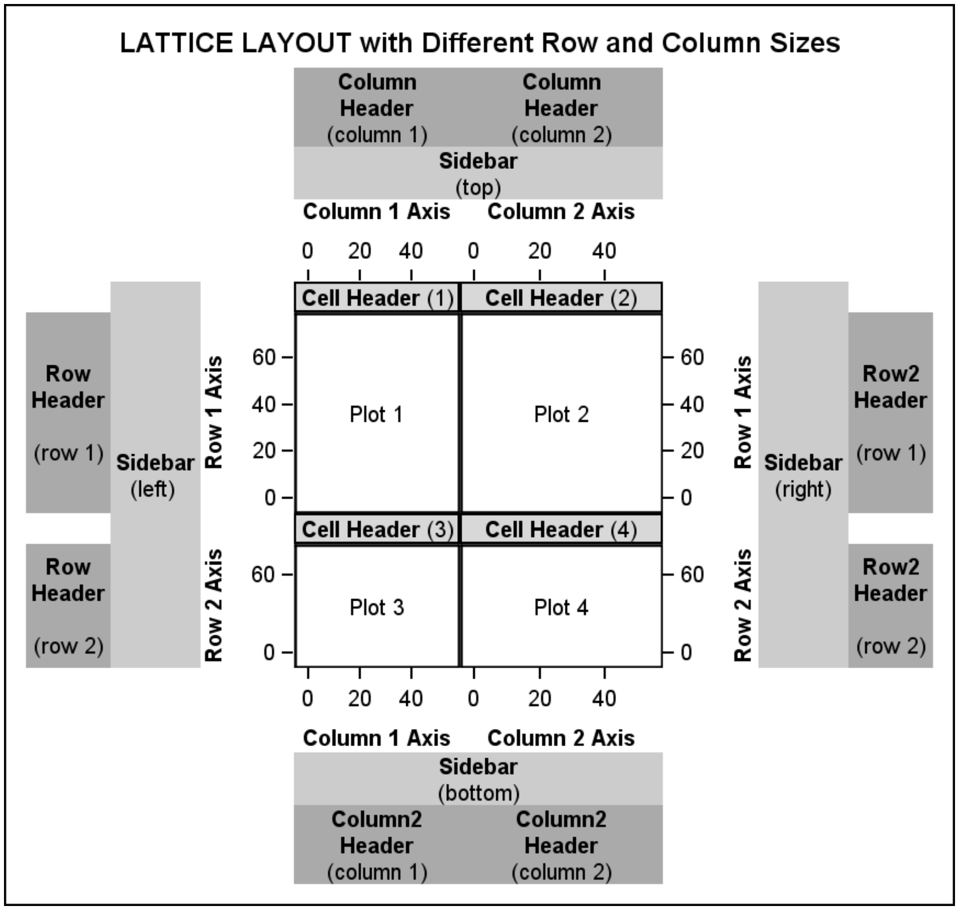
Adjusting the Sizes of Rows and Columns

By default, the rows and columns of the lattice are of the same depth and width. You can use the ROWWEIGHTS= and COLUMNWEIGHTS= options on the LAYOUT LATTICE statement to designate different row depths and/or column widths. Consider the following settings:

```
LAYOUT LATTICE /  ROW=2 COLUMNS=2
  ROWWEIGHTS=(.6 .4) COLUMNWEIGHTS=(.45 .65) ;
```

Figure 10.6 on page 179 uses these settings. The ROWWEIGHTS= setting specifies that the first row gets 60% of available row space, and the second row gets 40%. The COLUMNWEIGHTS= setting specifies that the first column gets 45% of available column space, and the second column gets 65%. Potentially, the settings on these options affect the space that is allocated to cell headers and to row and column headers.

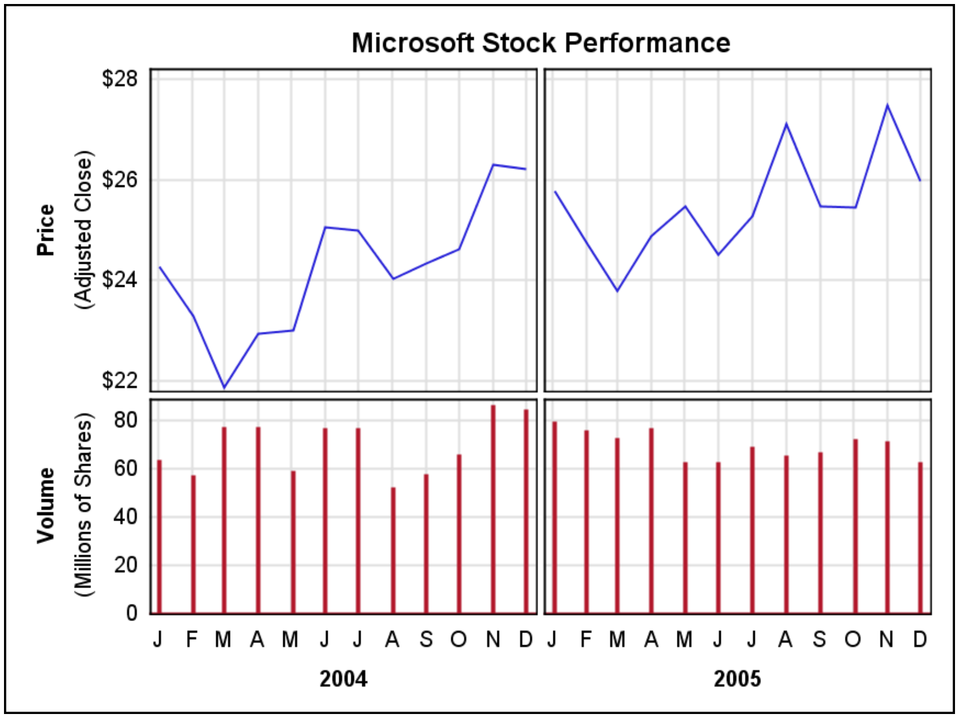
Figure 10.6 *LAYOUT LATTICE with different Row and Column Sizes*



In a traditional stock plot, the area devoted to price information is larger than the area devoted to the volume information. Here is the adjustment made to the row depths:

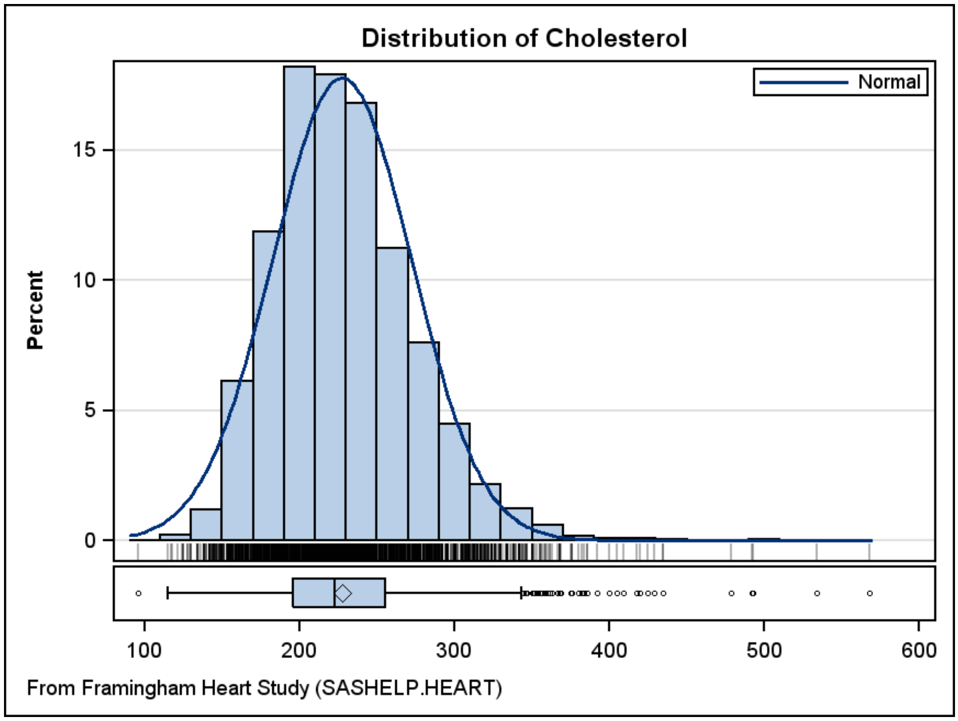
```
layout lattice / columns=2 rows=2 rowweights=(.6 .4)
                 rowdatarange=union columndatarange=union
                 rowgutter=3px columngutter=3px;
```

Figure 10.7 *LATTICE with Adjusted Row Depths*



This next example shows another way that the ROWWEIGHTS= and COLUMNWEIGHTS= options can be used. Figure 10.8 on page 181 shows a two row by one column lattice. The first row is an overlay of a histogram, a density plot, a fringe plot (the short vertical lines below the histogram) representing each observation, and a legend. The second row contains a box plot. The X axes have a uniform scale to ensure that the box plot aligns correctly with the histogram. Because the space that is required to show the second row (box plot) is so much less than the space that is required for the first row, the option ROWEIGHTS=(.9 .1) has been used to reapportion the row space.

Figure 10.8 *Graph with ROWEIGHTS=(.9 .1)*

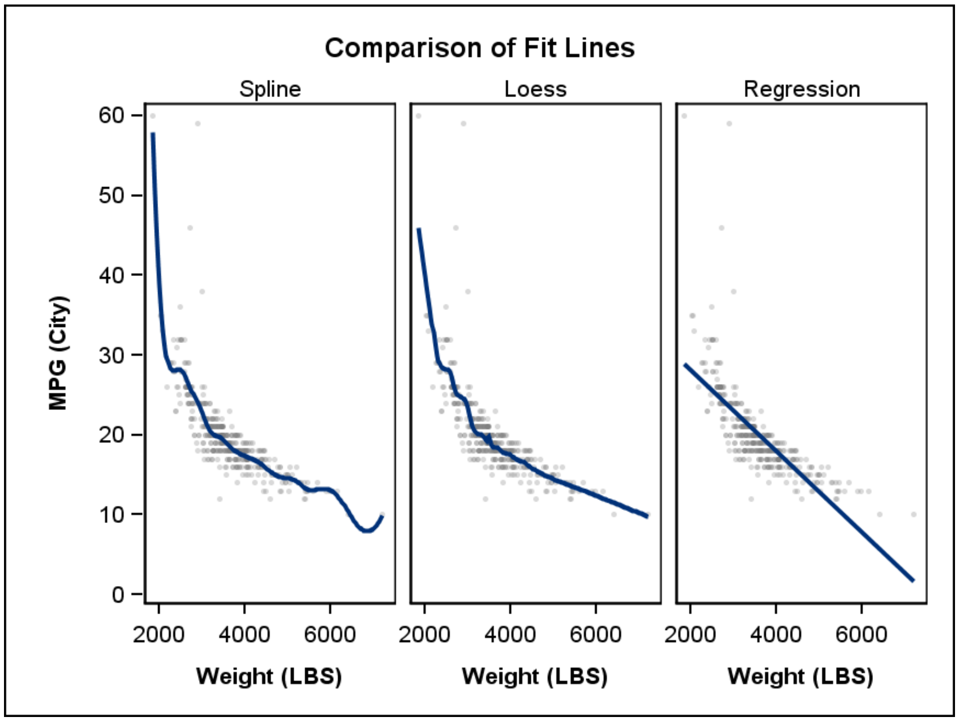



```
proc template;
  define statgraph distribution;
    begingraph;
      entrytitle "Distribution of Cholesterol";
      entryfootnote halign=left
                   "From Framingham Heart Study (SASHELP.HEART)";
      layout lattice / rowweights=(.9 .1)
                       columndatarange=union rowgutter=2px;
        columnaxes;
          columnaxis / display=(ticks tickvalues);
        endcolumnaxes;
        layout overlay / yaxisopts=(offsetmin=.04 griddisplay=auto_on);
          discretelegend "Normal" / location=inside
            autoalign=(topright topleft) opaque=true;
          histogram Cholesterol / scale=percent binaxis=false;
          densityplot Cholesterol / normal( ) name="Normal";
          fringeplot Cholesterol / datatransparency=.7;
        endlayout;
        boxplot y=Cholesterol / orient=horizontal  boxwidth=.9;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.heart template=distribution;
run;
```

For a generic version of this template, which can be used to show the distribution for any continuous variable without redefining the template, see Chapter 14, "Using Dynamics and Macro Variables to Make Flexible Templates," on page 251.

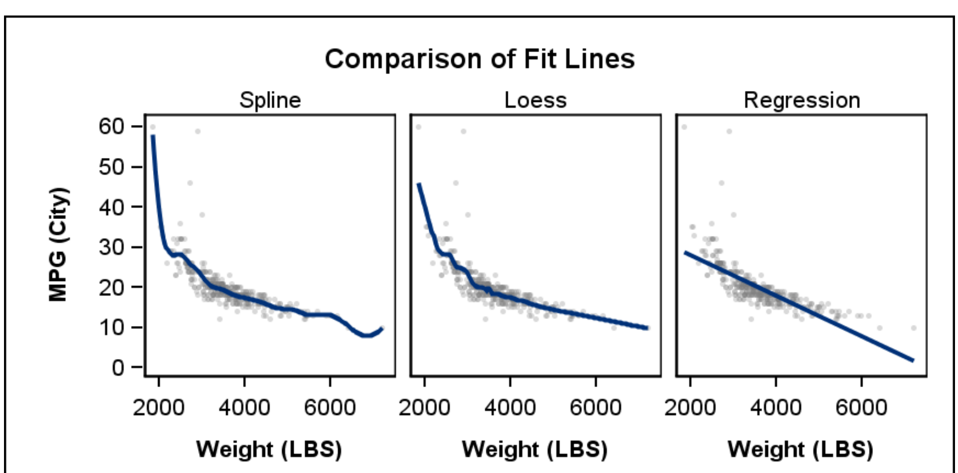
Adjusting the Graph Size

When defining the lattice grid size, you will generally have some idea of a good overall aspect ratio for the graph. For example, if you are creating a one row by three column grid, the graph has a default aspect ratio of 4:3. It would look something like this:



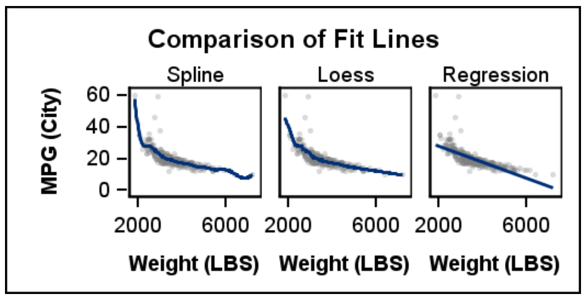
The graph would look better if the graph's height were smaller in relation to its width. You can establish a good default graph size in the template definition by setting the DESIGNWIDTH= and DESIGNHEIGHT= options in the BEGINGRAPH statement. After some experimentation, you might decide that a 2:1 aspect ratio looks good:

```
begingraph / designwidth=400px designheight=200px ;
```



The DESIGNWIDTH= and DESIGNHEIGHT= options set the graph size as part of the template definition so that if you later want a larger or smaller version of this graph, you can use the ODS GRAPHICS statement rather than resetting the design size and recompiling the template. You need only specify either a WIDTH= or a HEIGHT= option in the ODS GRAPHICS statement. The other dimension is automatically computed for you, based on the aspect ratio that is specified in the compiled template by the DESIGNWIDTH= and DESIGNHEIGHT= options.

```
ods graphics / reset width=300px;
proc sgrender data=sashelp.cars template=fitcompare;
run;
```



If you provide both the HEIGHT= and WIDTH= options in the ODS GRAPHICS statement, you completely override the design aspect ratio. If the WIDTH= or HEIGHT= options are not specified, the design size is in effect.

Setting the DESIGNHEIGHT= and DESIGNWIDTH= options is highly recommended for all multi-cell layouts that contain plots. This recommendation applies to the GRIDDED, LATTICE, DATAPANEL, and DATALATTICE layouts.

Chapter 11
Using Classification Panels

Introduction

Classification Panels in the GTL

A classification panel is a graph with one or more cells in which each cell shows a common graph (called a prototype). The prototypes that are displayed in the cells result from dividing input data into subsets that are determined by the values of one or more classification variables. GTL provides two layouts that can produce classification panels:

LAYOUT DATAPANEL
supports a list of class variables. The number of rows and columns are controlled by statement options. Each cell is labeled with the class variable values in the cell header.

LAYOUT DATALATTICE
supports up to two class variables, one for a row variable and one for a column variable. One row of cells is created for each value of the row class variable, and one column is created for each value of the column class variable. The rows and columns are labeled.

The LAYOUT DATAPANEL Statement

The example in this section uses the LAYOUT DATAPANEL statement to specify a list of two classification variables: DIVISION (two distinct values) and PRODUCT (three distinct values). Six combinations (crossings) of these unique values are possible, which produces a panel with six cells.

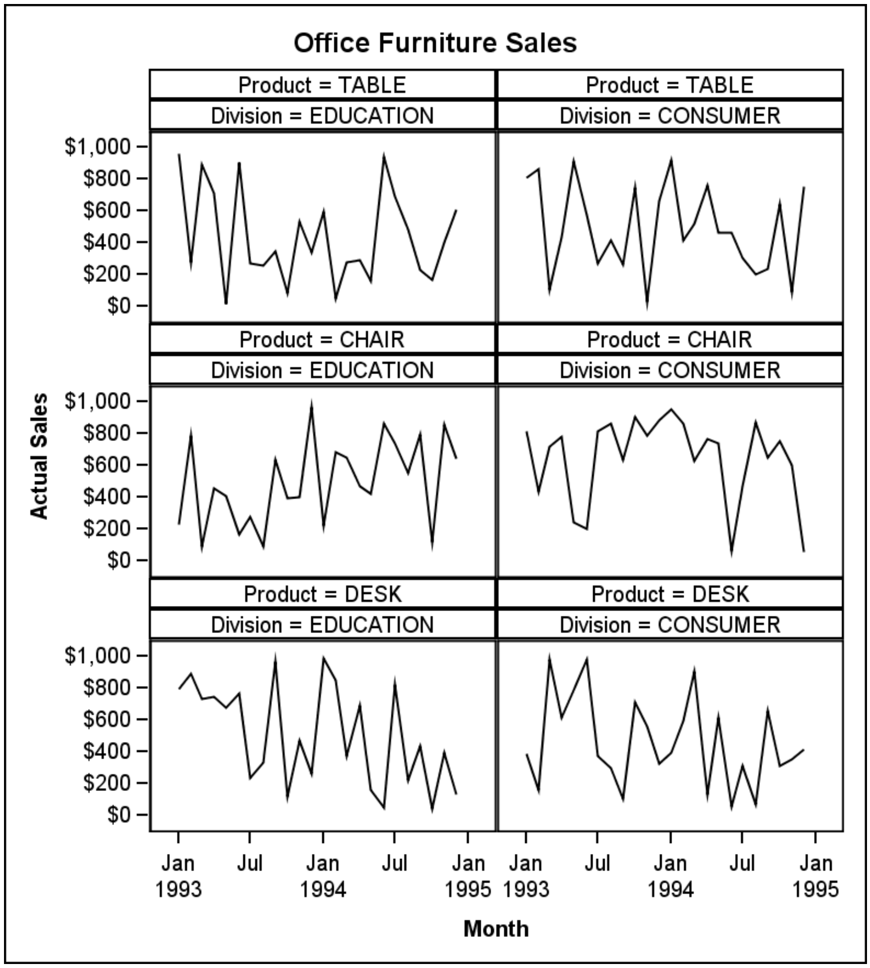
Notice the following details about the LAYOUT DATAPANEL statement:

- The CLASSVARS= option on the LAYOUT DATAPANEL statement can specify a list of one or more classifiers.
- In the resulting graph, the data crossings are identified by the cell headers.

The following template code generates Figure 11.1 on page 187.

```
proc template;
 define statgraph datapanel_intro;
  begingraph;
    entrytitle "Office Furniture Sales";
    layout datapanel classvars=(product division) / columns=2;
      layout prototype;
        seriesplot x=month y=actual;
      endlayout;
    endlayout;
  endgraph;
 end;
run;
```

Figure 11.1 *Classification Panel Created with LAYOUT DATAPANEL*

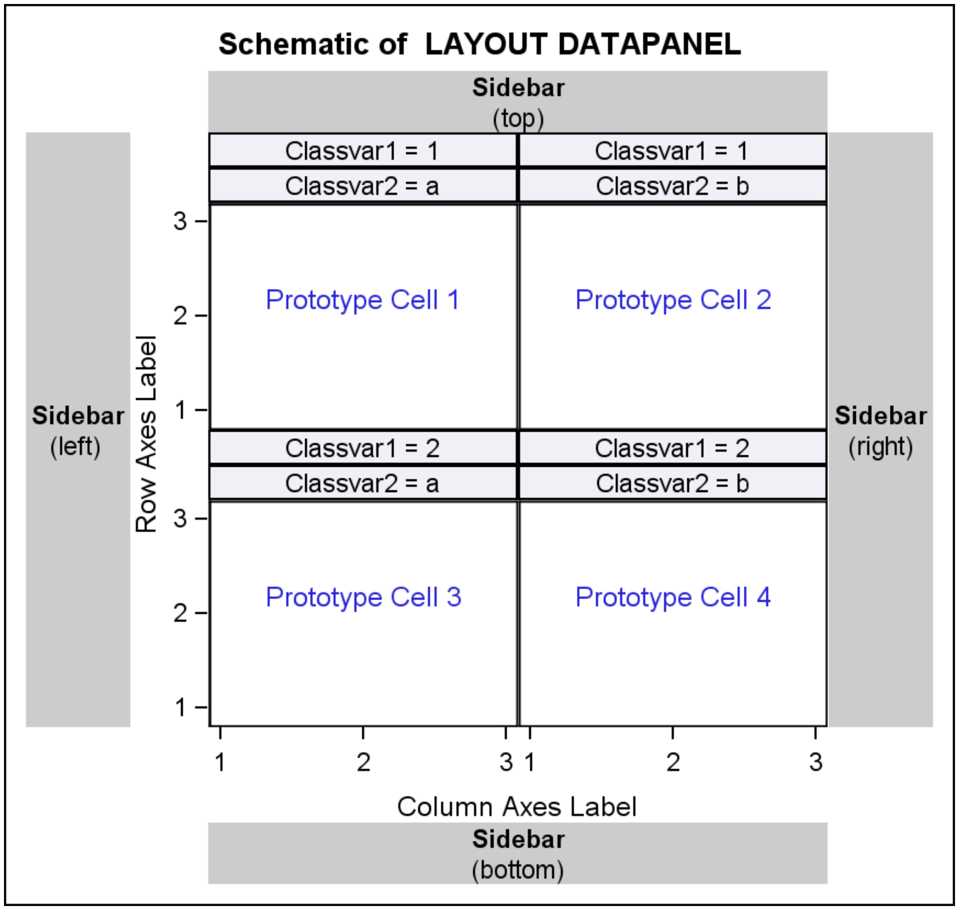


In the template code, notice the LAYOUT PROTOTYPE block, which is inside the LAYOUT DATAPANEL block. This nested block, a required part of the DATAPANEL layout, defines the graphical content of all of the cells. The COLUMNS=2 setting forces a DATAPANEL layout to display the cells in a two-column organization. The actual number of rows that are generated depends on the number of crossings that are in the data.

For some data, the number of data crossings can be quite large. Thus, when rendering the graph for a classification panel, it is common to use a WHERE expression to limit the number of crossings:

```
proc sgrender data=sashelp.prdsale template=datapanel_intro;
  where country="U.S.A." and region="EAST" and
        product in ("CHAIR" "DESK" "TABLE") ;
  format actual dollar.;
run;
```

The following schematic shows the general organization of a graph that is produced with the DATAPANEL layout. If the template code does not use the sidebar areas that are shown in the schematic, that space is reclaimed in the graph. Also, the order in which you specify the classification variables affects the cell ordering. The graph that is represented by the schematic could be produced with **`CLASSVARS=(classvar1 classvar2)`**.



The LAYOUT DATALATTICE Statement

The example in this section uses the LAYOUT DATALATTICE statement to specify the same two classification variables: DIVISION and PRODUCT. Notice the following details about the LAYOUT DATALATTICE statement:

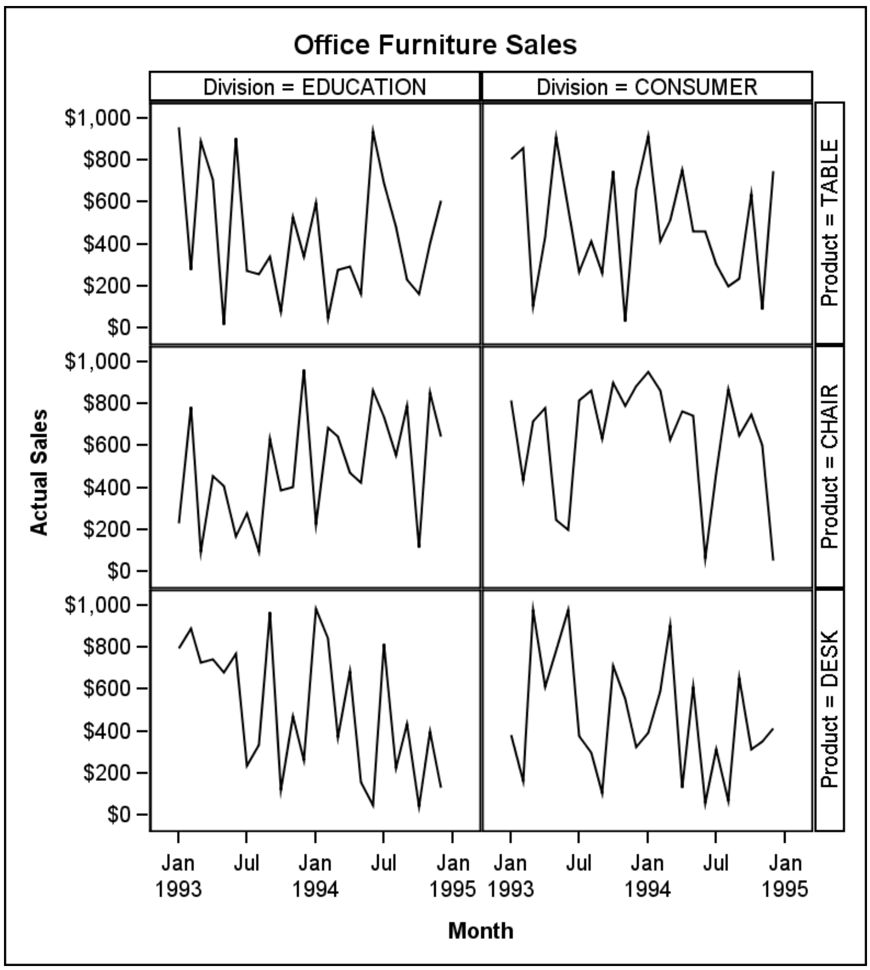
- One of the ROWVAR= or COLUMNVAR= arguments is required. Both can be specified. Each specifies a single classification variable, enabling you to specify either one or two classifiers for the graph.
- In the resulting graph, the data crossings are identified by row or column headers.
- The default number of columns equals the number of unique values for the COLUMNVAR classifier.
- The default number of rows equals the number of unique values for the ROWVAR classifier.

The following template code generates Figure 11.2 on page 189.

```
proc template;
define statgraph datalattice_intro;
  begingraph;
    entrytitle "Office Furniture Sales";
    layout datalattice rowvar=product columnvar=division;
      layout prototype;
        seriesplot x=month y=actual;
      endlayout;
```

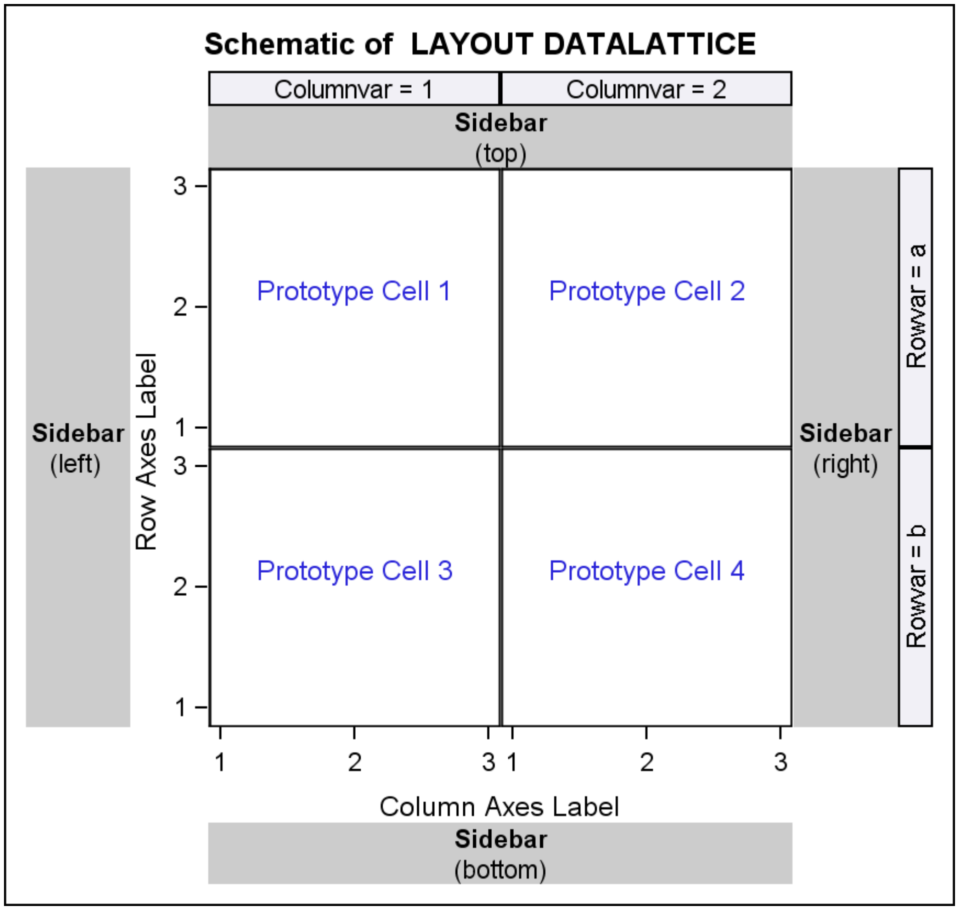
```
    endlayout;
  endgraph;
end;
run;

proc sgrender data=sashelp.prdsale template=datalattice_intro;
  where country="U.S.A." and region="EAST" and
        product in ("CHAIR" "DESK" "TABLE");
  format actual dollar.;
run;
```

Figure 11.2 *Classification Panel Created with LAYOUT DATALATTICE*



In this example, the grid dimensions are automatically determined by the number of distinct values of the classifiers PRODUCT and DIVISION.

The following schematic shows the general organization of a graph that is produced with the DATALATTICE layout. As with a DATAPANEL layout, if the sidebar areas are not used, that space is reclaimed. Notice that the sidebar area is between the cells and the row/column headers.



Coding Distinction Between DATAPANEL and DATALATTICE

The primary difference between coding the DATAPANEL and DATALATTICE layouts is the way that the classification variables are declared.

DATAPANEL takes one list of variables in parentheses. The number of class variables in the list is unlimited, though the effectiveness of the graph decreases as the number of class variables exceeds three or four. In such a case, it is better to use two class variables, and use the other class variables in the BY statement of the SGRENDER procedure.

```
layout datapanel classvars=(product division) / . . . ;
```

DATALATTICE, on the other hand, takes one variable for a row dimension and/or one variable for a column dimension:

```
layout datalattice rowvar=product colvar=division /  . . .;
```

The LAYOUT PROTOTYPE Statement

In both the DATAPANEL and the DATALATTICE blocks, the nested PROTOTYPE layout is similar to an OVERLAY layout, with the following major differences:

- Multiple plots can be overlaid, but BARCHART is the only computed plot that can be included in the prototype. This means that you cannot use BOXPLOT, DENSITYPLOT, ELLIPSE, HISTOGRAM, REGRESSIONPLOT, LOESSPLOT, PBSPLINE, or MODELBAND statements in the PROTOTYPE layout. See “Using

Non-computed Plots in Classification Panels" on page 217 for examples of how to work around this limitation.

- DISCRETELEGEND, CONTINUOUSLEGEND, and ENTRY statements cannot be included in the PROTOTYPE layout, nor can nested layouts. For information on adding a legend or other information outside of the cells, see "Using Sidebars" on page 204.
- Axis options for classification panels are specified on the LAYOUT DATALATTICE or LAYOUT DATAPANEL statement, not on the LAYOUT PROTOTYPE statement. For information on setting axis options for the layout, see "Setting Panel Axis Features" on page 198.

Organizing Panel Contents

Overview

When planning a classification panel, several factors will influence the layout specification:

- Grid dimensions (number of rows and columns)
- Cell population order as the layout is rendered
- Gutters between the cells
- Graph aspect ratio
- Cell size within the panel
- Prototype orientation.

Grid Dimensions and Cell Population Order

Assume you want to create a DATAPANEL layout with one classification variable that has five unique values. Before starting to write code, you must first decide what grid dimensions you want to set (how many columns and rows) and whether you want to permit empty cells in the grid. If do not want empty cells, you must limit the grid to five cells, which gives you two choices for the grid dimensions: five columns by one row (5x1), or one column by five rows (1x5). If you are willing to have empty cells in the grid, you could have several grid sizes, such as a 2x3 or a 3x2 grid.

The easiest way to specify a grid dimension is to set both the COLUMNS= and ROWS= options to the desired number of columns and rows. If one dimension is set, the other dimension automatically grows to accommodate the number of classification levels. By default, COLUMNS=1, and the ROWS= option is not set.

By default, the layout uses the ORDER=ROWMAJOR setting to populate grid cells. This specification essentially means "fill in all cells in the top row (starting at the top left) and then continue to the next row below." The following layout leaves the default ORDER=ROWMAJOR setting in effect:

```
layout datapanel  classvars=(var) / columns=3 rows=2 ;
  layout prototype;
    ... plot statements ...
  endlayout;
endlayout;
```

prototype 1	prototype 2	prototype 3
prototype 4	prototype 5	empty

Alternatively, you can specify ORDER=COLUMNMAJOR, which populates the grid by filling in all cells in the left column (starting at the top), and then continuing with the next column:

```
layout datapanel classvars=(var) / columns=3 rows=2 order=columnmajor ;
  layout prototype;
    ... plot statements ...
  endlayout;
endlayout;
```

prototype 1	prototype 3	prototype 5
prototype 2	prototype 4	empty

One last variation is to specify START=BOTTOMLEFT which produces the following grids, depending on the setting for the ORDER= option:

```
layout datapanel classvars=(var) / columns=3 rows=2 start=bottomleft ;
  layout prototype;
    ... plot statements ...
  endlayout;
endlayout;
```

prototype 4	prototype 5	empty
prototype 1	prototype 2	prototype 3

```
layout datapanel classvars=(var) / columns=3 rows=2
    order=columnmajor start=bottomleft ;
  layout prototype;
    ... plot statements ...
  endlayout;
endlayout;
```

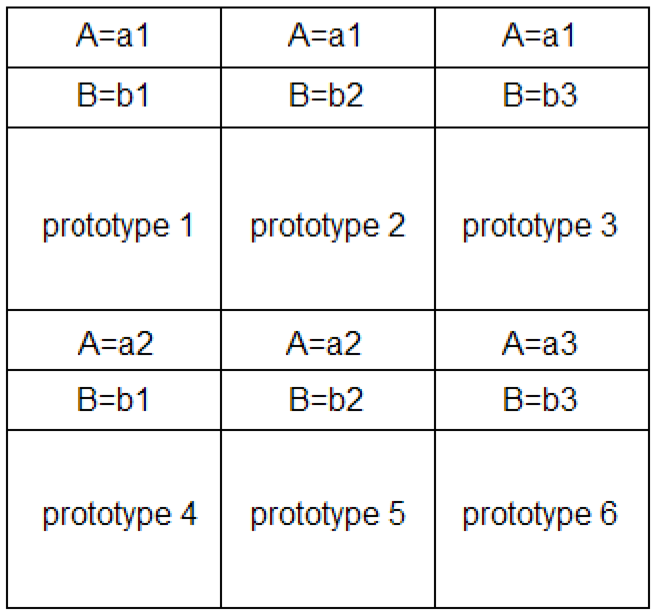
prototype 2	prototype 4	empty
prototype 1	prototype 3	prototype 5

Note: The ROWS=, COLUMNS=, and START= options are available on both the DATAPANEL and DATALATTICE layouts. The ORDER= option is available only on the DATAPANEL layout.

If the number of unique values of the classifiers exceeds the number of defined cells, you automatically get as many separate panels as it takes to exhaust all the classification levels (assuming the PANELNUMBER= option is not used). So if there are 17 classification levels and you define a 2x3 grid, three panels are created (with different names), and the last panel will have one empty cell. The effect that the classifier values have on the panel display is illustrated in "Controlling the Interactions of Classifiers" on page 206.

When you specify multiple classification variables, the crossings are always generated in a specific way: by cycling though the last classifier, and then the next-to-last, until all classifiers are exhausted. The following illustration assumes that classifier A has distinct values a1 and a2, and that classifier B has distinct values b1, b2, and b3:

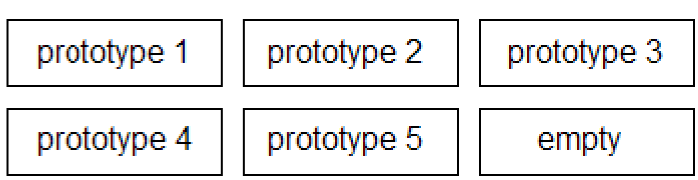
```
layout datapanel classvars=(A B) / columns=3 rows=2 ;
```



Gutters

To conserve space in the graph, the default DATAPANEL and DATALATTICE layouts do not include a gap between cell boundaries in the panel. In some cases, this might cause the cell contents to appear too congested. You can add a vertical gap between all cells with the COLUMNGUTTER= option, and you can add a horizontal gap between all rows with the ROWGUTTER= option. If no units are specified, pixels (PX) are assumed.

```
layout lattice classvars=(var) / columns=3 rows=2
    columngutter=5 rowgutter=5 ;
  layout prototype;
    ... plot statements ...
  endlayout;
endlayout;
```

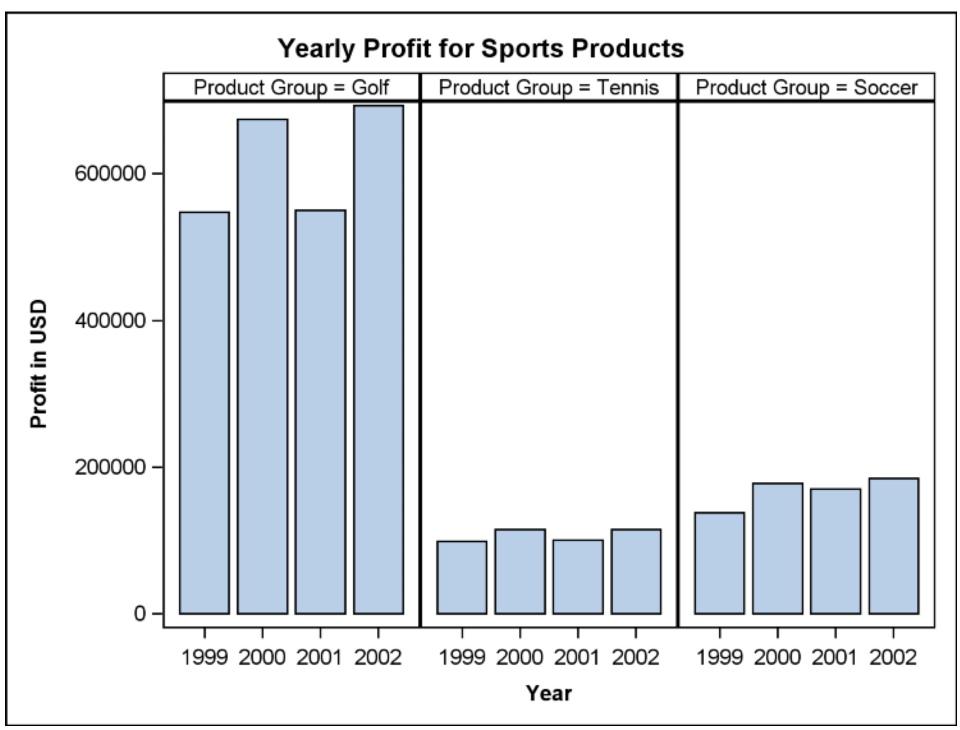


Note that by adding gutters, you do not increase the size of the graph. Instead, the cells shrink to accommodate the gutters. Depending on the number of cells in the grid and the size of the gutters, you will frequently want to adjust the size of the graph to obtain optimal results, especially if the cells contain complex graphs. The issues of graph size and cell size are discussed in the following sections.

Graph Aspect Ratio

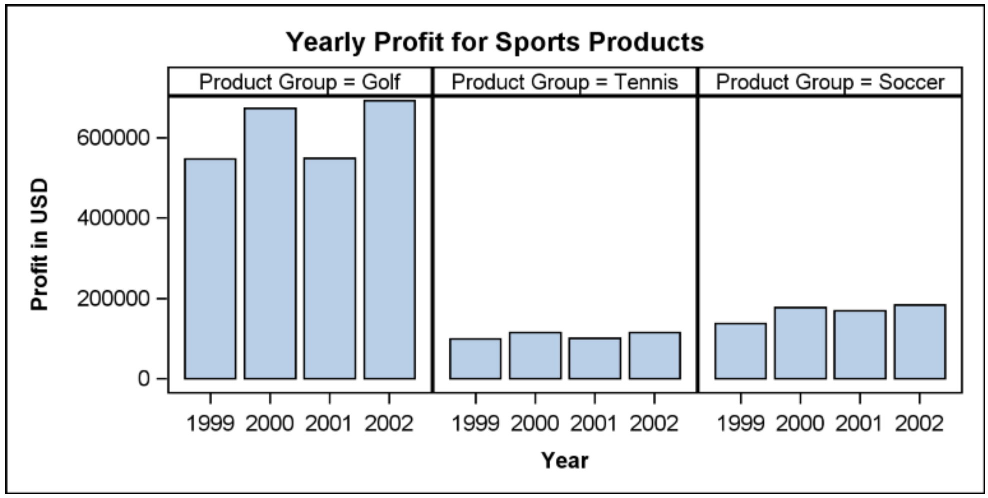
The default graph size is 640 pixels in width and 480 pixels in height, which sets a default aspect ratio of 4:3 (640:480). Depending on your grid size, you might want to adjust the aspect ratio to improve the appearance of the panel. The following example uses a three column by one row grid with the default aspect ratio:

```
proc template;
 define statgraph onerow;
  begingraph;
    entrytitle "Yearly Profit for Sports Products";
    layout datapanel   classvars=(product_group) / rows=1 ;
      layout prototype;
        barchart x=year y=profit / stat=sum ;
      endlayout;
    endlayout;
  endgraph;
 end;
run;

proc sgrender data=sashelp.orsales template=onerow;
  where product_group in ("Golf" "Tennis" "Soccer");
run;
```



In this case, the height of the cells could be reduced to improve the appearance. To adjust the size of the graph, use the DESIGNHEIGHT= and/or DESIGNWIDTH= options in the BEGINGRAPH statement. The following panel is rendered with a 2:1 aspect ratio.

```
begingraph / designwidth=640px designheight=320px ;
   . . .
endgraph;
```



IThe DESIGNWIDTH= and DESIGNHEIGHT= options set the graph size as part of the template definition so that if you later want a larger or smaller version of this graph, you do not have to reset the design size and recompiling the template. Rather, you can specify either a WIDTH= or a HEIGHT= option in the ODS GRAPHICS statement. The other dimension is automatically computed for you, based on the aspect ratio that is specified in the compiled template by the DESIGNWIDTH= and DESIGNHEIGHT= options. For example, the following template produces a 5 inch by 2.5 inch graph (the 2:1 aspect ratio is maintained).

```
ods graphics / reset width=5in ;

proc sgrender data=sashelp.orsales template=onerow;
  where product_group in ("Golf" "Tennis" "Soccer");
run;
```

The following template execution produces a 6 inch by 3 inch output (2:1 aspect ratio is maintained).

```
ods graphics / reset height=3in ;

proc sgrender data=sashelp.orsales template=onerow;
  where product_group in ("Golf" "Tennis" "Soccer");
run;
```

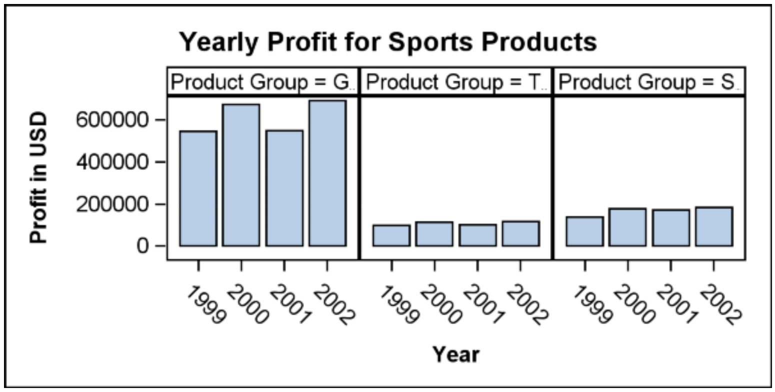
Cell Size

You might think that the panel size can be varied to be as big or small as desired. However, problems arise as the graph size shrinks. Several adjustments in the graph enable small images to be produced:

- Font sizes are reduced.
- Axis tick values are thinned, rotated, or truncated.
- Labels in the cell headers are truncated. (The options that are available for controlling the cell header content and size are discussed in “Controlling the Classification Headers” on page 202.)

For example, the following code sets a 200 pixel height for a classification panel:

```
ods graphics / reset height=200px ;

proc sgrender data=sashelp.orsales template=onerow;
  where product_group in ("Golf" "Tennis" "Soccer");
run;
```



This panel is approaching the limits of how small it can be. Reducing the size even more would eventually produce the following log messages:

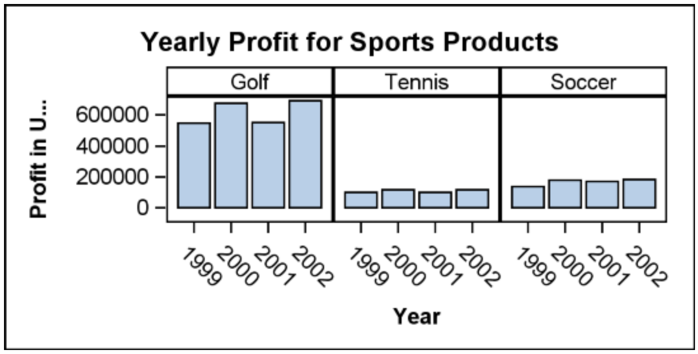
```
Cell width 72 is smaller than the minimum cell width 100. All contents are
removed from the layout.

NOTE: Listing image output written to SGRender.png.
NOTE: There were 48 observations read from the data set SASHELP.ORSALES.
      WHERE product_group in ('Golf', 'Soccer', 'Tennis');
NOTE: PROCEDURE SGRENDER used (Total process time):
      real time           0.50 seconds
      cpu time            0.28 seconds
```

Although an image is produced, it is empty. The GTL has an internal restriction on how small a cell in the panel can be: 100 pixels by 100 pixels. Cell size is computed after all titles, footnotes, and sidebar contents have been established. Thus, if we had additional titles in the panel design, log messages similar to the one just shown would be issued, even with a larger panel size.

The CELLWIDTHMIN= and CELLHEIGHTMIN= options on the LAYOUT DATAPANEL or LAYOUT DATALATTICE statements can be used to specify smaller cell sizes than 100 pixels:

```
proc template;
 define statgraph onerow;
  begingraph / designwidth=360px designheight=180px;
    entrytitle "Yearly Profit for Sports Products";
    layout datapanel classvars=(product_group) / rows=1
                     headerlabeldisplay=value
                     cellwidthmin=70 cellheightmin=70 ;
      layout prototype;
        barchart x=year y=profit / stat=sum;
      endlayout;
    endlayout;
  endgraph;
 end;
run;
```

```
proc sgrender data=sashelp.orsales template=onerow;
  where product_group in ("Golf" "Tennis" "Soccer");
run;
```

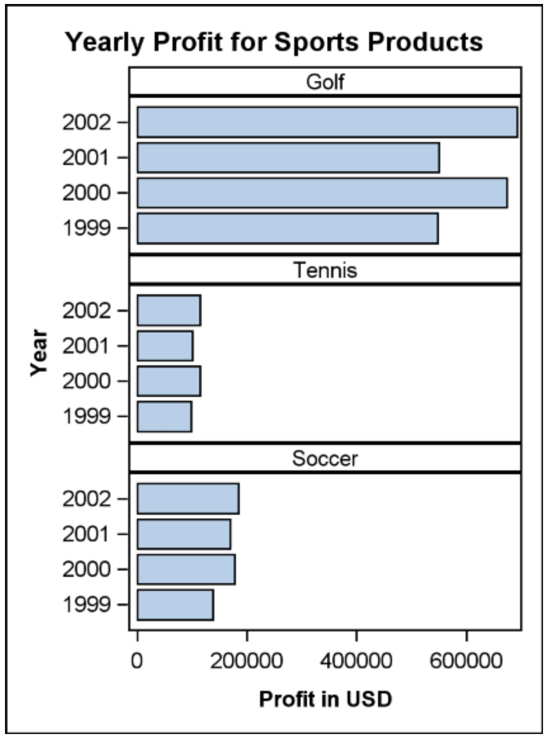


For graph templates that are intended for repeated use (such as the ones that are part of other SAS products), the effort has been made to set the CELLWIDTHMIN= and CELLHEIGHTMIN= option to the smallest values that produce a reasonable panel. Other strategies produce smaller cells without truncating text or resulting in other unwanted side effects. For example, you can change the orientation of the prototype layout.

Prototype Orientation

Rather than generating a graph with the default row orientation, you can present the same information in a column-oriented format. To do so, you should change the design size and also consider changing the orientation of the prototype plot. Prototype plots with discrete axes often benefit from a horizontal orientation because the horizontal alignment can display discrete axis tick values without rotation or truncation (although it might eventually thin or stagger the ticks). The following template code sets a horizontal orientation on a prototype graph.

```
proc template;
 define statgraph onecol;
  begingraph / designwidth=280px designheight=380px ;
    entrytitle "Yearly Profit for Sports Products";
    layout datapanel   classvars=(product_group) / columns=1
           headerlabeldisplay=value
           cellwidthmin=85 cellheightmin=85 ;
      layout prototype;
        barchart x=year y=profit / stat=sum
                                orient=horizontal ;
      endlayout;
    endlayout;
  endgraph;
 end;
run;

proc sgrender data=sashelp.orsales template=onecol;
  where product_group in ("Golf" "Tennis" "Soccer");
run;
```



Setting Panel Axis Features

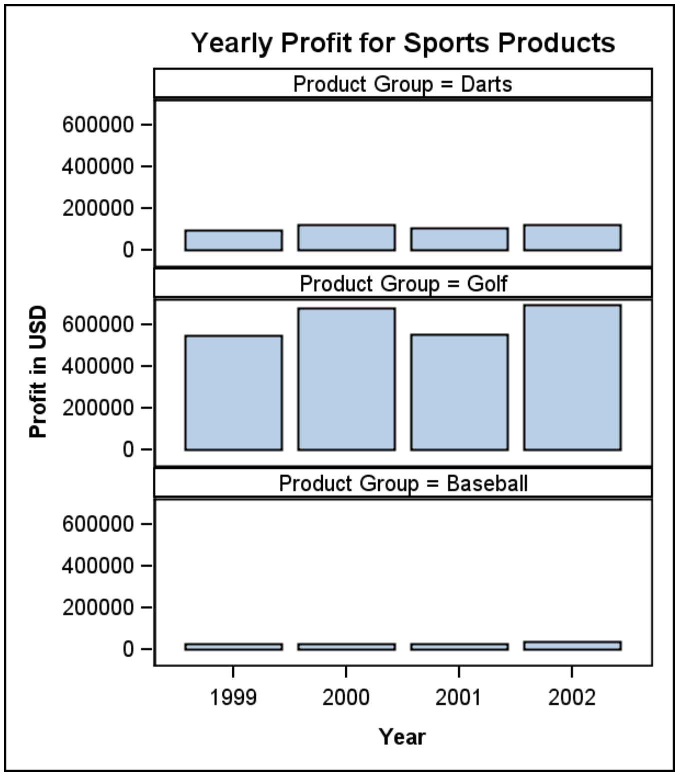
The axes for classification panels are always external to the cells and displayed as axes for the rows or columns.

Controlling Data Ranges of Rows or Columns

The strength of a classification panel presentation is that it makes it easy to visually compare similar plots across data categories. In the following example, the profits for Darts, Golf, and Baseball are compared:

```
proc template;
 define statgraph unionall;
  begingraph / designwidth=350px
               designheight=400px;
    entrytitle
      "Yearly Profit for Sports Products";

    layout datapanel
        classvars=(product_group)/
        rowdatarange=unionall ;
      layout prototype;
        barchart x=year y=profit /
          stat=sum;
      endlayout;
    endlayout;
  endgraph;
 end;
run;
```

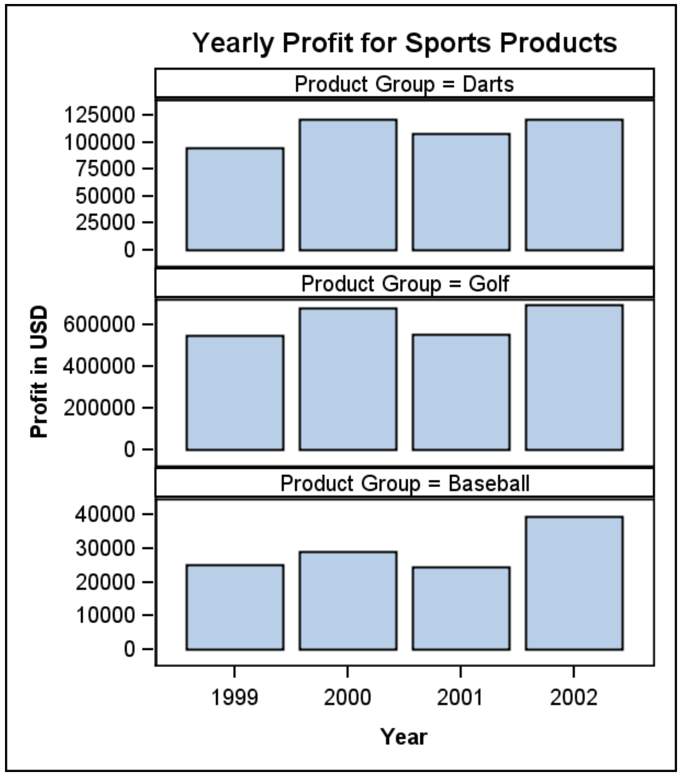
```
proc sgrender data=sashelp.orsales
              template=unionall;
  where product_group in
       ("Golf" "Darts" "Baseball");
run;
```



By default, the minimum and maximum data ranges over all rows in all panels are used to establish identical data ranges across for axes that appear in the rows. The same is true for columns. The options that set these defaults are ROWDATARANGE=UNIONALL and COLUMNDATARANGE=UNIONALL. In most cases, these settings simplify quick comparisons because the axis for each row is scaled identically. Likewise, all columns share a common scale. So the graph just shown does a good job of showing that Golf products in general provide more profits than Darts or Baseball, but it does not do a very good job of showing the yearly variation in Baseball profits because those profits are so small relative to Golf profits.

To set independent axis scaling within each row, you can set ROWDATARANGE=UNION. Similarly, to set independent axis scaling within each column, you can set COLUMNDATARANGE=UNION. The following panel shows independent axes for each row. Now only the data minimum and data maximum for the cells in each row are considered in deciding the axis range.

```
layout datapanel classvars=(product_group)  /
                rowdatarange=union ;
```



In this graph, the relative yearly trends for all product groups are equally apparent, but it is harder to judge which product group is most profitable because bar lengths are comparable only within each row.

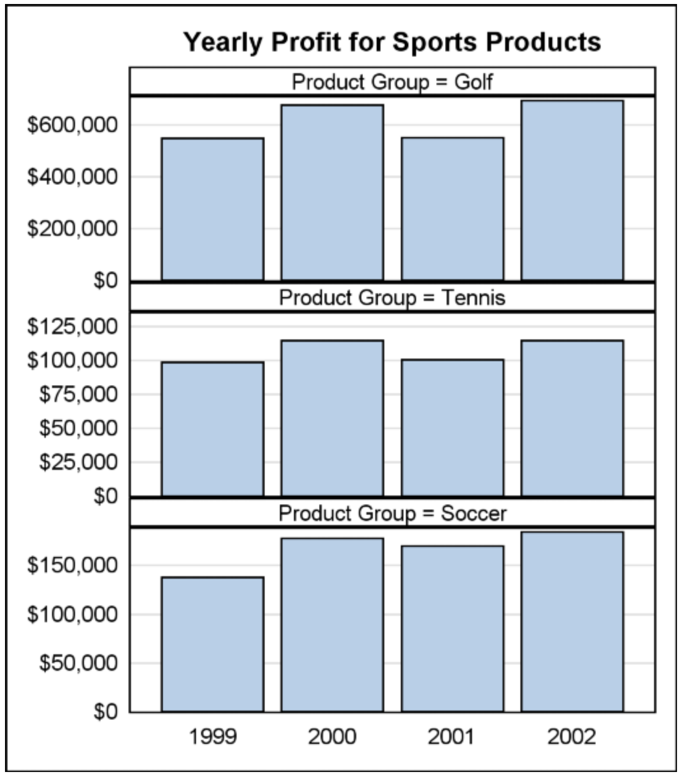
Setting Axis Options

Classification panels use the ROWAXISOPTS=(*axis-opts*) and COLUMNAXISOPTS=(*axis-opts*) options to set axis features. Options are available for all four axis types (LINEAR, DISCRETE, LOG, and TIME), and most of the available axis options are a slightly restricted set of the axis options that are available in an OVERLAY layout.

To demonstrate the use of axis options, the following example suppresses the row axis label because the tick values are formatted with the DOLLAR format and the axis label is therefore not needed. The column axis label is suppressed because the panel's title indicates what the bars represent. Adding title information and eliminating axis labels is a good way to make more space available to the panel's grid. Axis ticks on a discrete axis (YEAR) are often not needed, so the example suppresses them. It also turns on grid lines to make comparisons easier.

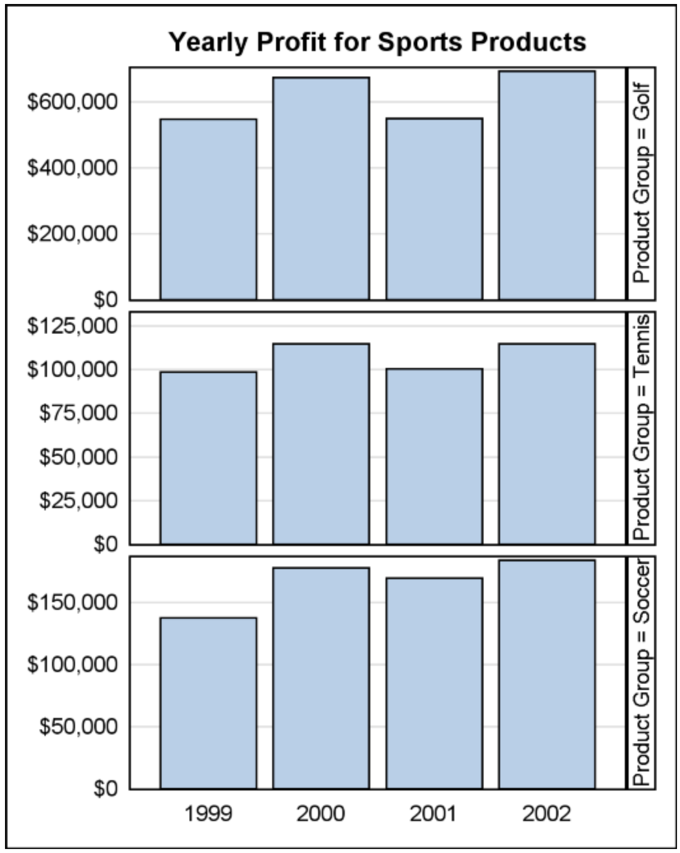
You have probably noticed in the examples with bar charts that the bars do not touch the axis. This happens because a default minimum axis offset is applied to the axis to avoid possible tick value collision with an adjacent cell. This example overrides the default offset by setting OFFSETMIN=0, thus enabling the bars to touch the horizontal axis.

```
layout datapanel  classvars=(product_group)  /
  rowdatarange=union
  columnaxisopts=(display=(tickvalues))
  rowaxisopts=   (display=(tickvalues)
                  linearopts=(tickvalueformat=dollar12.)
                  griddisplay=on offsetmin=0) ;
```



Any DATAPANEL display that uses one or two classifiers can be converted to a DATALATTICE display. When the ROWVAR= option is used on the LAYOUT DATALATTICE statement, the cell headers automatically become row headers. When the COLVAR= option is used, cell headers automatically become column headers. On the following LAYOUT DATALATTICE statement, the ROWVAR= option is used, and the values of the classifier are displayed as row headers:

```
layout datalattice rowvar=product_group  /
  rowdatarange=union
  rowgutter=5px
  columnaxisopts=(display=(tickvalues))
  rowaxisopts   =(display=(tickvalues)
                linearopts= (tickvalueformat=dollar12.)
                griddisplay=on offsetmin=0);
```



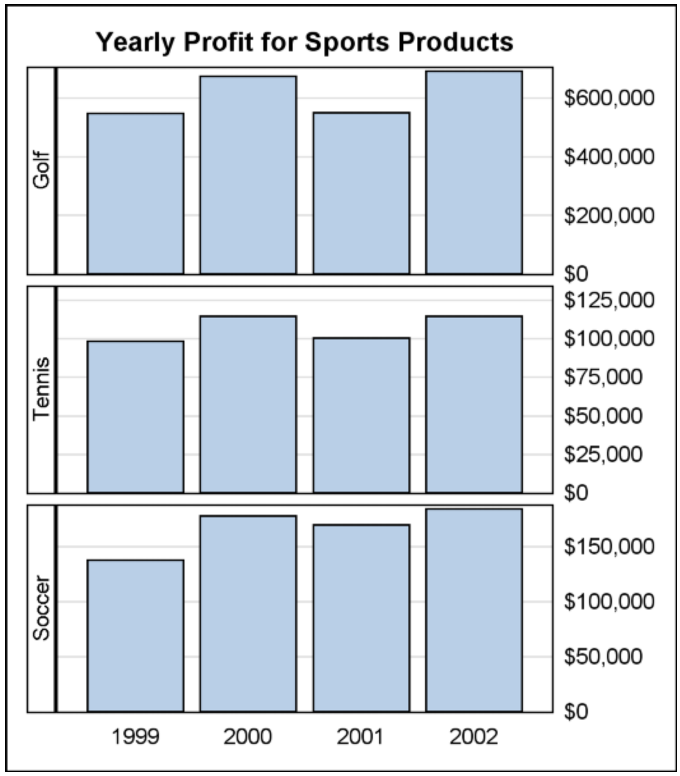
Controlling the Classification Headers

In many cases, it is not necessary to display the classification-variable name in the classification headers. Often, just the classification value is sufficient. Both the DATALATTICE and DATAPANEL layouts support the HEADERLABELDISPLAY= option. By default, HEADERLABELDISPLAY=NAMEVALUE, which shows both the variable name and the value. You can set HEADERLABELDISPLAY=VALUE to display only the value.

Row and column headers are unique to the DATALATTICE layout. By default, COLUMNHEADERS=TOP, but you can set COLUMNHEADERS=BOTTOM or COLUMNHEADERS=BOTH. Likewise, ROWHEADERS=RIGHT is the default setting, but you can set LEFT or BOTH on the ROWHEADERS= option. The location of the row or column axis information can be changed by using the DISPLAYSECONDARY= axis option. In this next example, the row headers are relocated to the left, and the axis information is relocated to the right. Note that DISPLAY=NONE is also needed to remove the default row axis information from the left side.

```
layout datalattice rowvar=product_group  /
  rowdatarange=union
  rowgutter=5px
  rowheaders=left
  headerlabeldisplay=value
  columnaxisopts=( display=(tickvalues) )
  rowaxisopts=   ( display=none displaysecondary=(tickvalues)
```

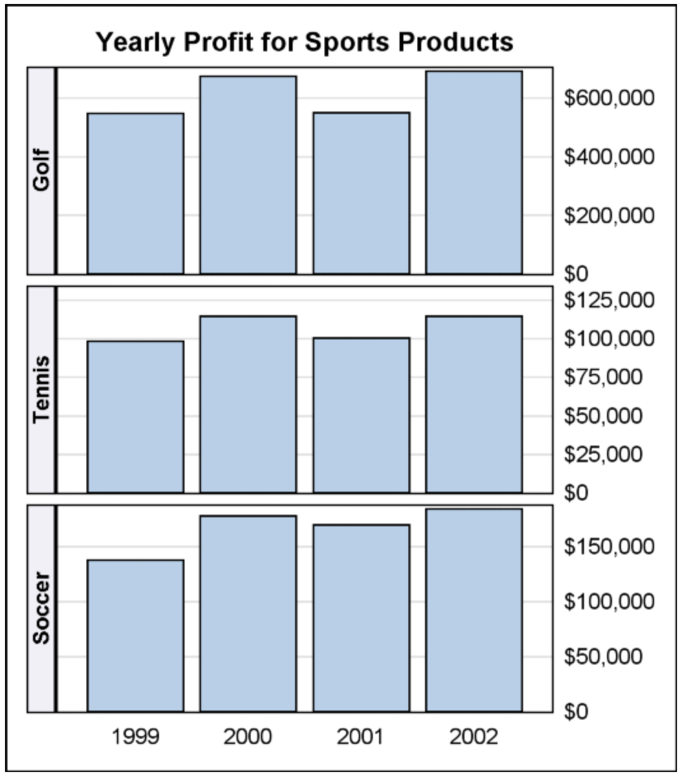
```
                  linearopts=(tickvalueformat=dollar12.)
                  griddisplay=on offsetmin=0 );
```



Both the DATAPANEL and DATALATTICE layouts support options that control the background and text properties of the classification headers. By default, the background of the cell headers is transparent (HEADEROPAQUE=FALSE).

To set a background color, you must set the HEADERBACKGROUNDCOLOR= option to a fill color. In the following example, the color is set as a style reference. You must also set HEADEROPAQUE=TRUE. You can use the HEADERLABELATTRS= option to set the text properties of the classification headers. For example, if the classification values are long, you can reduce their font size with HEADERLABELATTRS= (SIZE=6pt), or use the smallest font in the current style by setting HEADERLABELATTRS=GraphDataText. In the following example, the headers are set to be bold.

```
layout datalattice rowvar=product_group  /
  rowdatarange=union
  rowgutter=5px
  rowheaders=left
  headerlabeldisplay=value
  headerlabelattrs=(weight=bold)
  headeropaque=true
  headerbackgroundcolor=GraphAltBlock:color
  columnaxisopts=(display=(tickvalues) )
  rowaxisopts=   (display=none displaysecondary=(tickvalues)
                  linearopts=(tickvalueformat=dollar12.)
                  griddisplay=on offsetmin=0);
```



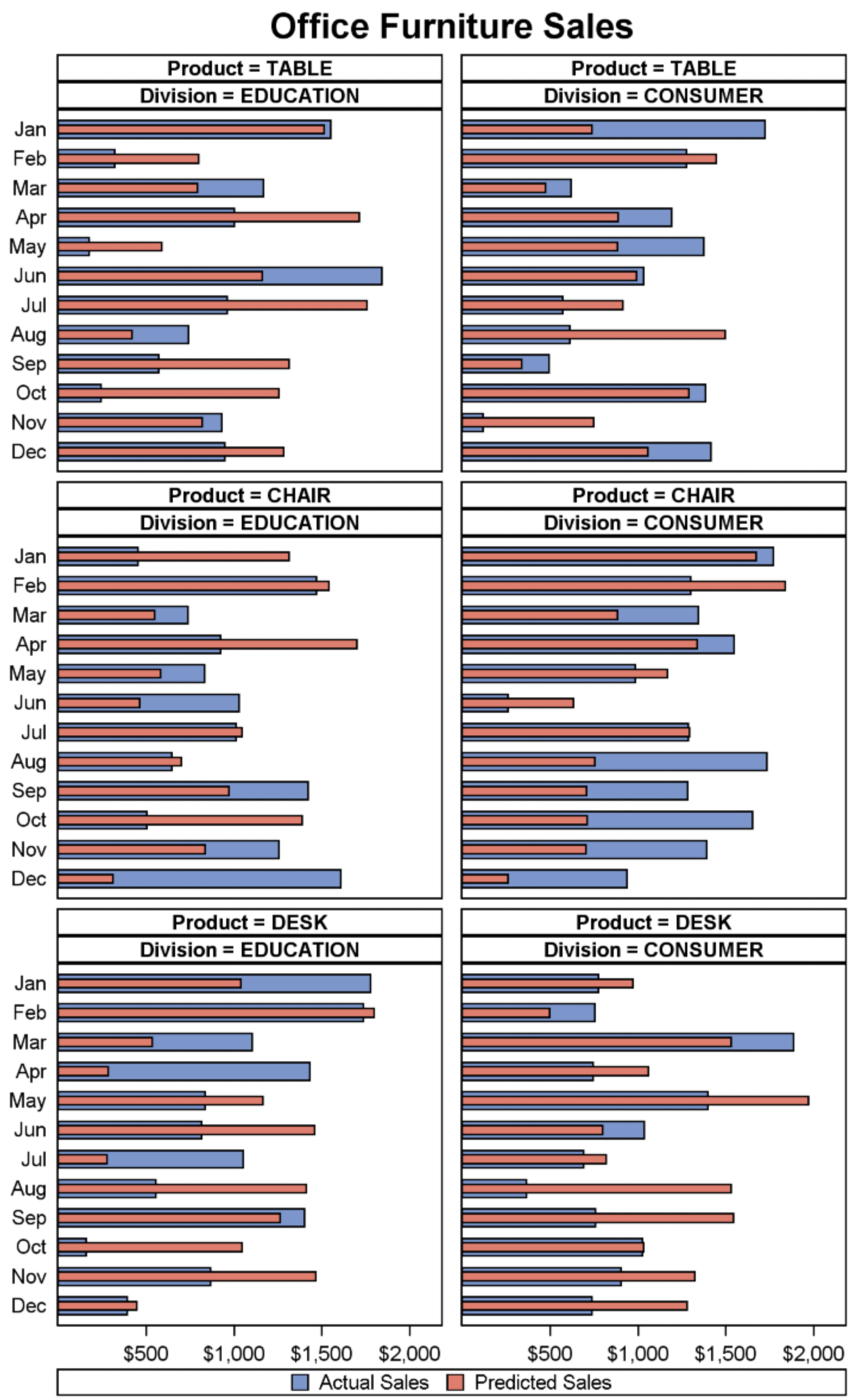
Using Sidebars

Sidebars are useful for aligning information outside of the grid. In the following example, a sidebar is used to display a graph title, rather than using an ENTRYTITLE statement. The advantage of using sidebars for title and footnote information is that a sidebar is always horizontally aligned on the grid itself, not on the complete graph width. Of course, you have to specify the title text on an ENTRY statement, and then set the appropriate text properties (TEXTATTRS= option), alignment (HALIGN= option), and padding (PAD= option). Compare the default centering of the "title" in this example with similar examples in this chapter that specify a title with the ENTRYTITLE statement.

This example also uses a sidebar to display a legend. A legend can be placed in any of the TOP, BOTTOM, RIGHT, or LEFT sidebars. The legend's alignment is based on the grid size, not the graph size.

```
proc template;
 define statgraph sidebar;
   begingraph / designwidth=490px designheight=800px border=false;

     layout datapanel classvars=(product division) / columns=2
         columngutter=10  rowgutter=5
         headerlabelattrs=GraphLabelText(weight=bold)
         rowaxisopts=(display=(tickvalues))
         columnaxisopts=(display=(ticks tickvalues)
           offsetmin=0
           linearopts=(tickvalueformat=dollar6. viewmax=2000
           tickvaluelist=(500 1000 1500 2000)));
       sidebar / align=top;
```

```
        entry "Office Furniture Sales" /
          textattrs=GraphTitleText(size=14pt) pad=(bottom=5px);
      endsidebar;
      sidebar / align=bottom;
        discretelegend "actual" "predict";
      endsidebar;

      layout prototype;
        barchart x=month y=actual  /
          orient=horizontal fillattrs=GraphData1
            barwidth=.6 name="actual";
        barchart x=month y=predict /
          orient=horizontal fillattrs=GraphData2
            barwidth=.3 name="predict";
      endlayout;
    endlayout;
  endgraph;
 end;
run;


proc sgrender data=sashelp.prdsale template=sidebar;
   where country="U.S.A." and region="EAST" and
         product in ("CHAIR" "DESK" "TABLE");
run;
```



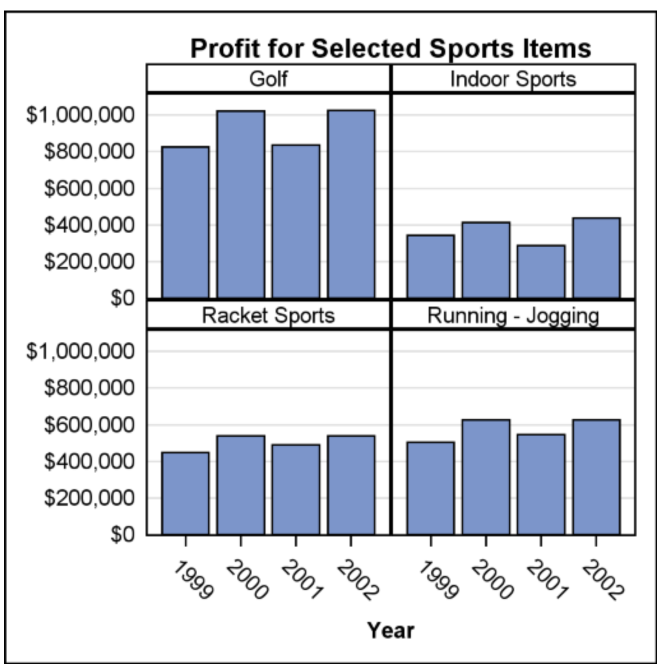
Controlling the Interactions of Classifiers

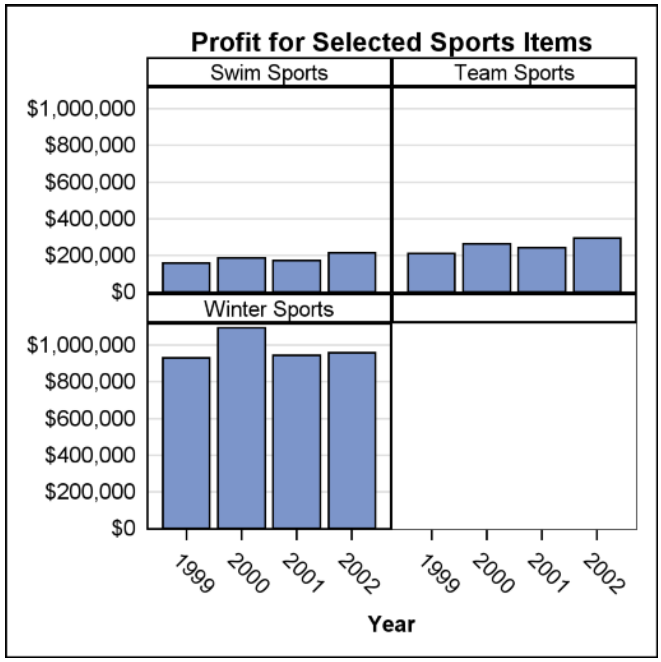
Whenever you have classifiers with a large number of unique levels, the potential exists for generating a large number of cells in the panel. If you do not want to see all classification levels, you can limit the crossings by using a WHERE expression when creating the input data. Or, you can use a WHERE expression as part of the PROC SGRENDER step that renders the graph.

Appearance of the Last Panel

If you set the ROWS= and COLUMNS= options to define a relatively small grid, PROC SGRENDER automatically generates as many separate panels as it takes to exhaust all the classification levels. Depending on the grid size and total number of classification levels, one or more empty cells might be created on the last panel to complete the grid. For example, if there are seven classification levels and you define a 2x2 grid, two panels are created (with different names), and the last panel contains one empty cell:

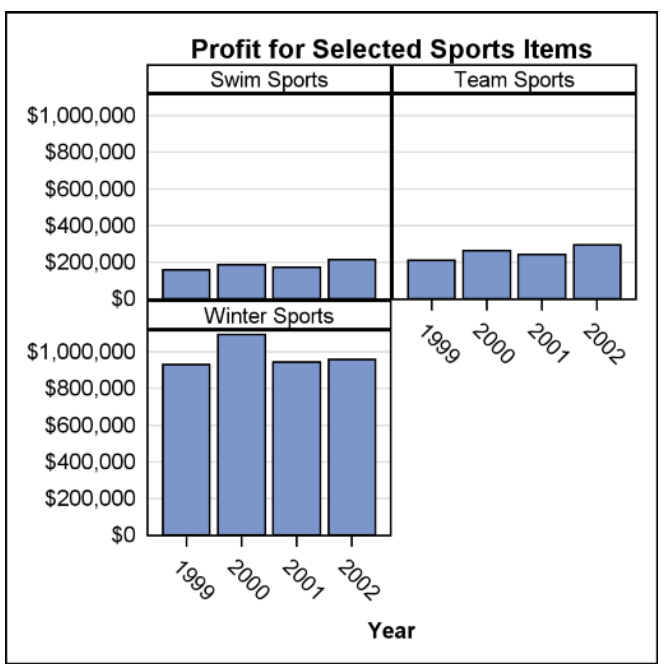
```
layout datapanel classvars=(product_category) /
    rows=2 columns=2
    headerlabeldisplay=value
    rowaxisopts=(griddisplay=on offsetmin=0
    display=(tickvalues) linearopts=(tickvalueformat=dollar12.));
  layout prototype;
    barchart x=year y=profit / fillattrs=GraphData1;
  endlayout;
  sidebar / align=top;
    entry  "Profit for Selected Sports Items" /
      textattrs=GraphTitleText;
  endsidebar;
endlayout;
```



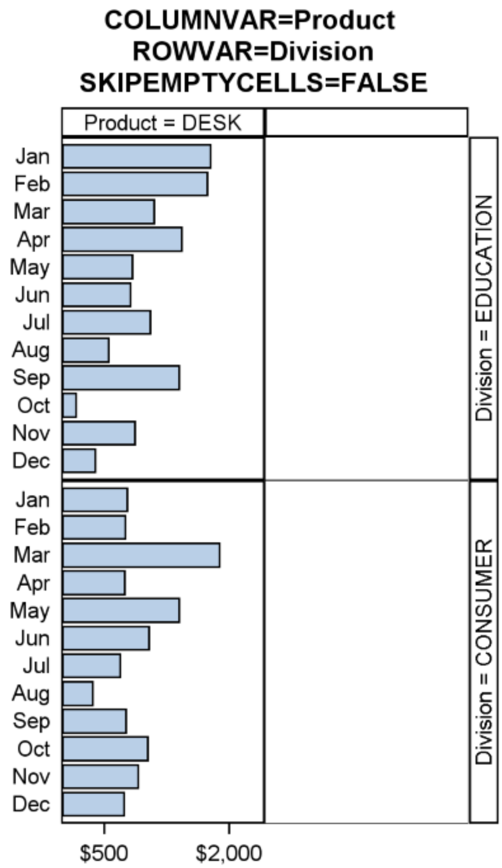


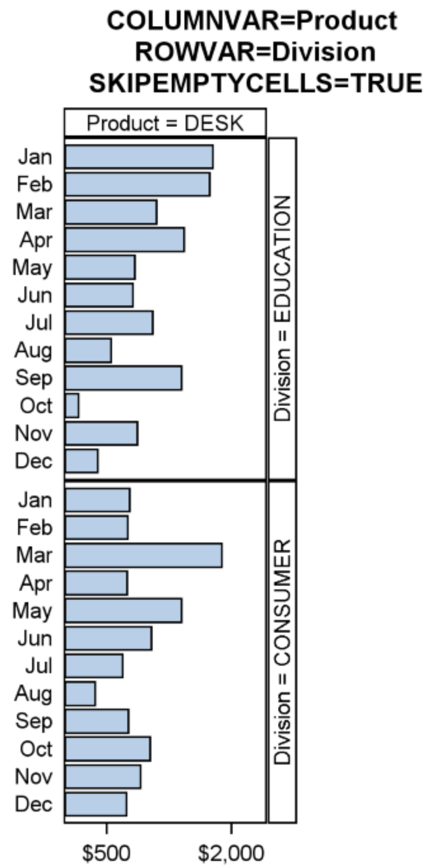
To eliminate empty cells on the last panel, you can specify SKIPEMPTYCELLS=TRUE:

```
layout datapanel classvars=(product_category) /
    rows=2 columns=2
    skipemptycells=true
    headerlabeldisplay=value
    rowaxisopts=(griddisplay=on offsetmin=0
    display=(tickvalues) linearopts=(tickvalueformat=dollar12.));
```



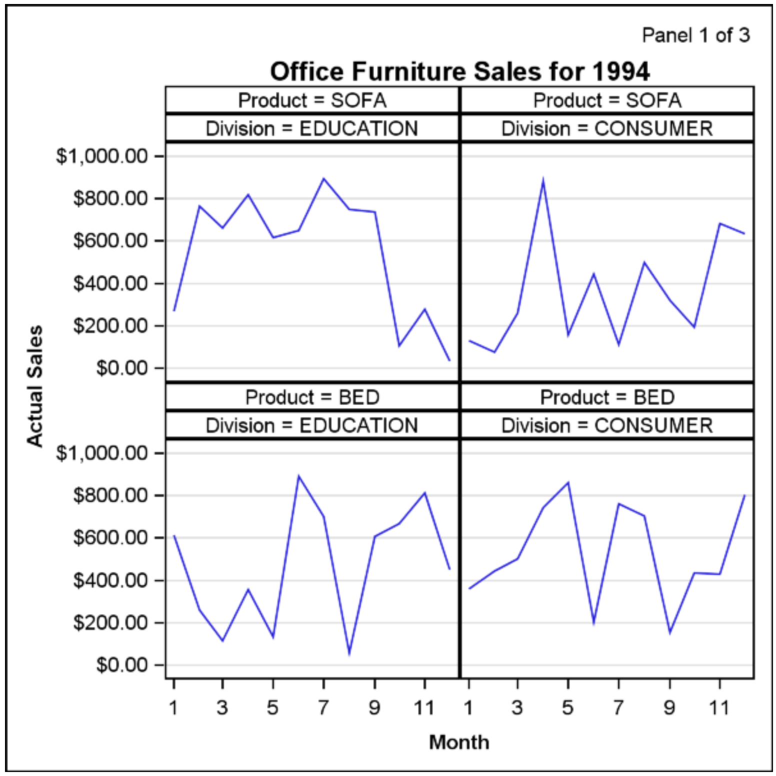
The SKIPEMPTYCELLS= option also applies to a DATALATTICE layout. The following output shows the last panel when Division has two levels and Product has three levels, while ROWS=2 and COLUMNS=2. When SKIPEMPTYCELLS=FALSE, the last panel will have a column of empty cells. Entire rows or columns of empty cells can be removed by setting SKIPEMPTYCELLS=TRUE.



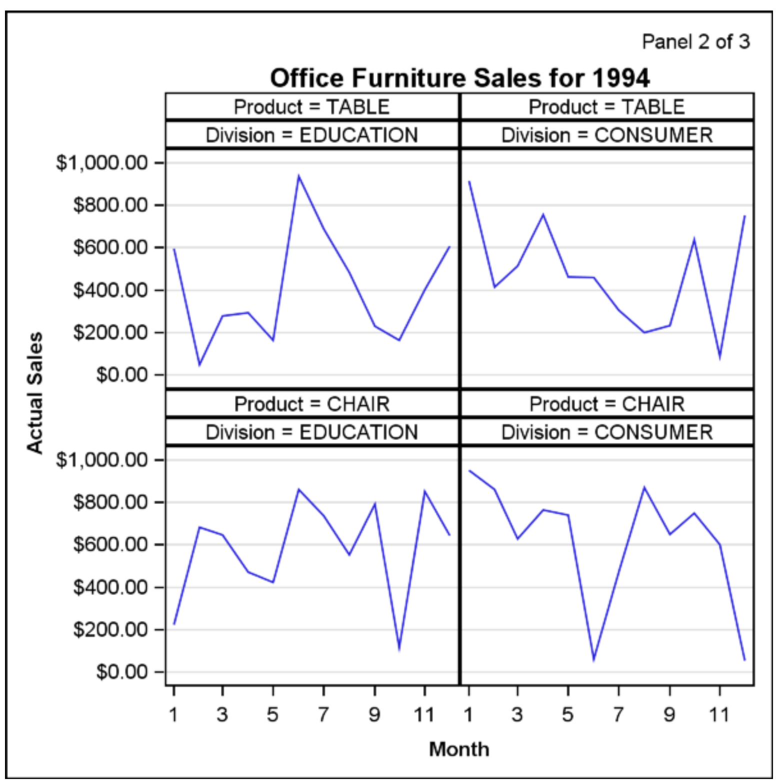


User Control of Panel Generation

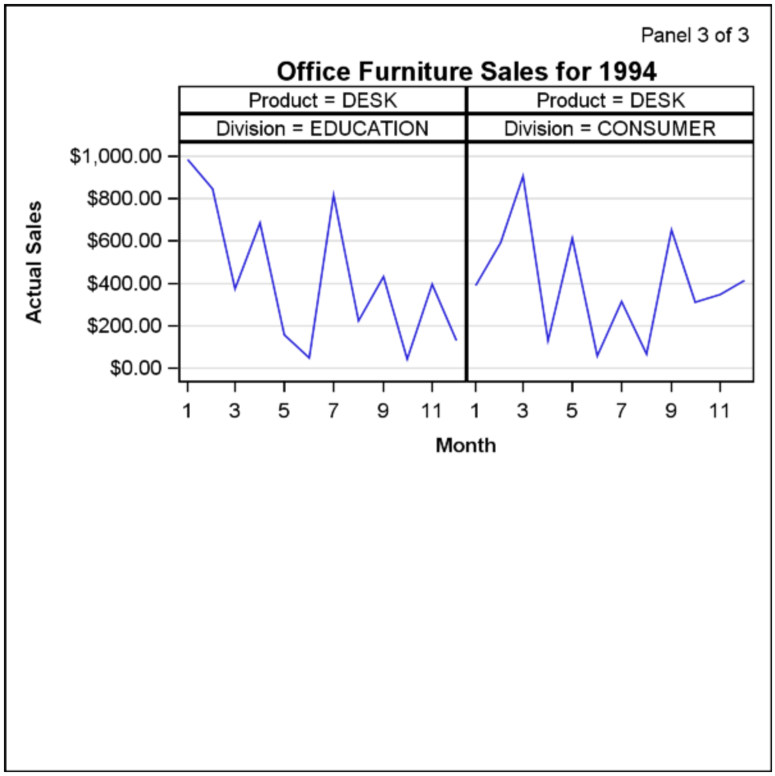
It is possible to control the generation of panels. Consider the following output, in which each panel displays in its upper-right corner the current panel number and the total number of panels:



Panel 1 of 3
Office Furniture Sales for 1994
Product = SOFA
Division = EDUCATION
Product = SOFA
Division = CONSUMER
Product = BED
Division = EDUCATION
Product = BED
Division = CONSUMER
Actual Sales
$1,000.00
$800.00
$600.00
$400.00
$200.00
$0.00
1 3 5 7 9 11
Month



Panel 2 of 3
Office Furniture Sales for 1994
Product = TABLE
Division = EDUCATION
Product = TABLE
Division = CONSUMER
Product = CHAIR
Division = EDUCATION
Product = CHAIR
Division = CONSUMER
Actual Sales
$1,000.00
$800.00
$600.00
$400.00
$200.00
$0.00
1 3 5 7 9 11
Month



Normally, when the number of cells to be created in a panel is greater than the defined panel size in the template (rows * columns), then the SGRENDER procedure automatically produces the number of panel graphs that are necessary to draw all of the cells in the data. However, you can instruct the template to create only one panel, which is specified by the PANELNUMBER= option. This feature can be used to control the creation of the panels.

For example, the preceding panels were generated with the following template code, which uses the NMVAR statement to declare macro variables that will resolve as numbers. The PANELNUMBER=PANELNUM setting is a directive indicating which panel to produce. The ENTRYTITLE statement changes as the panel number changes. For more information on how to pass information to a template at runtime, see Chapter 14, "Using Dynamics and Macro Variables to Make Flexible Templates," on page 251.

```
proc template;
define statgraph panelgen;
  nmvar PANELNUM TOTPANELS ROWS COLS YEAR;
  begingraph;
    entrytitle halign=right "Panel " PANELNUM " of " TOTPANELS /
        textattrs=GraphFootnoteText;
    layout datapanel classvars=(product division) /
        rows=ROWS columns=COLS
        cellheightmin=50 cellwidthmin=50
        skipemptycells=true
        columnaxisopts=(type=time timeopts=(tickvalueformat=month.))
        rowaxisopts=(griddisplay=on)
        panelnumber=PANELNUM;
      layout prototype;
        seriesplot x=month y=actual / lineattrs=GraphData1;
      endlayout;
      sidebar / align=top;
        entry "Office Furniture Sales for " YEAR /
          textattrs=GraphTitleText;
```

```
      endsidebar;
    endlayout;
  endgraph;
end;
run;
```

Now that the template is defined, a macro is needed to compute the number of panels that will be generated, execute PROC SGRENDER an appropriate number of times, and initialize the macro variables that are referenced in the template. The macro parameters ROWS and COLUMNS allow different grid sizes to be used. The graph size changes based on the grid size.

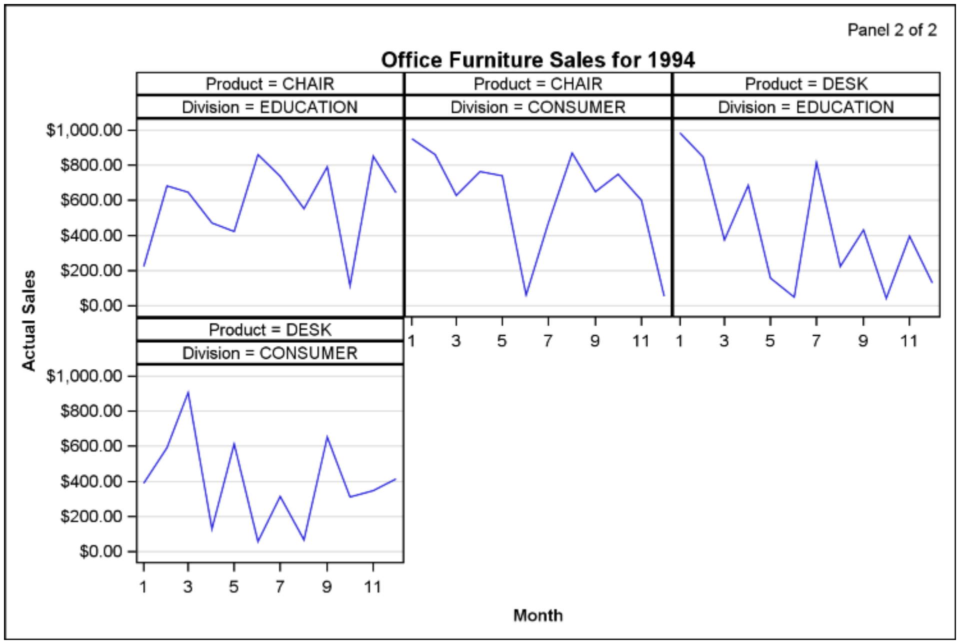
```
%macro panels(rows=1,cols=1,year=1994);
  %local div_vals prod_vals panels totpanels panelnumber;

  /* find the number of unique values for the classifiers */
  proc sql noprint;
    select n(distinct division) into: div_vals from sashelp.prdsale;
    select n(distinct product) into: prod_vals from sashelp.prdsale;
  quit;

  /* compute the number of panels based on input rows and cols */
  %let panels=%sysevalf(&div_vals * &prod_vals / (&rows * &cols));
  %let totpanels=%sysfunc(ceil(&panels)); /* round up to next integer */

  ods graphics / reset ;
  ods listing gpath="C:\temp" image_dpi=200;

  %do panelnum=1 %to &totpanels;

    ods graphics / imagename="Panel&panelnum"
      width=%sysevalf(200*&cols)px height=%sysevalf(200*&rows)px;

    proc sgrender data=sashelp.prdsale template=panelgen;
      where country="U.S.A." and region="EAST" and year=&year;
    run;

  %end;

%mend;
```

The three panels that are shown at the beginning of this section were produced with the following macro call:

```
%panels(rows=2,cols=2)
```

If you invoke the macro with different grid dimensions, the number of panels is recomputed and a new graph size is set. For example, if the following macro call is issued, two panels are generated (only the last panel is shown here):

```
%panels(rows=2,cols=3)
```



Sparse Data

Multiple classifiers sometimes have a hierarchical relationship, which results in very sparse data when the classifier values are crossed. For example, consider the following LAYOUT DATAPANEL statement:

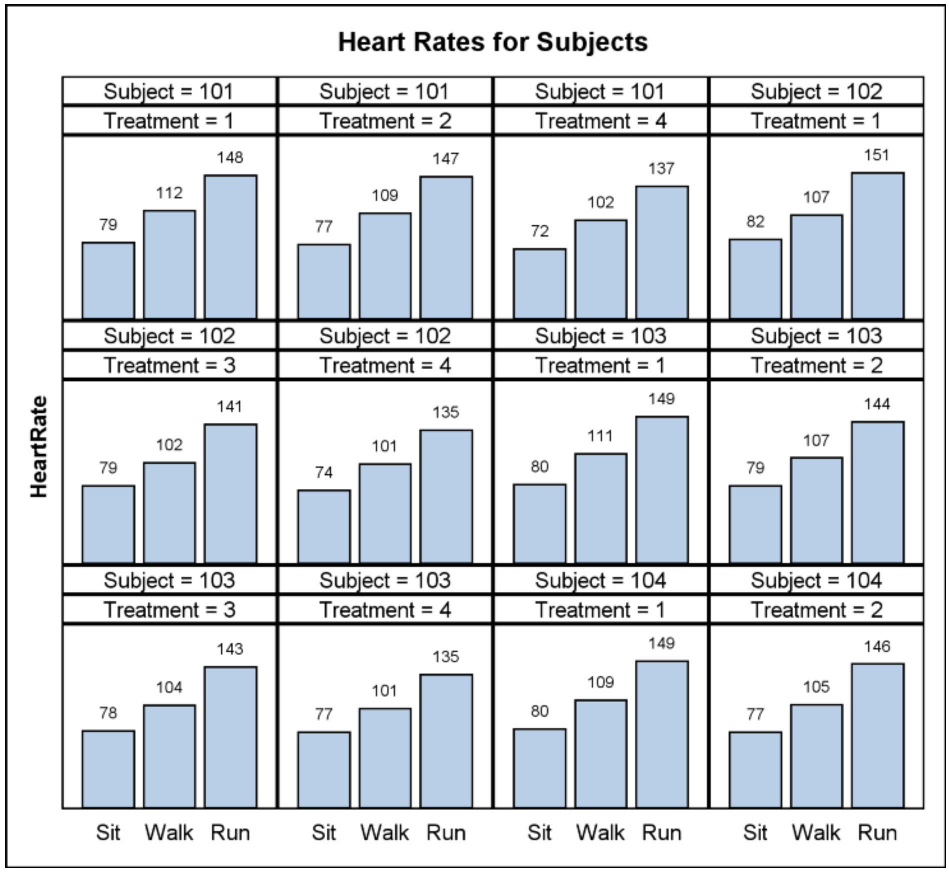
```
layout datapanel classvars=(state city) / rows=4 columns=5;
```

Assume that the data for the STATE and CITY classifiers contains information for 20 states and their capitals. How many panels would you expect to produce? One, or twenty? Or 400?

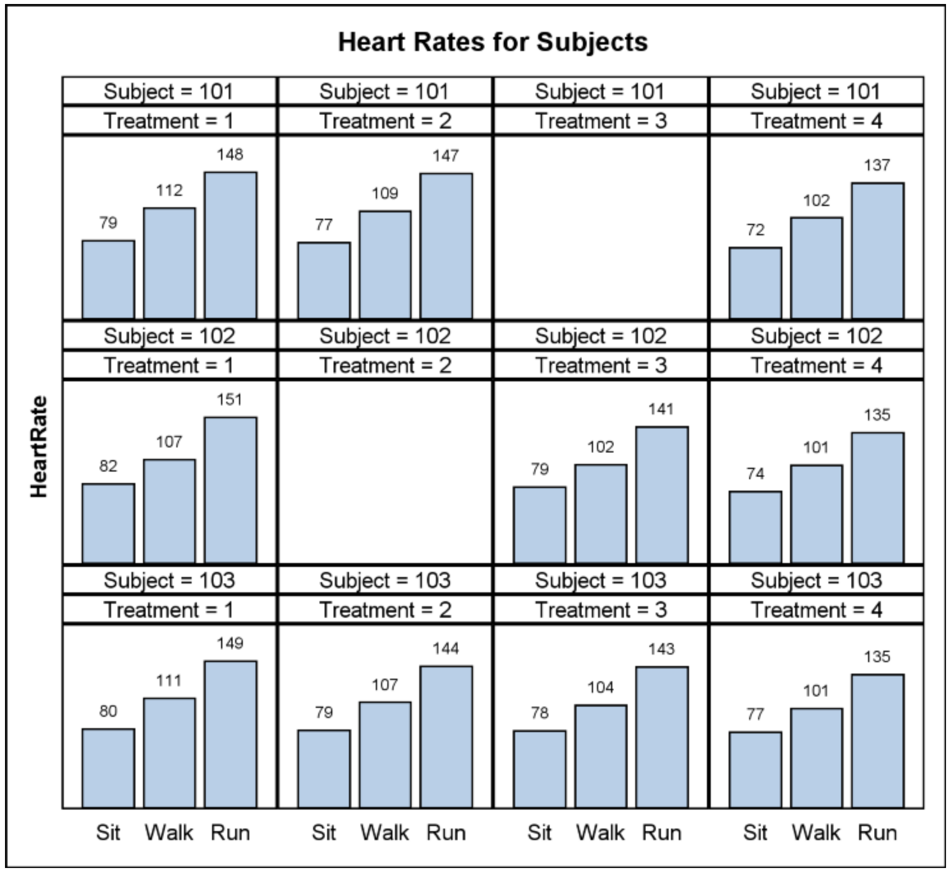
The answer is one panel, which is the desired result. A single panel is produced because even though the default DATAPANEL layout attempts to generate a complete Cartesian product of the crossing values (400 STATE*CITY crossings in this case), it does not create panel cells for crossings that have no data. The SPARSE= option controls whether panel cells are created when you have no observations for a crossing, and by default SPARSE=FALSE.

The DATALATTICE layout does not support a SPARSE= option. The DATALATTICE creates a row / column for each unique value of the ROWVAR / COLUMNVAR. So a cell is created for all crossings of the two variable values, thus creating 400 cells.

Sometimes there are unexpected gaps in the data when classification variables are crossed. For example, suppose you are conducting a study where a number of subjects each receives over time four treatments that might lower the subject's heart rate after various amounts of physical activity. However, assume that Subject 101 didn't get Treatment 3, and Subject 102 didn't get Treatment 2. In this case, when you create a DATAPANEL layout presenting four treatments for three subjects per panel, the expected alignment of the columns does not work:

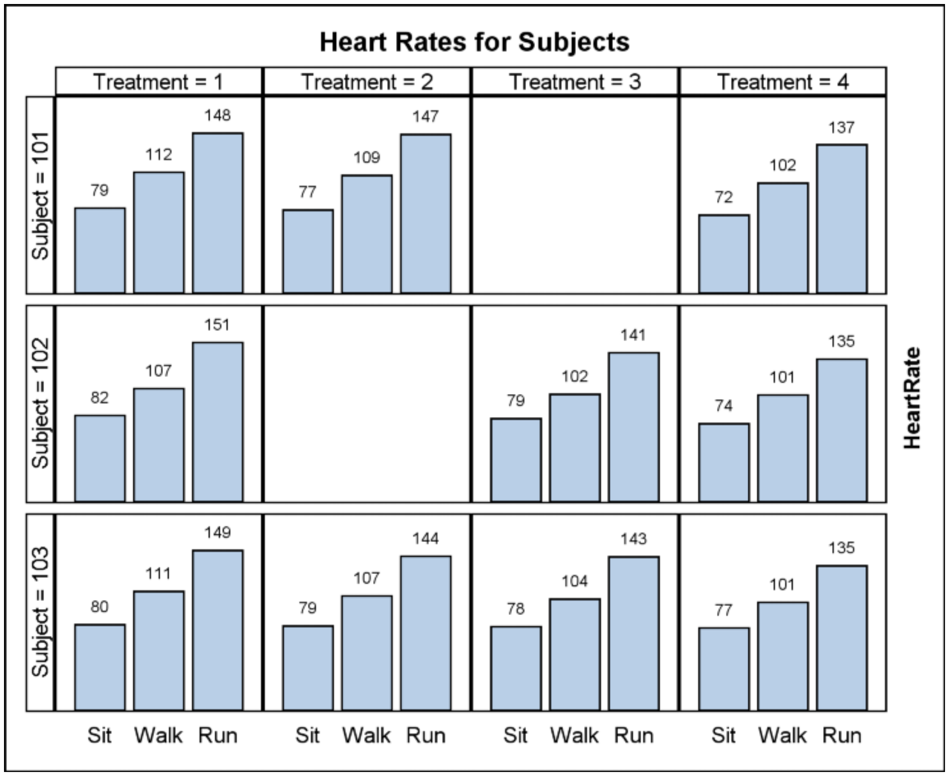


In this situation, you can generate a placeholder cell whenever a subject misses a treatment. To do so, specify SPARSE=TRUE for the layout panel.




```
proc template;
 define statgraph sparse;
  begingraph / designwidth=490px designheight=450px;
    entrytitle  "Heart Rates for Subjects";
    layout datapanel classvars=(subject treatment) /
        columns=4 rows=3
        cellheightmin=50 cellwidthmin=50
        skipemptycells=true
        sparse=true
        columnaxisopts=(display=(tickvalues))
        rowaxisopts=(display=(label) offsetmin=0);
      layout prototype;
        barchart x=task y=heartrate / barlabel=true;
      endlayout;
    endlayout;
  endgraph;
 end;
run;
```

The SPARSE= option does not apply to DATALATTICE layouts because they are inherently sparse. When you specify two classifiers, the DATALATTICE layout manages this situation automatically.

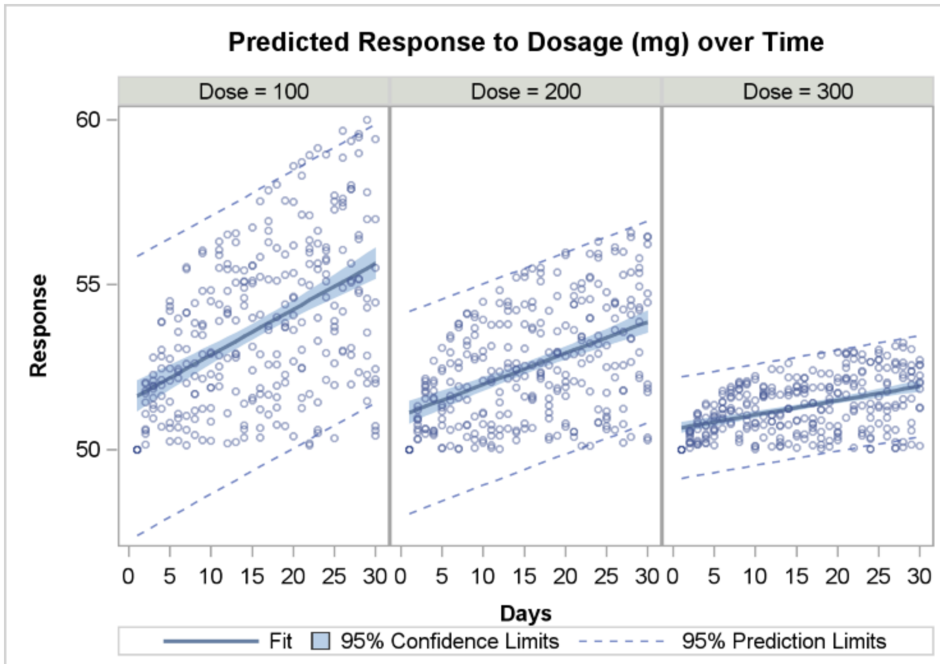



```
proc template;
 define statgraph datalattice;
  begingraph / designwidth=490px designheight=400px;
    entrytitle  "Heart Rates for Subjects";
    layout datalattice rowvar=subject columnvar=treatment /
        rows=3  rowgutter=5px
        cellheightmin=50 cellwidthmin=50
        rowheaders=left
        skipemptycells=true
        columnaxisopts=(display=(tickvalues))
        rowaxisopts=(display=none displaysecondary=(label) offsetmin=0);
      layout prototype;
        barchart x=task y=heartrate / barlabel=true;
      endlayout;
    endlayout;
  endgraph;
 end;
run;
```

Using Non-computed Plots in Classification Panels

So far the discussion has focused on how to set up the grid and axes of the panel using simple prototype examples. However, complex prototype plots can also be specified, although BARCHART is the only computed plot that can be used in the prototype. The restriction of using only non-computed plots in the prototype is mitigated by the fact that most computed plot types are available in a non-computed (parameterized) version—BOXPLOTPARM, ELLIPSEPARM, and HISTOGRAMPARM. Also, the fit line statements (REGRESSIONPLOT, LOESSPLOT, or PBSPLINEPLOT) can be emulated with a SERIESPLOT, and the MODELBAND statement can be emulated with a more general BANDPLOT statement, provided the appropriate variables have been created in the input data. Many SAS/STAT and SAS/ETS procedures can create output data sets with this information.

The following example uses PROC GLM to create an output data set that is suitable for showing a panel of scatter plots with overlaid fit lines and confidence bands.




```
proc template;
define statgraph dosepanel;
 begingraph / designwidth=490px designheight=350px;
   layout datapanel classvars=(dose) / rows=1;
     layout prototype;
       bandplot    x=days limitupper=uclm limitlower=lclm / name="clm"
                     display=(fill)  fillattrs=GraphConfidence
                     legendlabel="95% Confidence Limits";
       bandplot    x=days limitupper=ucl limitlower=lcl / name="cli"
                     display=(outline) outlineattrs=GraphPredictionLimits
                     legendlabel="95% Prediction Limits";
       seriesplot  x=days y=predicted / name="reg"
                     lineattrs=graphFit legendlabel="Fit";
       scatterplot x=days y=response /     primary=true
                     markerattrs=(size=5px) datatransparency=.5;
     endlayout;
     sidebar / align=top;
       entry  "Predicted Response to Dosage (mg) over Time" /
               textattrs=GraphTitleText pad=(bottom=10px);
     endsidebar;
     sidebar / align=bottom;
       discretelegend "reg" "clm" "cli" / across=3;
     endsidebar;
   endlayout;
  endgraph;
```

```
 end;
run;
```

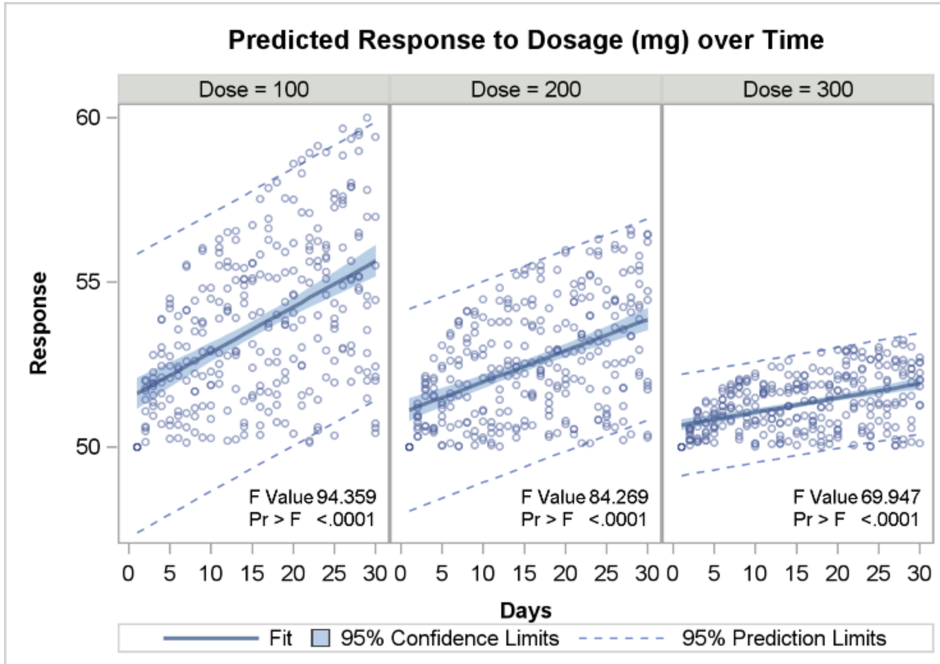
The following procedure code creates the required input data set for the template. It uses a BY statement with the procedure to request the same classification variable that is used in the panel.

```
data trial;
 do Dose = 100 to 300 by 100;
   do Days=1 to 30;
     do Subject=1 to 10;
       Response=log(days)*(400-dose)* .01*ranuni(1) + 50;
     output;
   end;
  end;
end;
run;

proc glm data=trial alpha=.05 noprint;
 by dose;
 model response=days / p cli clm;
 output out=stats
        lclm=lclm uclm=uclm
        lcl=lcl ucl=ucl
        predicted=predicted;
run; quit;

ods listing style=statistical;
proc sgrender data=stats template=dosepanel;
run;
```

The advantage of using a procedure to generate the data is that the statistical procedures provide many options for controlling the model. A robust model enhances the output data set and therefore benefits the graph.

Adding an Inset to Each Cell

You can define a unique inset for each cell of the classification panel with the INSET= and INSETOPTS= options. The following graph builds on the last example by adding insets:




```
proc template;
define statgraph panelinset;
 begingraph / designwidth=495px designheight=350px;
  layout datapanel classvars=(dose) / rows=1
         inset=(F PROB)
         insetopts=(textattrs=(size=7pt) halign=right valign=bottom) ;
    layout prototype;
      bandplot    x=days limitupper=uclm limitlower=lclm / name="clm"
                    display=(fill)  fillattrs=GraphConfidence
                    legendlabel="95% Confidence Limits";
      bandplot    x=days limitupper=ucl limitlower=lcl / name="cli"
                    display=(outline) outlineattrs=GraphPredictionLimits
                    legendlabel="95% Prediction Limits";
      seriesplot  x=days y=predicted / name="reg"
                    lineattrs=graphFit legendlabel="Fit";
      scatterplot x=days y=response /     primary=true
                    markerattrs=(size=5px) datatransparency=.5;
    endlayout;
    sidebar / align=top;
      entry  "Predicted Response to Dosage (mg) over Time" /
               textattrs=GraphTitleText pad=(bottom=10px);
    endsidebar;
    sidebar / align=bottom;
      discretelegend "reg" "clm" "cli" / across=3;
    endsidebar;
  endlayout;
 endgraph;
end;
run;

data trial;
 do Dose = 100 to 300 by 100;
  do Days=1 to 30;
   do Subject=1 to 10;
```

```
      Response=log(days)*(400-dose)* .01*ranuni(1) + 50;
      output;
      end;
    end;
  end;
run;

proc glm data=trial alpha=.05 noprint outstat=outstat ;
  by dose;
  model response=days / p cli clm;
  output out=stats
         lclm=lclm uclm=uclm lcl=lcl ucl=ucl predicted=predicted;
run; quit;

data inset;
  set outstat (keep=F PROB _TYPE_ where=(_TYPE_="SS1"));
  label F="F Value " PROB="Pr > F ";
  format F best6. PROB pvalue6.4;
run;

data stats2;
  merge stats inset;
run;

proc sgrender data=stats2 template=panelinset;
run;
```

In this template definition,

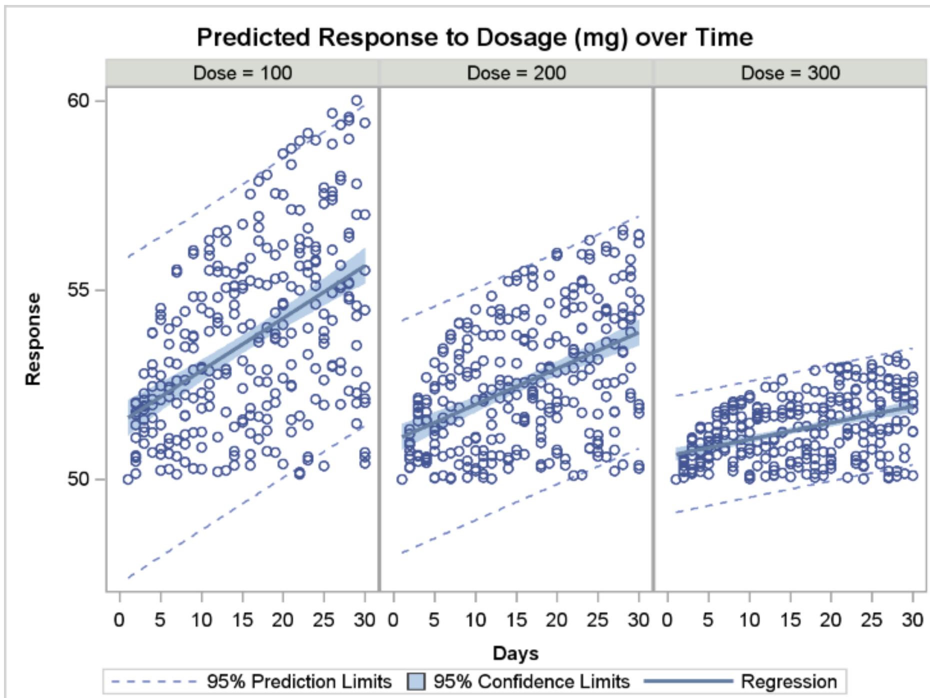
- The INSET=(F PROB) option names two variables that contain the values for the F statistic and its p value. The INSETOPTS= option positions the inset and sets its text properties.
- The OUTSTAT= option of PROC GLM creates a data set with several statistics for each BY value.
- The DATA INSET step selects the appropriate three observations from the OUTSTAT data set. The F and PROB variables are assigned labels and formats.
- The DATA STATS2 step creates a new input data set by performing a non-match merge on the STATS and INSET data sets. It is important to structure the input data in this fashion.

"Adding Insets to Classification Panels" on page 288discusses this topic in detail and shows the coding for another example in which the inset information must align correctly in a multi-row and multi-column classification panel.

Using PROC SGPANEL to Create Classification Panels

When creating a panel like the one shown in "Using Non-computed Plots in Classification Panels" on page 217, you might find it easier to create the panel by using PROC SGPANEL in SAS/GRAPH because the procedure does all the necessary data computations for you. For example, the REGRESSIONPLOT, LOESSPLOT, and PBSPLINEPLOT statements have been incorporated into the SGPANEL procedure as REG, LOESS, and PBSPLINE statements. (SGPANEL can also generate other plot types.) By default on PROC SGPANEL, the PANELBY statement creates a DATAPANEL layout.

```
ods listing style=statistical;

title "Predicted Response to Dosage (mg) over Time";
proc sgpanel data=trial;
  panelby dose / rows=1;
  reg x=days y=response / cli clm;
run;
```



Most, but not all, features of the DATALATTICE and DATAPANEL layouts are provided in the SGPANEL procedure.

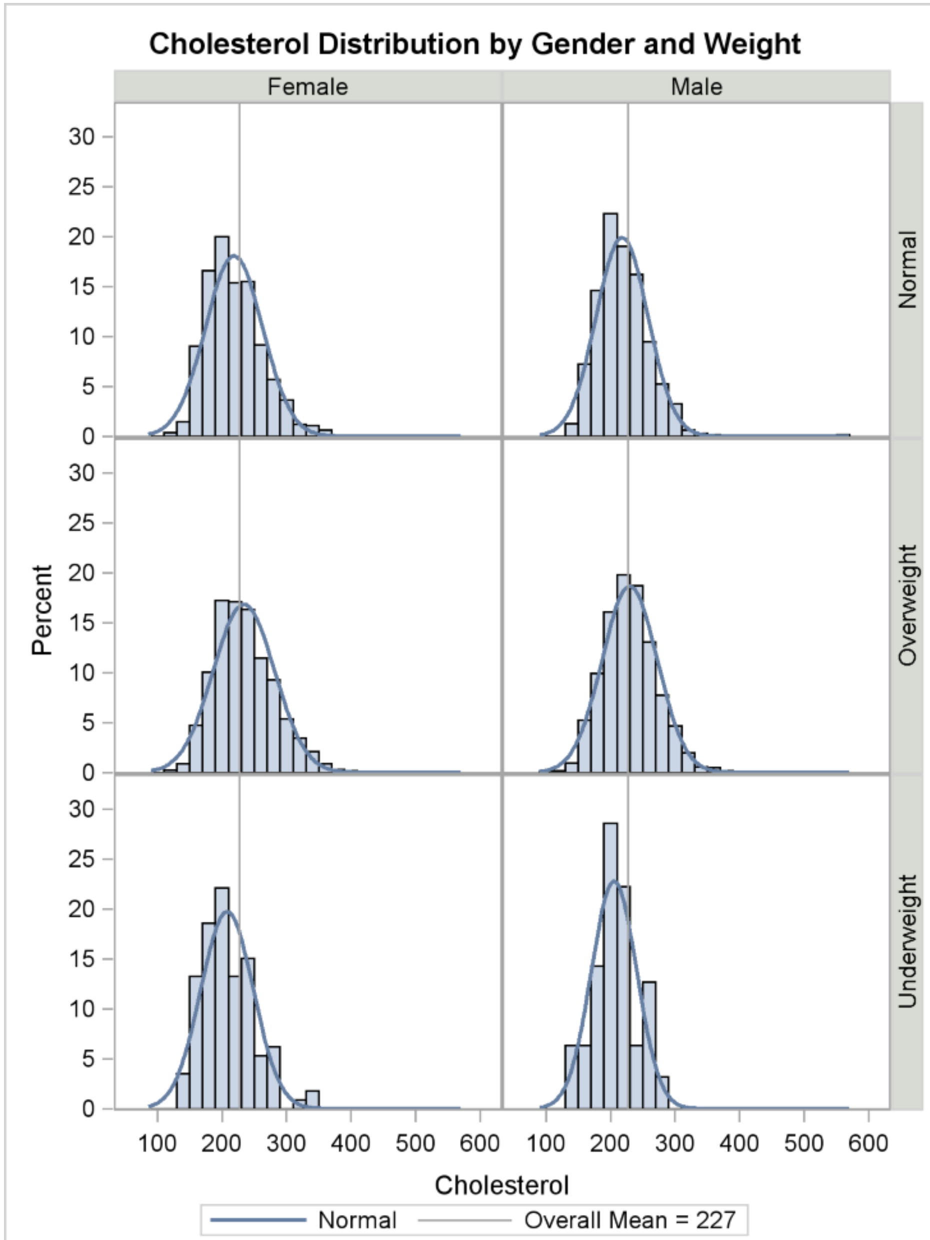
The SGPANEL procedure supports computed plot statements such as HISTOGRAM, DENSITY, DOT, VBOX, and HBOX (vertical and horizontal box plots). The PANELBY statement controls the layout, determining whether a DATAPANEL, DATALATTICE, or other layout is used to produce the graph. ROWAXIS and COLAXIS statements control the external axes, and the KEYLEGEND statement creates legends, which are placed in sidebars for you.

The SGPANEL procedure does not have a PROTOTYPE block because all of the plot statements after PANELBY are considered part of the prototype. The SGPANEL procedure generates GTL template code behind the scenes and executes the template to create its output. See the *SAS/GRAPH Statistical Graphics Procedures Guide* documentation for details.

The following example shows additional features of SGPANEL:

```
ods listing style=statistical;

title "Cholesterol Distribution by Gender and Weight";
proc sgpanel data=sashelp.heart;
  panelby sex weight_status /  layout=lattice onepanel novarname;
  histogram cholesterol;
```

```
  density cholesterol / name="density";
  refline 227 / axis=x name="ref" legendlabel="Overall Mean = 227";
  rowaxis  offsetmin=0 offsetmax=.1 max=30;
  keylegend  "density" "ref";
run;
```



Chapter 12
Using an Equated Layout

The LAYOUT OVERLAYEQUATED Statement

Several SAS procedures create plots where the X and Y axes are scaled in the same units. Here are some samples of such plots taken from the Examples section of the procedure documentation.

Figure 12.1 *Sample Plot from PROC PRINQUAL*

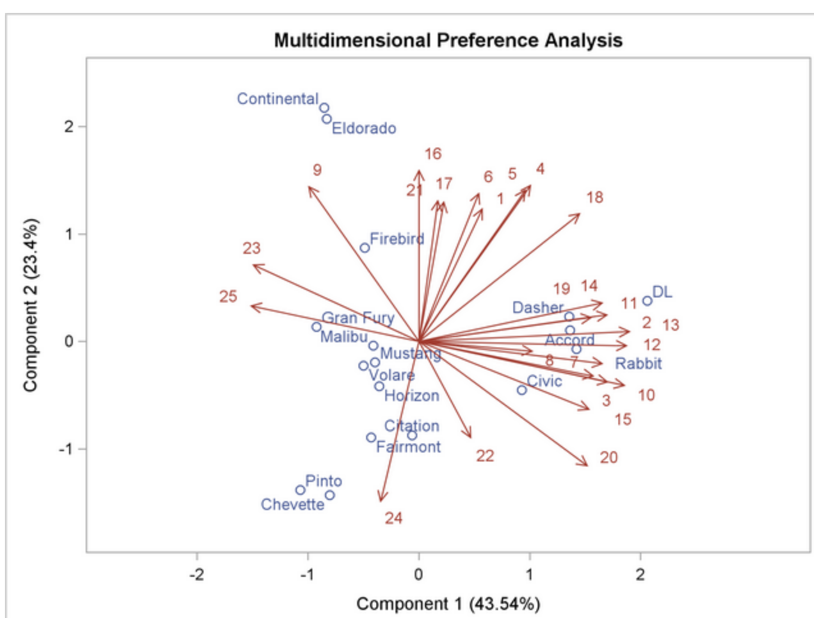


Figure 12.2 Sample Plot from PROC LOGISTIC

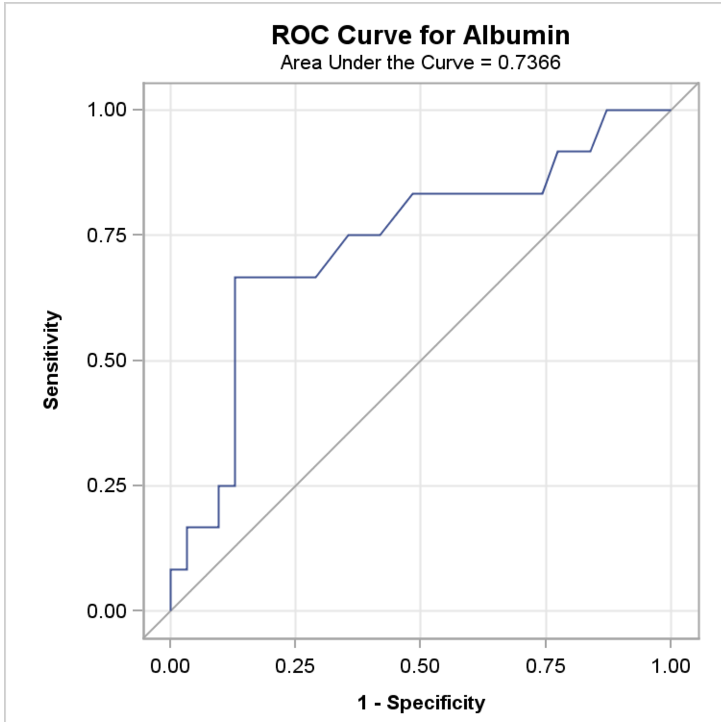
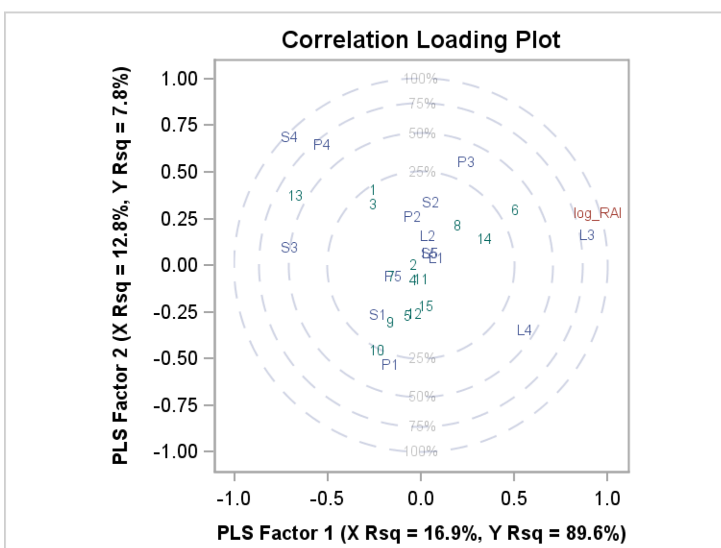


Figure 12.3 Sample Plot from PROC PLS

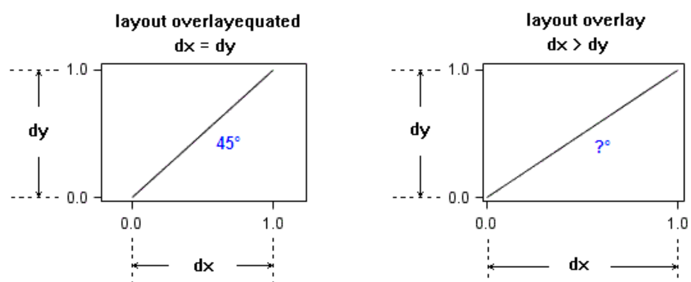


Whenever the same units of measure are used on both axes, it is desirable that the distance displayed between the same data interval be the same on both axes. To achieve this effect, you must use an OVERLAYEQUATED layout.

For specifying plot statements, the OVERLAYEQUATED layout is similar to the OVERLAY layout: you can specify one or more 2D plot statements within the layout block. However, OVERLAYEQUATED imposes a restriction on the plot axes and differs from OVERLAY in several ways. With OVERLAYEQUATED,

- Both X and Y axes are always numeric (TYPE=LINEAR). Thus, plot types that have discrete or binned axes (BOXPLOT, BOXPLOTPARM, BARCHARTPARM, HISTOGRAM, and HISTOGRAMPARM) cannot be used within this layout.
- For equal data intervals on both axes, the display distance is the same. For example, an interval of 2 units on the X axis maps to the same display distance as an interval of 2 units on the Y axis.
- The slope of a line in the display is the same as the slope in the data. In other words, a 45° slope in data will be represented by a 45° slope in the display. The EQUATETYPE= option offers different ways of presenting the data ranges while preserving the 45° display slope (see “Types of Equated Axes” on page 227).

The following figure illustrates how a series plot might be displayed when it is specified within an OVERLAYEQUATED layout rather than an OVERLAY layout:

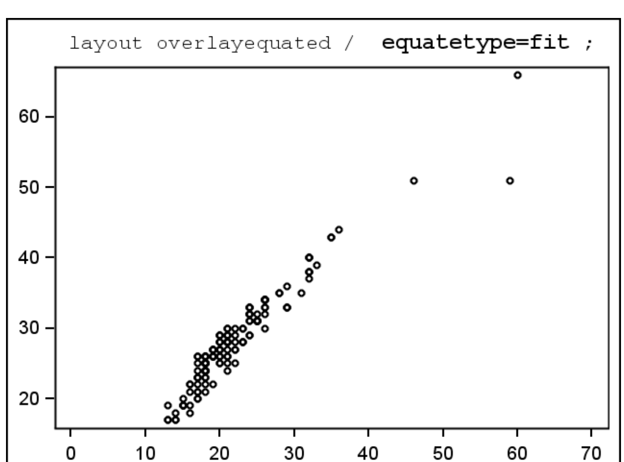


Basic Display Features of Equated Plots

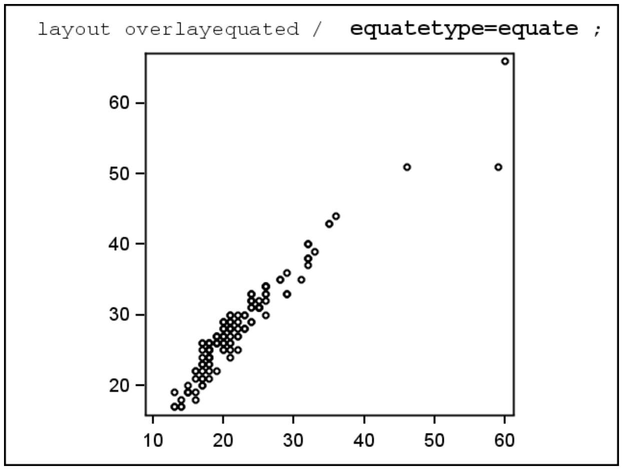
Types of Equated Axes

The EQUATETYPE= option of the LAYOUT OVELAYEQUATED statement manages the display of the axes. The following values are available:

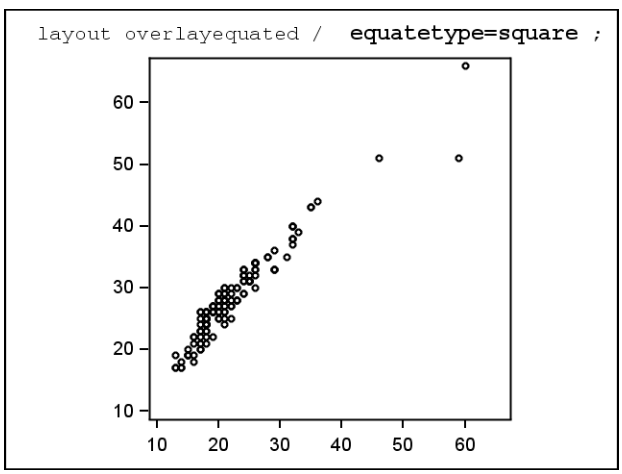
FIT
: X and Y axes have equal increments between tick values. The data ranges of both axes are compared to establish a common increment size. The axes can be of different lengths and have a different number of tick marks. Each axis represents its own data range. One axis can be extended to use available space in the plot area. This is the default.



EQUATE
: Same as FIT except that neither axis is extended to use available space in the plot area.



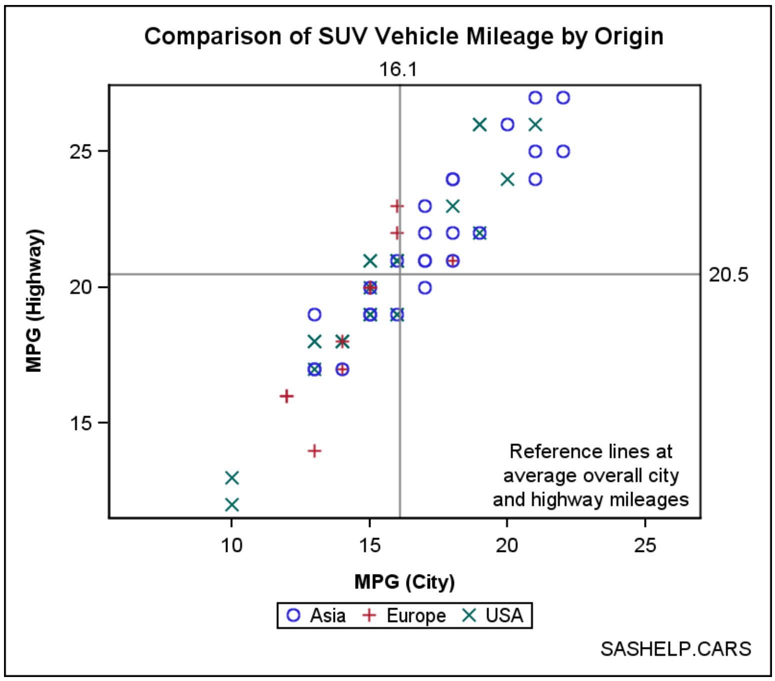
SQUARE
: Both the X and Y axes have the same length and the same tick values. The axis length and tick values are chosen so that the minimum and maximum of both X and Y appear in the range of values appearing on both axes.



The following example template uses the EQUATETYPE= option:

```
proc template;
  define statgraph mpg;
    mvar TYPE;
    begingraph;
      entrytitle "Comparison of " TYPE " Vehicle Mileage by Origin";
      entryfootnote halign=right "SASHELP.CARS";
      layout overlayequated / equatetype=fit;
        scatterplot x=mpg_city y=mpg_highway / group=origin
          name="s" markerattrs=(size=7px);
        referenceline x=eval(mean(mpg_city)) /
          curvelabel=eval(put(mean(mpg_city),4.1));
        referenceline y=eval(mean(mpg_highway)) /
          curvelabel=eval(put(mean(mpg_highway),4.1));
        discretelegend "s";
        layout gridded / columns=1 halign=right valign=bottom;
          entry "Reference lines at";
          entry "average overall city";
```

```
          entry "and highway mileages";
        endlayout;
      endlayout;
    endgraph;
  end;
run;

%let type=SUV;
proc sgrender data=sashelp.cars template=mpg;
  where type="&type";
run;
```



Note: This program uses several features, such as runtime macro variable resolution, EVAL expressions, and insets. All of these features are discussed in detail in other chapters.

Defining Axes for Equated Layouts

Axes for the OVERLAYEQUATED layout are similar to axes for the OVERLAY layout with the following exceptions:

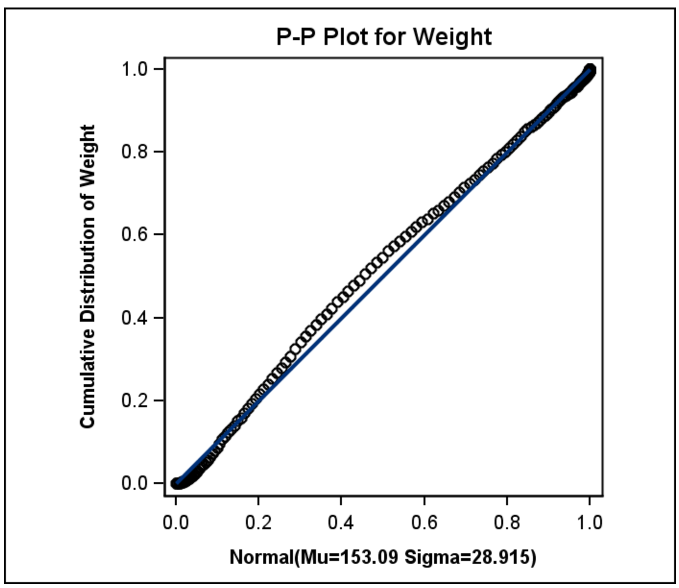
- Both axes are always of TYPE=LINEAR.
- Some axis options that always apply to both axes are specified in a COMMONAXISOPTS= option. Some of the supported options are INTEGER, TICKVALUELIST, TICKVALUESEQUENCE, VIEWMAX, and VIEWMIN.
- XAXISOPTS= and YAXISOPTS= options are supported (with a different set of sub-options from those of OVERLAY), but X2AXISOPTS= and Y2AXISOPTS= options are not supported. Some of the supported options are DISPLAY, LABEL, GRIDDISPLAY, DISPLAYSECONDARY, OFFSETMAX, OFFSETMIN, THRESHOLDMAX, THRESHOLDMIN, and TICKVALUEFORMAT.
- No independent secondary (X2, Y2) axes are available, although secondary axes that mirror the primary axes can be displayed.

Chapter 5, “Managing Axes in an OVERLAY Layout,” on page 53 discusses many of the axis options that are available for managing graph axes.

To illustrate how to control axes for the equated layout, we will look at a simplified version of the PPPLOT template that is supplied with PROC UNIVARIATE, which is delivered with Base SAS. The following code shows a SAS program that can be used to run PROC UNIVARIATE:

```
ods graphics on;

proc univariate data=sashelp.heart;
  var weight;
  ppplot / normal square;
run;
quit;
```

When the code is run, it creates the following plot. The plot uses the PPPLOT template, which is stored in the BASE.UNIVARIATE.GRAPHICS folder of the SASHELP.TMPLMST item store:



In PROC UNIVARIATE, the PPPLOT statement creates a probability-probability plot (also referred to as a P-P plot or percent plot), which compares the empirical cumulative distribution function (ecdf) of a variable with a specified theoretical cumulative distribution function such as the normal. If the two distributions match, the points on the plot form a linear pattern that passes through the origin and has unit slope. Thus, you can use a P-P plot to determine how well a theoretical distribution models a set of measurements.

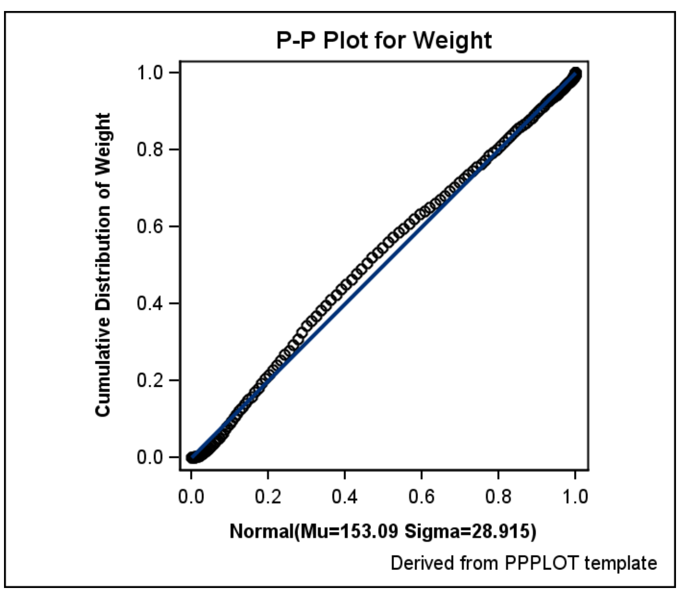
The supplied PPPLOT template uses several dynamics to pass in values for options, but in essence, the following template is equivalent. The dynamics for the title and axis labels have been converted into literals appropriate for this set of data.

```
proc template;
define statgraph pp_plot;
  begingraph;
    entrytitle "P-P Plot for Weight";
    entryfootnote halign=right "Derived from PPPLOT template";

    layout overlayequated / equatetype=square
           xaxisopts=(label="Normal(Mu=153.09 Sigma=28.915)"
```

```
                    thresholdmin=1 thresholdmax=1)
          yaxisopts=(label="Cumulative Distribution of Weight"
                    thresholdmin=1 thresholdmax=1)
          commonaxisopts=(viewmin=0.0 viewmax=1.0) ;

      scatterplot x=Theoretical y=Empirical;
      lineparm x=0 y=0 slope=1 / lineattrs=GraphFit;
    endlayout;

  endgraph;
end;
run;
```

This simplified template produces a similar plot if it is rendered with the same data as the UNIVARIATE plot. An ODS OUTPUT statement can convert the output object from UNIVARIATE into a SAS data set:

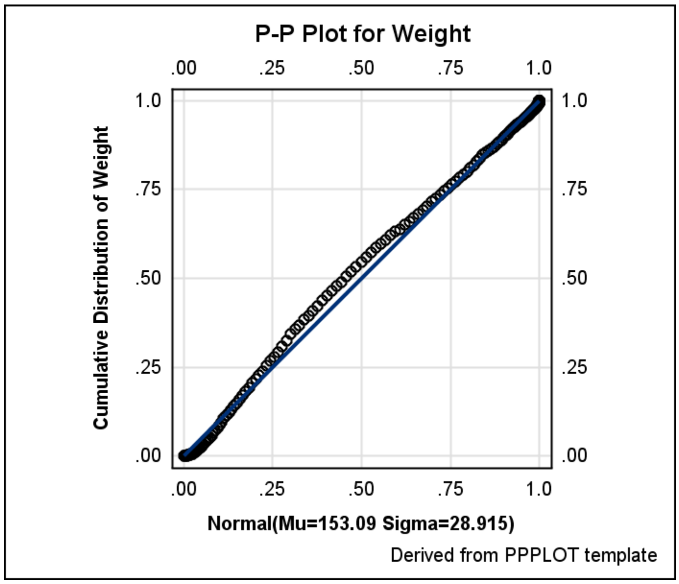
```
ods graphics on;
ods select ppplot;
ods output ppplot=ppdata;
proc univariate data=sashelp.heart;
  var weight;
  ppplot / normal square;
run;
quit;

proc sgrender data=ppdata
              template=pp_plot;
run;
```



The following template modifies the equated axes, as shown in the next graph:

```
layout overlayequated /  equatetype=square
   xaxisopts=(label="Normal(Mu=153.09 Sigma=28.915)"
              thresholdmin=1 thresholdmax=1
              tickvalueformat=3.2
              display=(label tickvalues)
              displaysecondary=(tickvalues)
```

```
                    griddisplay=on)
          yaxisopts=(label="Cumulative Distribution of Weight"
                    thresholdmin=1 thresholdmax=1
                    tickvalueformat=3.2
                    display=(label tickvalues)
                    displaysecondary=(tickvalues)
                    griddisplay=on)
          commonaxisopts=(viewmin=0.0 viewmax=1.0
                    tickvaluesequence=(start=0 end=1 increment=.25) );
```



Chapter 13
Using 3D Graphics

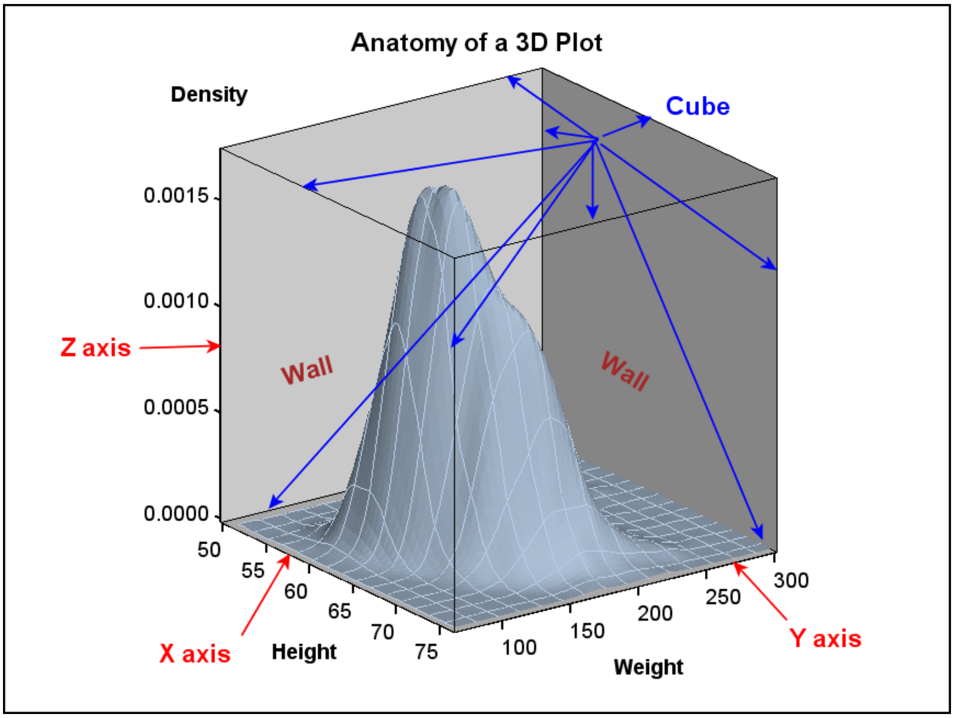
The LAYOUT OVERLAY3D Statement

GTL has one layout for 3D graphics: the LAYOUT OVERLAY3D statement. Two 3D plot statements can be placed within this layout: BIHISTO3DPARM and SURFACEPLOTPARM. No 2D plot statements can be used in this layout, although text statements such as ENTRY can be used.

Typical applications of OVERLAY3D layout are to create a 3D representation of a surface or a bi-variate histogram (possibly overlaid together). The 3D layout has features that 2D layouts do not have. For example, it can do each of the following:

- generate axes for three independent variables (X, Y,and Z)
- set a viewpoint of the graph (TILT=, ROTATE=, and ZOOM= options)
- display lines that represent the intersection of axis walls (CUBE= option).

The following figure shows the basic anatomy of a 3D graph:

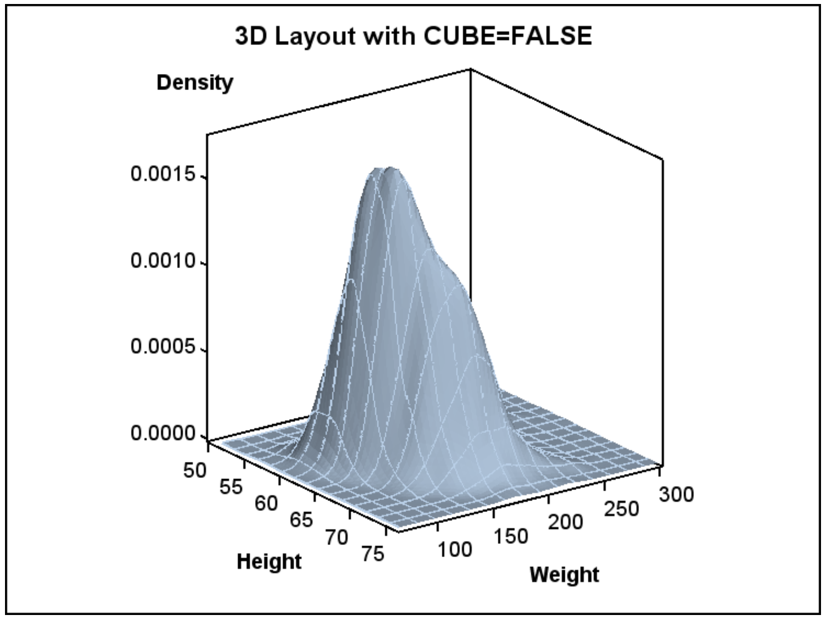


Basic Display Features of 3D Graphs

Managing the Display of Cube Lines

You can control whether the additional nine lines representing the intersection of all axis planes are displayed with the CUBE= option in the LAYOUT OVERLAY3D statement. The default is CUBE=TRUE.

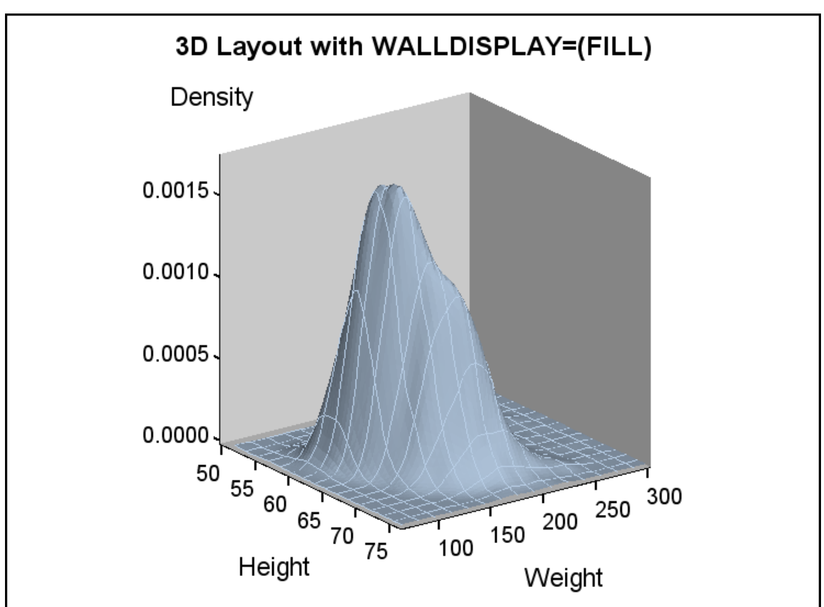
```
layout overlay3d / cube=false ;
  surfaceplotparm x=height y=weight
                  z=density;
endlayout;
```



Displaying a Fill in the Graph Walls

By default, only the outlines of the walls bounding the XY, XZ, and YZ axis planes are shown. You can display filled walls by including the WALLDISPLAY=(FILL) or WALLDISPLAY=(FILL OUTLINE) settings in the LAYOUT OVERLAY3D statement. You can change the wall color (when filled) with the WALLCOLOR=option. When filled, the wall lighting is adjusted to give a 3D effect, based on the graph viewpoint.

```
layout overlay3d / cube=false
    walldisplay=(fill) ;
  surfaceplotparm x=height y=weight
                  z=density;
endlayout;
```

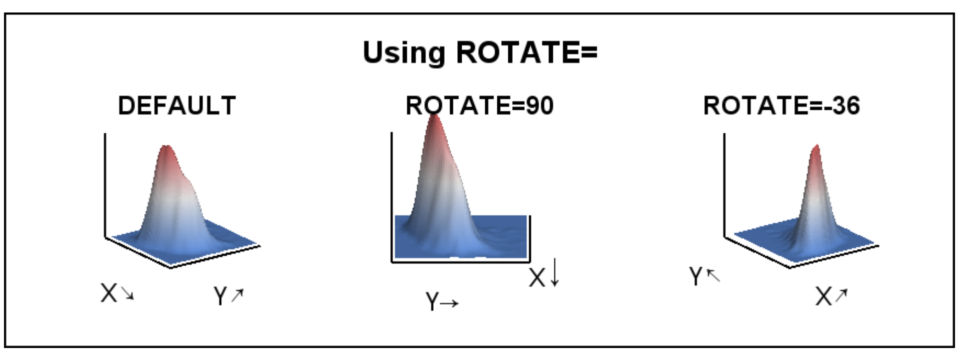


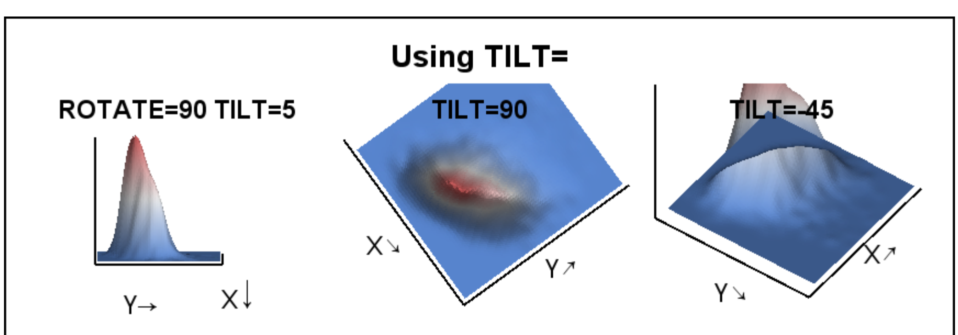
Defining a Viewpoint

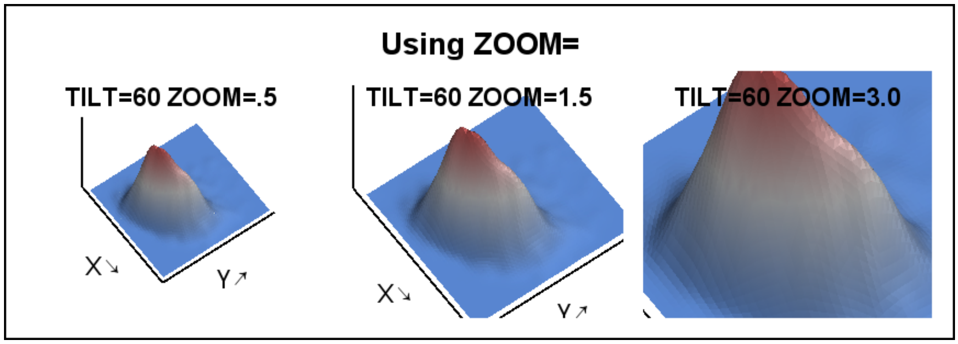
Representing a 3D graph statically in two dimensions often obscures details that are better viewed from a different viewpoint. Three options on the LAYOUT OVERLAY3D statement can be independently set to obtain a different viewpoint.

Option	Value Range	Default	Description
ROTATE=	-360 to 360	54	Specifies the angle of rotation. Rotation is measured in a clockwise direction about a virtual axis, parallel to the Z axis (vertical) and passing through the center of the bounding cube. A counterclockwise rotation can be specified with a negative value.
TILT=	-360 to 360	20	Specifies the angle of tilt in degrees. Tilt is measured in a clockwise direction about a virtual axis parallel to the X axis (vertical) and passing through the center of the bounding cube. A counterclockwise tilt can be specified with a negative value.
ZOOM=	> 0	1	Specifies a zoom factor. Factors greater than 1 move closer to the bounding cube (zoom in), less than 1 move farther away (zoom out).

These options can be used in combination with each other to obtain a desired perspective. The following figures show some examples. To generate the figures, a LATTICE layout was used to "grid" a series of OVERLAY3D layouts of the same plot with different viewpoints. The arrows on the X and Y axes indicate increasing X and Y values.







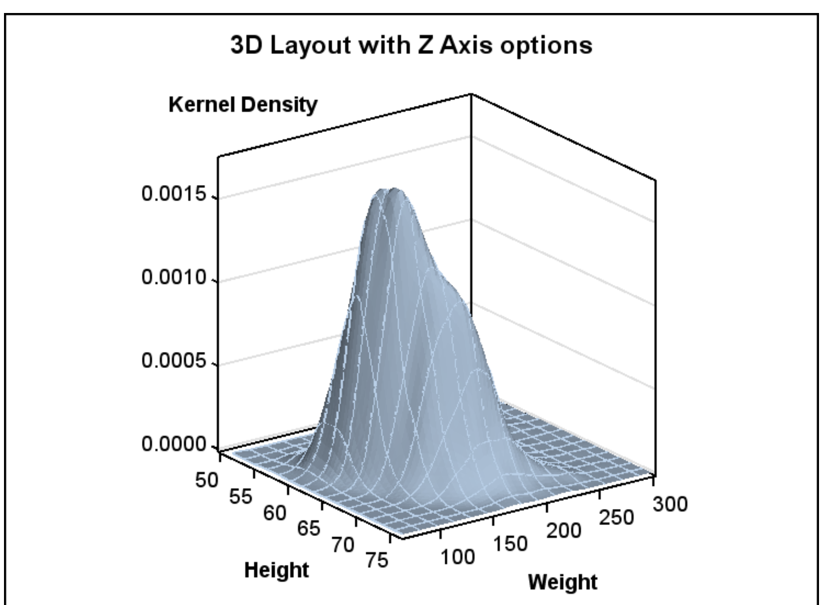
Defining Axes

Axes for the OVERLAY3D layout are similar to axes for the OVERLAY layout, although the following exceptions apply to OVERLAY3D layouts:

- An additional ZAXISOPTS=( ) option is available for managing the Z axis.
- All three axis types can be either LINEAR, LOG, or TIME. A DISCRETE axis is not supported on OVERLAY3D layouts.
- No secondary (X2, Y2, Z2) axes are available on OVERLAY3D layouts.
- Axis tick values are automatically thinned. No other fitting policy for OVERLAY3D layout is available.
- For any axis, the location of the displayed axis features (line, ticks, tick values, and label) might shift, based on the specified viewpoint.

The following layout specification displays grid lines and a label for the Z axis:

```
layout overlay3d / cube=false
    zaxisopts=(griddisplay=on
    label="Kernel Density") ;
  surfaceplotparm x=height y=weight
                  z=density;
endlayout;
```



Data Requirements for 3D Plots

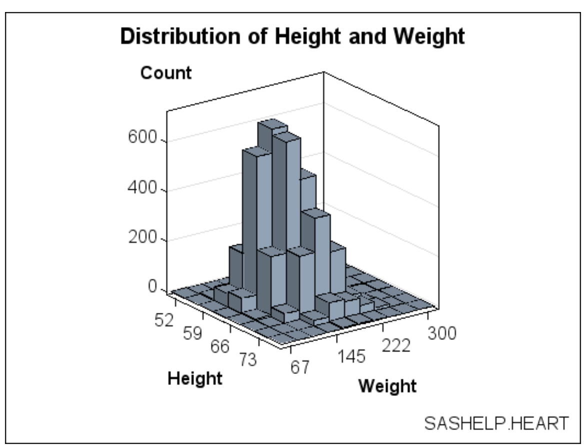
Overview

Both of the plot statements that can be used in the OVERLAY3D layout are parameterized plots (see “Plot Statements—Terminology and Concepts” on page 21). This means that the input data must conform to certain prerequisites in order for the plot to be drawn.

Parameterized plots do not perform any internal data transformations or computing for you. So, in most cases, you will need to perform some kind of preliminary data manipulation to set up the input data correctly before executing the template. The types of data transformations that you need to perform are commonly known as "binning" and "gridding."

Producing Bivariate Histograms

A bivariate histogram shows the distribution of data for two continuous numeric variables. In the following graph, the X axis displays HEIGHT values and the Y axis displays WEIGHT values. The Z axis represents the frequency count of observations. The Z values could be some other measure (for example, percentage of observations), but they can never be negative.



As with a standard histogram, the X and Y variables in the bivariate histogram have been uniformly binned, which means that their data ranges have been divided into equal sized intervals (bins), and that observations are distributed into one of these bin combinations.

The BIHISTOGRAM3DPARM statement, which produced this plot, does not perform any binning computation on the input columns. Thus, you must pre-bin the data. In the following example, the binning is done with PROC KDE (part of the SAS/STAT product).

```
proc kde data=sashelp.heart;
  bivar height(ngrid=8) weight(ngrid=10) /
        out=kde(keep=value1 value2 count) noprint plots=none;
run;
```

In this program, the NGRID= option sets the number of bins to create for each variable. The default for NGRID is 60. The binned values for HEIGHT are stored in VALUE1, and the binned values for WEIGHT are stored in VALUE2. This selection of bins produces 1 observation for each of the 80 bin combinations. Frequency counts for each bin combination are placed in a COUNT variable in the output data set.

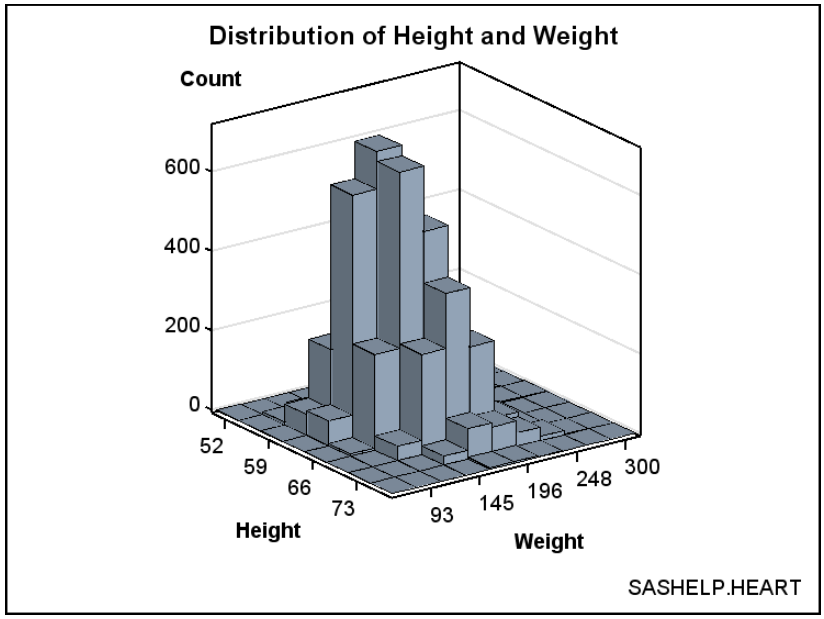
FSVIEW: WORK.KDE (B)

Obs	value1	value2	count
1	51.5	67	1
2	51.5	92.888888889	0
3	51.5	118.77777778	0
4	51.5	144.66666667	0
5	51.5	170.55555556	0
6	51.5	196.44444444	0
7	51.5	222.33333333	0
8	51.5	248.22222222	0
9	51.5	274.11111111	0
10	51.5	300	0
11	55.071428571	67	1
12	55.071428571	92.888888889	8
13	55.071428571	118.77777778	16
14	55.071428571	144.66666667	11
15	55.071428571	170.55555556	1
16	55.071428571	196.44444444	1

Notice that when you form the grid by choosing the number of bins, the bin widths (about 3.5 for HEIGHT and about 26 for WEIGHT) are most often non-integer.

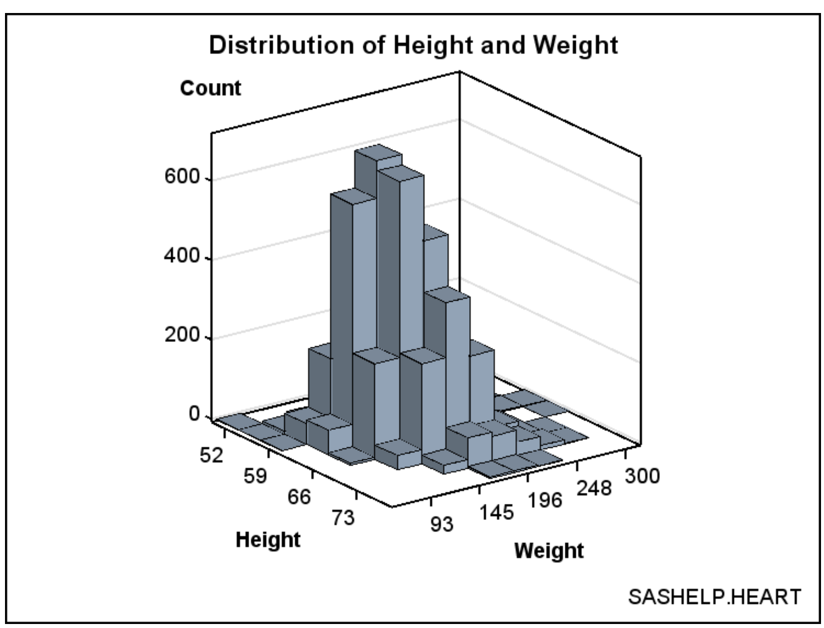
The following template definition displays this data. By default, the BINAXIS=TRUE setting requests that X and Y axes show tick values at bin boundaries. Also by default, XVALUES=MIDPOINTS and YVALUES=MIDPOINTS, which means that the X and Y columns represent midpoint values rather than lower bin boundaries (LEFTPOINTS) or upper bin boundaries (RIGHTPOINTS). Not all of the bins in this graph can be labelled without collision because the graph is small. Thus, the ticks and tick values were thinned. The non-integer bin values are converted to integers (TICKVALUEFORMAT=5.) to simplify the axis tick values. DISPLAY=ALL means "show outlined, filled bins."

```
proc template;
  define statgraph bihistogram1a;
    begingraph;
      entrytitle "Distribution of Height and Weight";
      entryfootnote halign=right "SASHELP.HEART";
      layout overlay3d / cube=false zaxisopts=(griddisplay=on)
          xaxisopts=(linearopts=(tickvalueformat=5.))
          yaxisopts=(linearopts=(tickvalueformat=5.));
        bihistogram3dparm x=value1 y=value2 z=count /
          display=all;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data= kde template=bihistogram1a;
  label value1="Height" value2="Weight";
run;
```



Eliminating Bins that Have No Data. Notice that the bins of 0 frequency (there are several) are included in the plot. If you want to eliminate the bins where there is no data, you can generate a subset of the data. The subset makes it a bit clearer where there are bins with small frequency counts verses portions of the grid with no data.

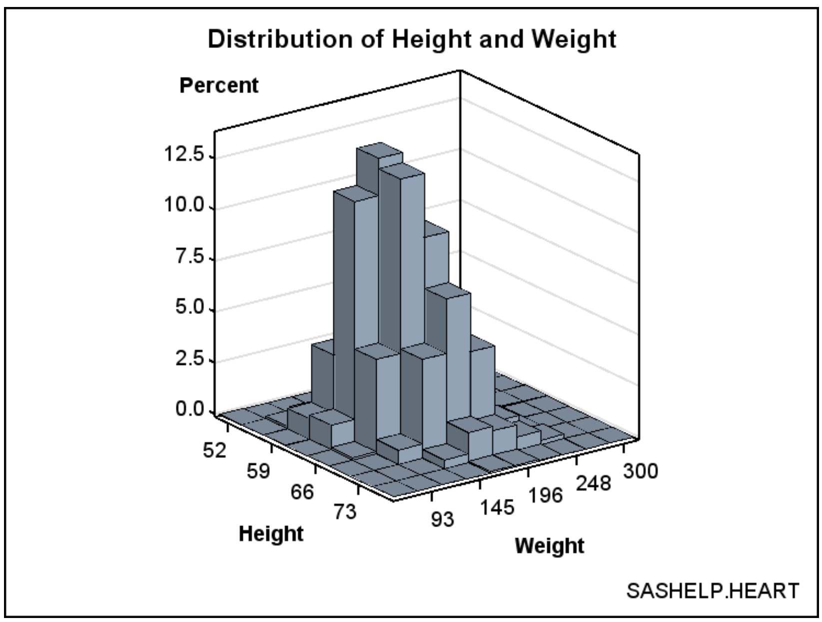
```
proc sgrender data= kde template=bihistogram1a;
  where count > 0;
  label value1="Height" value2="Weight";
run;
```



Displaying Percentages on Z Axis. To display the percentage of observations on the Z axis instead of the actual count, you need to perform an additional data transformation to convert the counts to percentages.

```
proc kde data=sashelp.heart;
  bivar height(ngrid=8) weight(ngrid=10) /
```

```
        out=kde(keep=value1 value2 count) noprint plots=none;
run;

data kde;
  if _n_ = 1 then do i=1 to rows;
    set kde(keep=count) point=i nobs=rows;
    TotalObs+count;
  end;
  set kde;
  Count=100*(Count/TotalObs);
  label Count="Percent";
run;

proc sgrender data= kde template=bihistogram1a;
   label value1="Height" value2="Weight";
run;
```



Setting Bin Width. Another technique for binning data is to set a bin width and compute the number of observations in each bin. In the DATA step below, 5 is the bin width for HEIGHT and 25 for WEIGHT. With this technique you do not know the exact number of bins, but you can assure that the bins are of a "good" size.

```
data heart;
  set sashelp.heart(keep=height weight);
  if height ne . and weight ne .;
  height=round(height,5);
  weight=round(weight,25);
run;
```

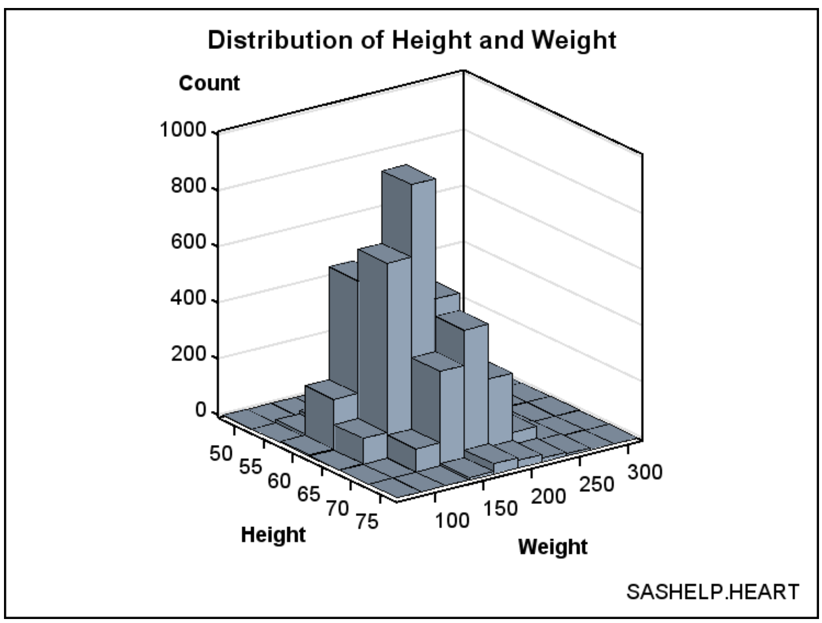
After rounding, HEIGHT and WEIGHT can be used as classifiers for a summarization. Notice that the COMPLETETYPES option forces all possible combinations of the two variables to be output, even if no data exists for a particular crossing.

```
proc summary data=heart nway completetypes;
  class height weight;
  var height;
```

```
  output out=stats(keep=height weight count) N=Count;
run;
```

The template can be simplified because we know that the bin midpoints are uniformly spaced integers. For this selection of bin widths, 6 bins were produced for HEIGHT and 10 for WEIGHT.

```
proc template;
  define statgraph bihistogram2a;
    begingraph;
      entrytitle "Distribution of Height and Weight";
      entryfootnote halign=right "SASHELP.HEART";
      layout overlay3d / cube=false zaxisopts=(griddisplay=on);
        bihistogram3dparm x=height y=weight z=count /
          display=all;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=stats template=bihistogram2a;
run;
```

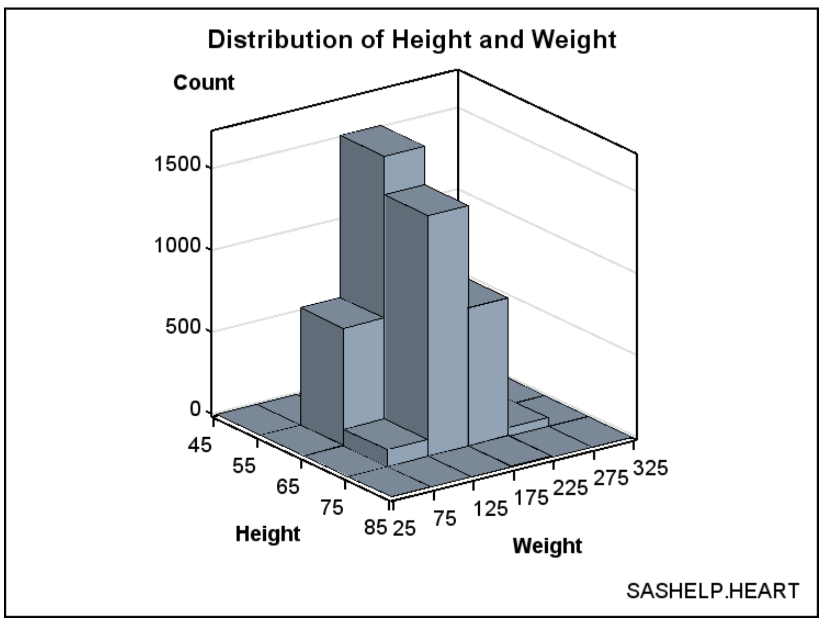


If you prefer to see the axes labeled with the bin endpoints rather the bin midpoints, you can use the ENDLABELS=TRUE setting on the BIHISTOGRAM3DPARM statement. Note that the ENDLABELS= option is independent of the XVALUES= and YVALUES= options.

In the following example, the bin widths are changed to even numbers (10 and 50) to make the bin endpoints even numbers:

```
proc template;
  define statgraph bihistogram2a;
    begingraph;
      entrytitle "Distribution of Height and Weight";
      entryfootnote halign=right "SASHELP.HEART";
      layout overlay3d / cube=false zaxisopts=(griddisplay=on);
        bihistogram3dparm x=height y=weight z=count /
```

```
         binaxis=true endlabels=true display=all;
       endlayout;
     endgraph;
   end;
 run;
 data heart;
   set sashelp.heart(keep=height weight);
   height=round(height,10);
   weight=round(weight,50);
 run;
 proc summary data=heart nway completetypes;
   class height weight;
   var height;
   output out=stats(keep=height weight count) N=Count;
 run;

 proc sgrender data=stats template=bihistogram2a;
 run;
```



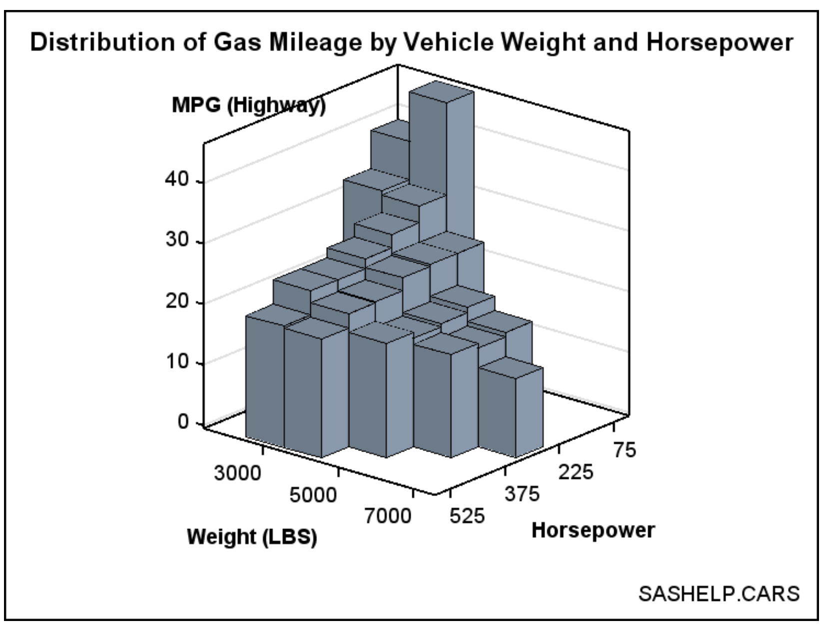
If you choose bin widths that are too small, "gaps" might be displayed among axis ticks values, which might cause the following message:

```
WARNING: The data for a HISTOGRAMPARM statement is not appropriate.
         HISTOGRAMPARM statement expects uniformly-binned data. The
         histogram might not be drawn correctly.
```

Because BIHISTOGRAM3DPARM is a parameterized plot, you can use it to show the 3D data summarization of a response variable Z, which must have non-negative values, by two numeric classification variables that are uniformly spaced (X and Y). That is, even though the graphical representation is a bivariate histogram, the Z axis does not have to display a frequency count or a percent.

```
 data cars;
   set sashelp.cars(keep=weight horsepower mpg_highway);
   if horsepower ne . and weight ne .;
   horsepower=round(horsepower,75);
   weight=round(weight,1000);
```

```
run;

proc summary data=cars nway completetypes;
  class weight horsepower;
  var mpg_highway;
  output out=stats mean=Mean ;
run;

proc template;
  define statgraph bihistogram2b;
    begingraph;
      entrytitle
       "Distribution of Gas Mileage by Vehicle Weight and Horsepower";
      entryfootnote halign=right "SASHELP.CARS";
      layout overlay3d / cube=false zaxisopts=(griddisplay=on) rotate=130;
        bihistogram3dparm y=weight x=horsepower z=mean / binaxis=true
          display=all;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=stats template=bihistogram2b;
run;
```



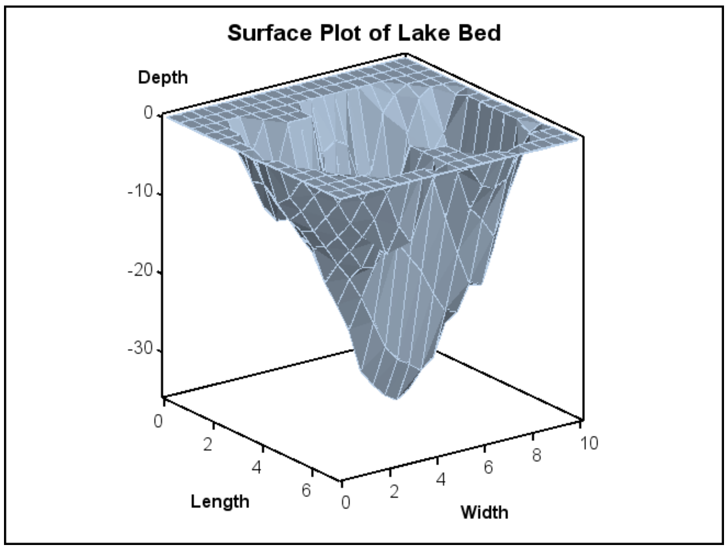
Producing Surface Plots

A surface plot shows points that are defined by three continuous numeric variables and connected with a polygon mesh. A polygon mesh is a collection of vertices, edges, and faces that defines the shape of a polyhedral object, which simulates the surface. For a surface to be drawn, the input data must be "gridded"; that is, the X and Y data ranges are split into uniform intervals (the grid), and the corresponding Z values are computed for each X,Y pair. Smaller data grid intervals produce a smoother surface because more smaller polygons are used but are more resource intensive because of the large number of polygons that are generated. Larger data grid intervals produce a coarser, faceted surface because the polygon mesh has fewer faces and is less resource intensive.

The faces of the polygons can be filled, and lighting is applied to the polygon mesh to create the 3D effect. It is possible to superimpose a grid on the surface. The grid display is a sampling of the data grid boundaries that intersect the surface. The grid display can be thought of as a simpler see-through line version of the surface and can be rendered with or without displaying the filled surface.

The default appearance of a surface is a filled polygon mesh with superimposed grid lines.

```
surfaceplotparm x=length y=width z=depth;
```



The SURFACEPLOTPARM statement assumes that the response/Z values have been provided for a uniform X-Y grid. Missing Z values will leave a "hole" in the surface.

The observations in the input data set should form an evenly spaced grid of horizontal (X and Y) values and one vertical (Z) value for each of these combinations. The observations should be in sorted order of Y and X to obtain an accurate graph.

In the following example, 315 observations in SASHELP.LAKE are gridded into a 15 by 21 grid. The length of the grid is from 0 to 7 by .5, and the width of the grid is from 0 to 10 by .5 There are no missing Depth values.

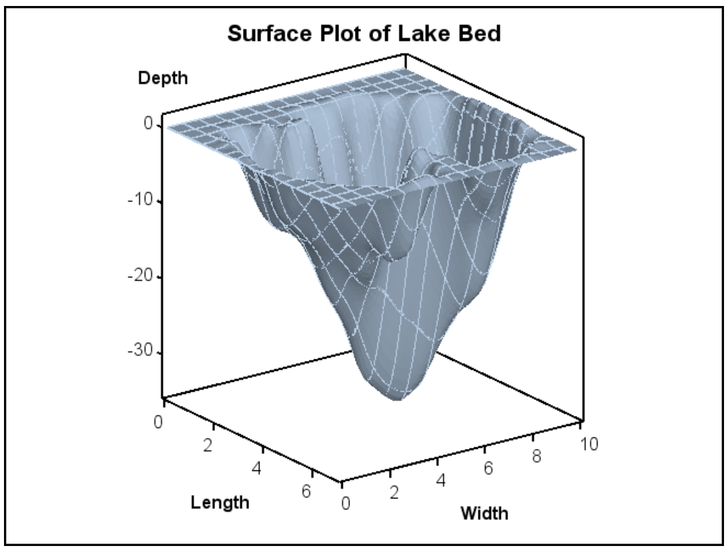
FSVIEW: SASHELP.LAKE (B)

Obs	Width	Length	Depth
1	0	0	0
2	0	0.5	0
3	0	1	0
4	0	1.5	-0.005977326
5	0	2	0
6	0	2.5	0
7	0	3	0
8	0	3.5	-0.212870131
9	0	4	0
10	0	4.5	0
11	0	5	0
12	0	5.5	-0.002988663
13	0	6	0
14	0	6.5	0
15	0	7	0
16	0.5	0	0
17	0.5	0.5	0
18	0.5	1	0
19	0.5	1.5	-0.027228609

Input data with non-gridded columns should be preprocessed with PROC G3GRID. This procedure creates an output data set, and it allows specification of the grid size and various methods for computed interpolated Z column(s). For further details, see the documentation for PROC G3GRID in the *SAS/GRAPH Reference*.

Using PROC G3GRID, the following code performs a Spline interpolation and generates a surface plot. By increasing the grid size and specifying a SPLINE interpolation, a smoother surface is rendered.

```
proc g3grid data=sashelp.lake out=spline;
  grid width*length = depth / naxis1=75 naxis2=75 spline;
run;

proc sgrender data=spline template=surfaceplotparm;
run;
```

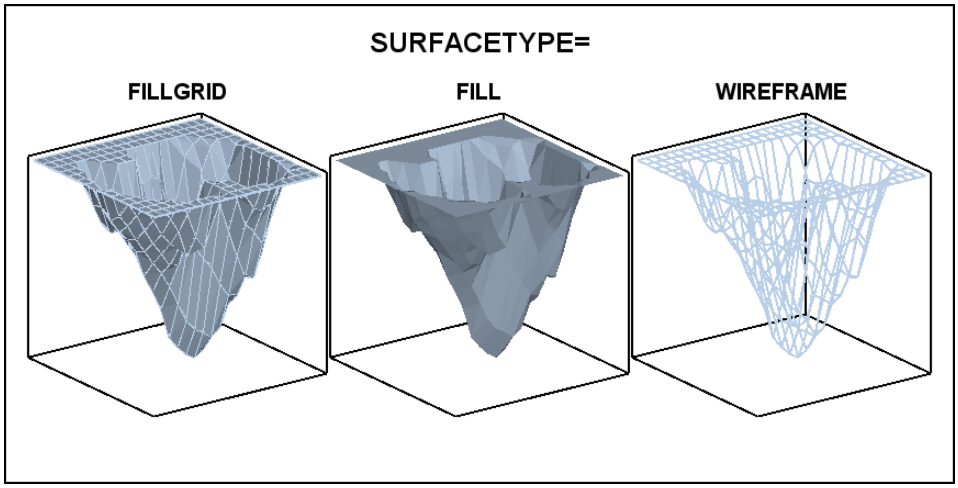


The SURFACETYPE= option offers three different types of surface rendering:

FILLGRID
: a filled surface with grid outlines (the default)

FILL
: a filled surface without grid outlines

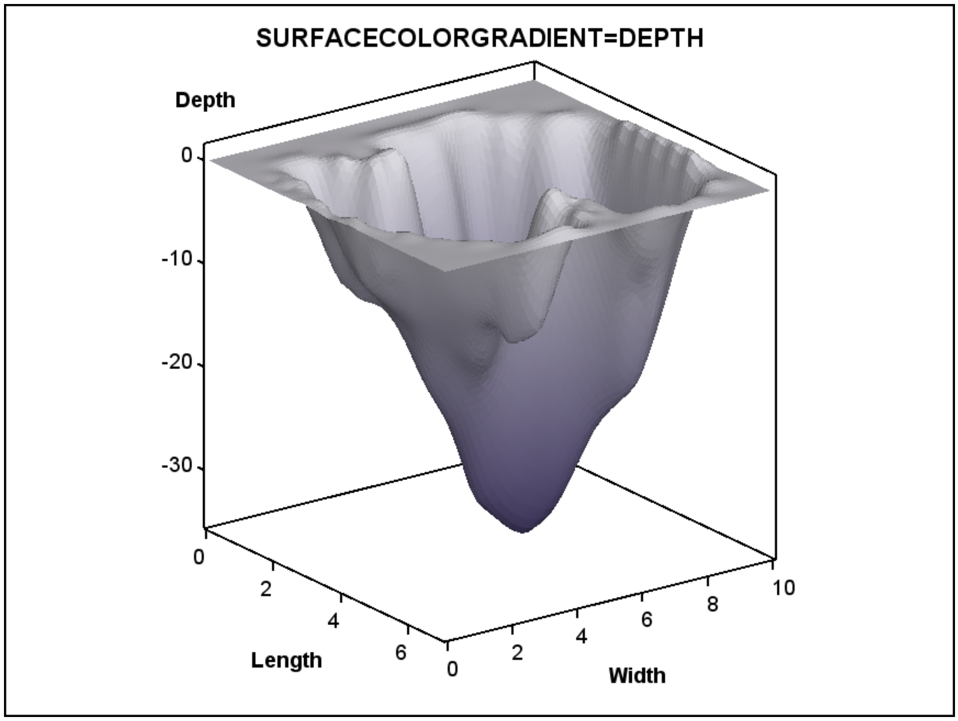
WIREFRAME
: an unfilled (see through) surface with grid outlines



Adding a Color Gradient. The surface can be colored with a gradient that is based on a response variable by setting a column on the SURFACECOLORGRADIENT= option. The following example uses the DEPTH variable:

```
proc template;
  define statgraph surfaceplotparm;
    begingraph;
      entrytitle "SURFACECOLORGRADIENT=DEPTH";
      layout overlay3d / cube=false;
      surfaceplotparm x=length y=width z=depth /
        surfacetype=fill
        surfacecolorgradient=depth
        colormodel=twocolorramp
        reversecolormodel=true ;
      endlayout;
    endgraph;
  end;
run;

/* create gridded data for surface */
proc g3grid data=sashelp.lake out=spline;
  grid width*length = depth / naxis1=75 naxis2=75 spline;
run;

proc sgrender data=spline template=surfaceplotparm;
run;
```

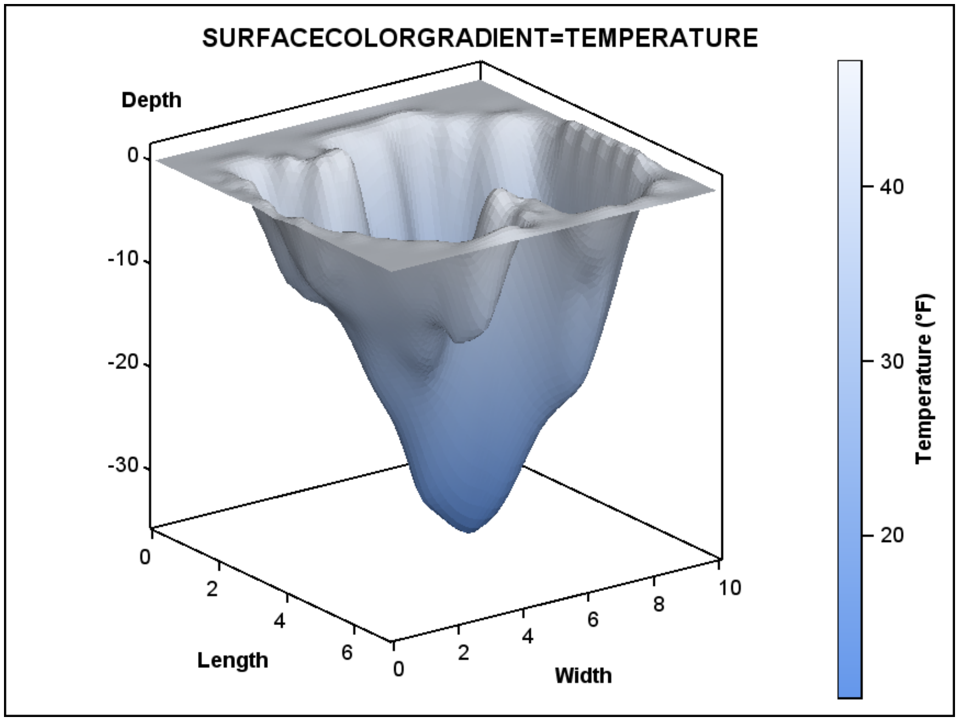
The COLORMODEL=TWOCOLORRAMP setting indicates a style element. Four possible color ramps are supplied in every style. The REVERSECOLORMODEL=TRUE setting exchanges (reverses) the start color and end color that is defined by the color model. The colors were reversed so that the darker color maps to the lower depths.



Using Color to Show an Additional Response Variable. The SURFACECOLORGRADIENT= option does not have to use the Z= variable. In the next example, another variable, TEMPERATURE is used. Notice that it is possible to display a continuous legend when you use the SURFACECOLORGRADIENT= option. Several legend options can be used. Using other color ramps and continuous legends are discussed in more detail in Chapter 8, "Adding Legends to a Graph," on page 117.

```
proc template;
  define statgraph surfaceplot;
   begingraph;
     entrytitle "SURFACECOLORGRADIENT=TEMPERATURE";
     layout overlay3d / cube=false;
       surfaceplotparm x=length y=width z=depth / name="surf"
           surfacetype=fill
           surfacecolorgradient=temperature
           reversecolormodel=true
           colormodel=twocoloraltramp ;
       continuouslegend "surf" /
           title="Temperature ((*ESC*){unicode '00B0'x}F)" ;
     endlayout;
   endgraph;
  end;
run;

data lake;
  set sashelp.lake;
  if depth = 0 then Temperature=46;
  else Temperature=46+depth;
run;

/* create gridded data for surface */
proc g3grid data=lake out=spline;
   grid width*length = depth temperature / naxis1=75 naxis2=75 spline;
```

```
run;

proc sgrender data=spline template=surfaceplot;
run;
```



Chapter 14
Using Dynamics and Macro Variables to Make Flexible Templates

Introduction to Dynamics and Macro Variables

If all of the variable names and options that are referenced in GTL templates had to be "hard coded" in the compiled template, it would require that you redefine and recompile the template every time you created the same type of graph with different variables. SAS programmers are familiar with using SAS macros and macro variables to build application code in which variables and other parameters can be specified by a calling program. The same techniques, as well as other techniques unique to ODS templates, can be applied in GTL to create reusable templates.

Note:

Declaring Dynamics and Macro Variables

Within the scope of a template definition, GTL supports the DYNAMIC statement for declaring dynamic variables, and the MVAR and NMVAR statements for declaring macro variables. These statements must appear after the DEFINE statement and before the BEGINGRAPH block. The following syntax shows the overall template structure:

PROC TEMPLATE;

DEFINE STATGRAPH *template-name*;

DYNAMIC *variable-1* <"*text-1*"> <*…variable-n*<"*text-n*">>;

MVAR *variable-1* <"*text-1*"> <*…variable-n*<"*text-n*">>;

NMVAR *variable-1* <"*text-1*"> <*…variable-n*<"*text-n*">>;

BEGINGRAPH;

GTL statements;

ENDGRAPH;

END;

RUN;

The difference between the MVAR and NMVAR declaration of macro variables is that NMVAR always converts the supplied value to a numeric token (like the SYMGETN function of the DATA step). Macro variables that are defined by MVAR resolve to strings (like the SYMGET function of the DATA step).

Each of the DYNAMIC, MVAR, and NMVAR statements can define multiple variables and an optional text string that denotes its purpose or usage:

```
dynamic YVAR "required" YLABEL "optional";
mvar LOCATE "can be INSIDE or OUTSIDE" SYSDATE;
nmvar TRANS "transparency factor";
```

Note: To make the template code more readable, it is helpful to adopt a naming convention for these variables to distinguish them from actual option values or column names. Common conventions include capitalization or adding leading or trailing underscores to their names. The examples in this document use capitalization to indicate a dynamic or macro variable.

Referencing Dynamics and Macro Variables

After dynamics and macro variables are declared, you can make one or more template references to them by simply using the name of the dynamic or macro variable in any valid context. These contexts include the following:

- as argument or option values:

```
seriesplot x=date y=YVAR / curvelabel=YLABEL
  curvelabellocation=LOCATE datatransparency=TRANS;
```

- as parts of concatenated text strings:

```
entrytitle "Time Series for " YLABEL;
entryfootnote "Created on " SYSDATE;
```

Dynamics and runtime macro variable references cannot be used in place of statement or option keywords, or in place of punctuation that is part of the syntax (parentheses, semicolons, and so on).

Note: If you precede a macro variable reference with an ampersand (&), the reference will be resolved when the template is compiled, not when it is executed.

For example, it is permissible to define TRANS as an MVAR for use in the following context:

```
proc template;
 define statgraph timeseries;
  dynamic YVAR YLABEL;
  mvar LOCATE TRANS;
  begingraph;
    layout overlay;
      seriesplot x=date y=YVAR  / curvelabel=YLABEL
        curvelabellocation= LOCATE datatransparency= TRANS ;
    endlayout;
  endgraph;
 end;
run;
```

This context is valid because an automatic, internal conversion using the BEST. format will be performed (with no warning messages).

Initializing Dynamics and Macro Variables

The main difference between dynamics and macro variables is that they are initialized differently.

For dynamics, use the DYNAMIC statement with PROC SGRENDER. Values for dynamics that resolve to column names or strings should be quoted. Numeric values should not be quoted:

```
proc sgrender data=financial template=timeseries;
  dynamic yvar="inflation" ylabel="Inflation Rate";
run;
```

For macro variables, use the current symbol table (local or global) to look up the macro variable values at runtime:

```
%let locate=inside;
%let trans=.3;
proc sgrender data=financial template=timeseries;
  dynamic yvar="inflation" ylabel="Inflation Rate";
run;
```

No initialization is needed for automatic macro variables like the system date and time value SYSDATE.

It is the responsibility of the person or process that initializes the dynamics or macro variables to ensure that the expected value type and value that is supplied is appropriate for the substitution context. If necessary, you can use conditional logic to evaluate the supplied values of dynamics or macro variables. Conditional logic is discussed in Chapter 15, “Using Conditional Logic and Expressions,” on page 259.

If a dynamic is used to supply a GTL option with a specific value and the supplied value is not valid or it is not initialized, then the option specification is ignored and the option's default value is used. For example, the HALIGN= option accepts the values RIGHT, CENTER, and LEFT. If the dynamic variable ALIGN is defined and then the template code specifies HALIGN=ALIGN, the ALIGN dynamic must be initialized with one of the values RIGHT, CENTER, or LEFT. If it is initialized with another value, TOP for example, the HALIGN= specification in the template is ignored, the default setting for HALIGN= is used, and you might see a warning in the SAS log.

If a dynamic is used to supply a required argument such as a column name, and the name is misspelled or not provided, then a warning is issued and that plot statement drops out of the final graph. A graph will still be produced, but it might be a blank graph, or it might show the results of all statements except those that are in error.

The following example shows how to create a generalized template that can be used to show the distribution of any numeric variable. The dynamic named VAR must be set, but the other dynamics are optional: BINS (sets the number of histogram bins) and FOOTNOTE. In the example, the DYNAMIC and MVAR variables are highlighted to emphasize where they are being used.

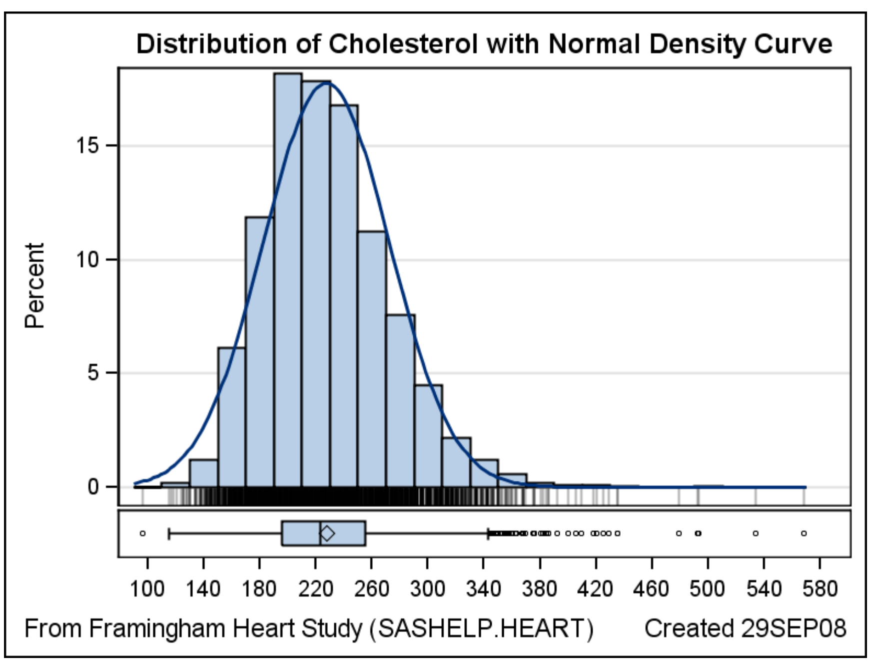
```
proc template;
 define statgraph distribution;
  dynamic VAR BINS FOOTNOTE ;
  mvar SYSDATE ;
```

```
    begingraph;
      entrytitle "Distribution of " VAR " with Normal Density Curve";
      entryfootnote halign=left FOOTNOTE halign=right "Created " SYSDATE ;
      layout lattice / rowweights=(.9 .1) columndatarange=union
                       rowgutter=2px;
        columnaxes;
          columnaxis / display=(ticks tickvalues);
        endcolumnaxes;
        layout overlay / yaxisopts=(offsetmin=.04 griddisplay=auto_on);
          histogram VAR / scale=percent nbins=BINS ;
          densityplot VAR / normal( ) name="Normal";
          fringeplot VAR / datatransparency=.7;
        endlayout;
        boxplot y=VAR / orient=horizontal primary=true boxwidth=.9;
      endlayout;
    endgraph;
   end;
  run;
```

The following execution of the template initializes the dynamic variables VAR and FOOTNOTE, but it does not initialize BIN:

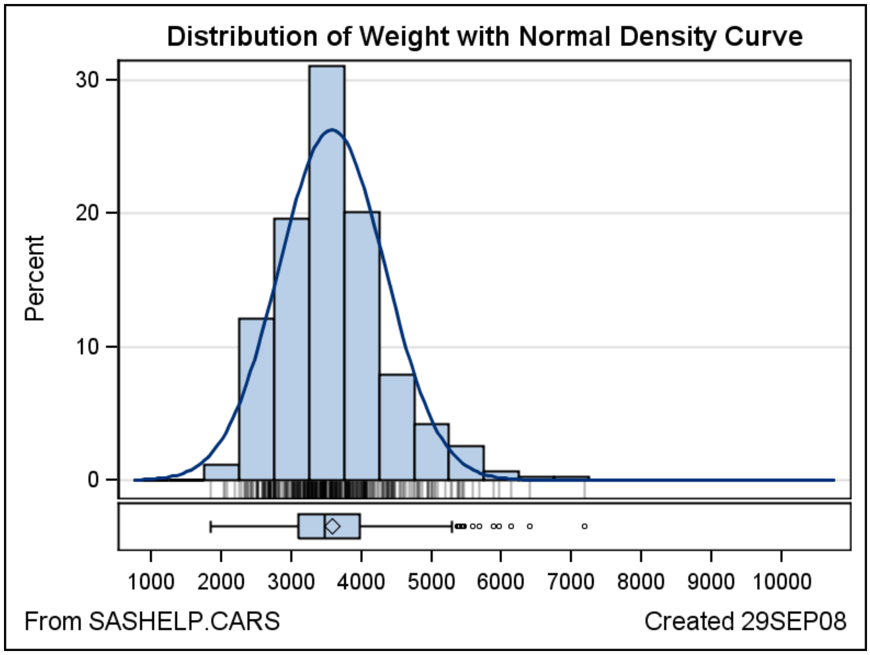
```
proc sgrender data=sashelp.heart template=distribution;
  dynamic var="Cholesterol"
  footnote="From Framingham Heart Study (SASHELP.HEART)";
run;
```

In this case, the template option **bins=BINS** drops out because the BINS dynamic has not been initialized.



This next execution of the template assigns values to each of the dynamics VAR, BIN, and FOOTNOTE, using different values from the previous example:

```
proc sgrender data=sashelp.cars template=distribution;
  dynamic var="Invoice" bins=20 footnote="From SASHELP.CARS";
run;
```



The next example shows a simplified version of the previous graph, this time adding an inset. The inset statistics are computed external to the template and passed into the template at runtime, using dynamics and macro variables. For more information on coding insets in graphs, see Chapter 16, “Adding Insets to a Graph,” on page 271.

```
proc template;
 define statgraph inset;
  dynamic VAR FOOTNOTE;
  mvar N MEAN STD;
  begingraph;
   entrytitle "Distribution of " VAR;
   entryfootnote halign=left FOOTNOTE;
   layout overlay / yaxisopts=(griddisplay=on);
    histogram VAR / scale=percent;
    layout gridded / columns=2
        autoalign=(topleft topright) border=true
        opaque=true backgroundcolor=GraphWalls:color;
      entry halign=left "N"; entry  halign=left N ;
      entry halign=left "Mean"; entry halign=left MEAN ;
      entry halign=left "Std Dev"; entry halign=left STD ;
    endlayout;
   endlayout;
  endgraph;
 end;
run;
```

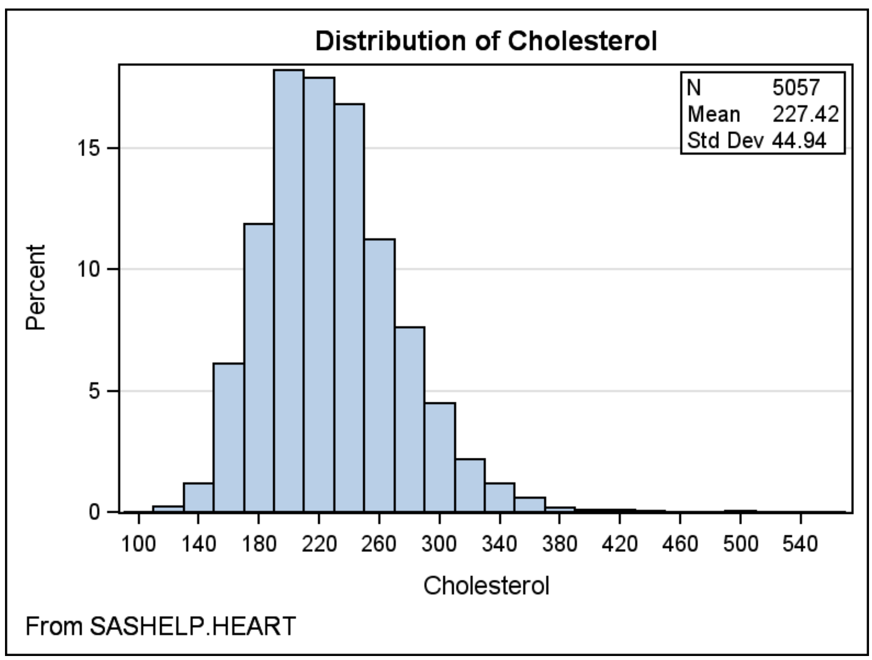
We will now define a macro that can pass values to this template. For a given numeric variable, the macro computes the number of observations, the mean, and the standard deviation, storing these statistics in macro variables N, MEAN, and STD. The macro variables are available to the SGRENDER step when the macro executes. Here is the definition for the macro, which we will name HIST:

```
%macro hist(dsn,numvar,footnote);
  /* these macro variables are declared in the template */
  %local N MEAN STD;
  proc sql noprint;
```

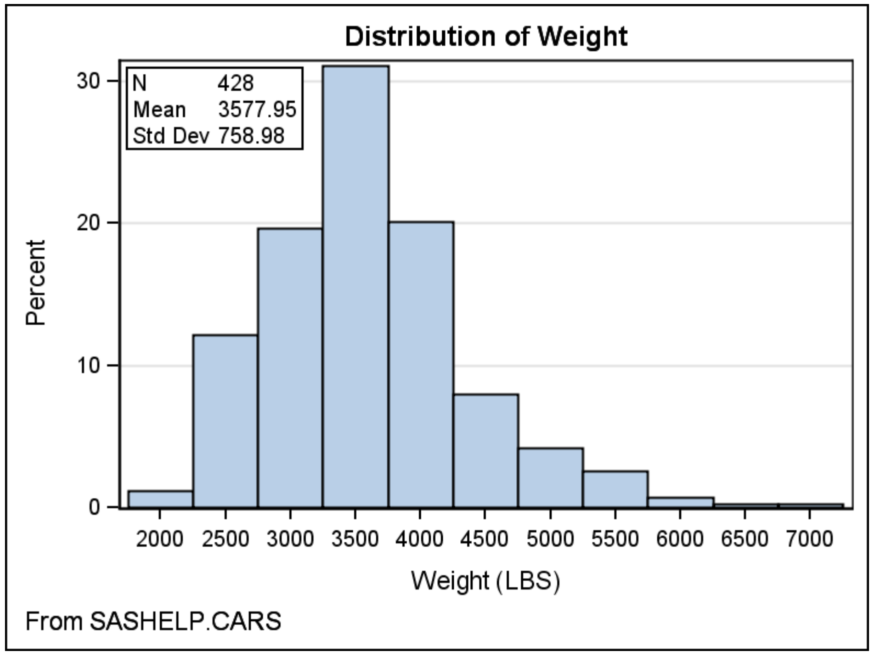
```
    select  put(n(&numvar),12. -L),
            put(mean(&numvar),12.2 -L),
            put(std(&numvar),12.2 -L) into :N, :MEAN, :STD
    from &dsn;
  quit;

  /* remove trailing blanks */
  %let N=&N; %let MEAN=&MEAN; %let STD=&STD;

  proc sgrender data=&dsn template=inset;
    dynamic VAR="&numvar" FOOTNOTE="&footnote";
  run;

%mend;
```

Here are results of two executions of the macro with different input data. Notice the placement of the inset might change on based on the amount of space that is available and the setting for the AUTOALIGN= option.

```
%hist(sashelp.heart, cholesterol, From SASHELP.HEART)
```




```
%hist(sashelp.cars, Weight, From SASHELP.CARS)
```



If you are familiar with the macro facility, you can create macros that validate the parameters before executing the template. It is also possible to validate the parameters within the compiled template, using the conditional logic syntax of GTL. For more information, see Chapter 15, “Using Conditional Logic and Expressions,” on page 259.

GTL supports user-defined computed expressions within compiled templates. This means that the inset statistics could have been computed directly within template, eliminating the need to pass them in with dynamics or macro variables. An example of how to do this is also discussed in Chapter 15, “Using Conditional Logic and Expressions,” on page 259.

For developers who would like to create a library of reusable templates, see the discussion on creating shared templates in “Creating Shared Templates” on page 339.

Chapter 15
Using Conditional Logic and Expressions

Constructs Available for Runtime Programming

GTL has several constructs that can take advantage of runtime programming:

- Dynamics and macro variables
- Expressions
- Conditional processing

This chapter discusses expressions and conditional processing. Dynamics and macro variables are discussed in Chapter 14, “Using Dynamics and Macro Variables to Make Flexible Templates,” on page 251.

Expressions

In GTL, as in Base SAS, an expression is an arithmetic or logical expression that consists of a sequence of operators, operands, and functions. An operand is a dynamic, a macro variable, a column, a function, or a constant. An operator is a symbol that requests a comparison, logical operation, arithmetic calculation, or character concatenation.

Expressions can be used to set an option value that is any one of the following:

- a constant (character or numeric)
- a column
- part of the text for ENTRYTITLE, ENTRYFOOTNOTE, and ENTRY statements.

In GTL, an expression must be enclosed in an EVAL function.

The following examples show how to specify an expression. This first example uses the MEAN function to compute several constants:

```
/* create reference lines at computed positions */
referenceline y=eval(mean(weight)+2*std(weight)) / curvelabel="+2 STD";
referenceline y=eval(mean(weight)) / curvelabel="Mean";
referenceline y=eval(mean(weight)-2*std(weight)) / curvelabel="-2 STD";
```

This next example creates a new column:

```
/* create a new column as a log transformation */
scatterplot x=date y=eval(log10(amount));
```

This final example builds a text string:

```
/* create a date and time stamp as a footnote */
entryfootnote  eval(put(today(),date9.)||" : "||put(time(),timeampm8.));
```

Valid GTL expressions are identical to valid WHERE expressions. See the WHERE statement documentation in Base SAS for a comprehensive list of operators and operands. Unlike WHERE expressions, however, GTL expressions do not perform operations that create subsets. For example, the difference between the result of a WHERE expression and that of a logical GTL expression on a column is that the GTL expression returns a Boolean value for each observation, without changing the number of observations.

For example, the expression for the Y= argument below does not reduce the number of observations that are plotted.

```
scatterplot x=name y=eval(height between 40 and 60);
```

Instead, the computed numeric column for the Y= argument consists of 0s and 1s, based on whether each observation's HEIGHT value is between 40 and 60.

Whenever expressions are used to create new columns, a new column name is internally manufactured so that it does not collide with other columns in use.

Expressions in Statement Syntax. Throughout GTL documentation, you see *expression* used in statement documentation:

BOXPLOT X= *column* | *expression*
  Y= *numeric-column* | *expression* </ option(s)>;

For the X= argument in this BOXPLOT syntax, *expression* means any EVAL(*expression*) that results in either a numeric or character column. An expression that yields a constant is not valid.

For the Y= argument, *expression* means any EVAL(*expression*) that results in a numeric column. The *expression* cannot result in a character column or any constant.

REFERENCELINE X= *x-axis-value* | *column* | *expression* </ option(s)>;

For a single line in this REFERENCELINE syntax, the X= argument can be a constant (*x-axis-value*). For multiple lines, it can be a column. In either case, the supplied value(s) must have the same data type as the axis. Thus, EVAL(*expression*) can result in a constant, or it can result in a numeric or character column. In either case, the data type of the result must agree with the axis type.

Type Conversion in GTL Expressions. Although expressions that are used in a DATA step perform automatic type conversion, GTL expression evaluation does not. Thus, you must use one or more functions to perform required type conversions in an expression; otherwise, the expression generates an error condition without warning when the template is executed.

For example, consider the following GTL expression:

```
if(substr(value, 1, 2) = "11")
```

This expression uses the SUBSTR function to determine whether the first two characters from VALUE evaluate to the string value "11". If VALUE is a string, the expression works fine. However, if VALUE is numeric, then the expression generates an error condition. For a numeric, you must convert the value to a string before passing it to the SUBSTR function. The following modification uses the CATS function to perform the type conversion when necessary:

```
if(substr(cats(value, 1, 2)) = "11")
```

Functions

Overview

GTL supports a large number of functions, including SAS functions that can be used in the context of a WHERE expression, and other functions that are defined only in GTL.

SAS functions that can be used in a WHERE expression include the following types of functions:

- character handling functions
- date and time functions
- mathematical and statistical functions.

Note: Not all SAS functions are available in WHERE expressions. Call routines and other functions that are restricted to the DATA step (LAG, VNAME, and OPEN, for example) are the types of functions that cannot be used.

All of the functions that are used in GTL must be enclosed within an EVAL function.

General Functions Supported Only in GTL

The following table shows functions that are used only in GTL. In all of these functions, *column* can be either the name of a column in the input data set, or a dynamic variable or macro variable that resolves to a column.

Function Name	Description
COLNAME(*column*)	returns the case-sensitive name of the column
COLLABEL(*column*)	returns the case-sensitive label of the column. If no label is defined for the column, the case-sensitive name of the column is returned.
EXISTS (*item*)	returns 1 if the specified *item* exists, 0 otherwise. If *item* is a column, EXISTS tests for the presence of the column in the input data set. If *item* is a dynamic variable or a macro variable, EXISTS tests whether the variable has been initialized at runtime.
EXPAND(*numeric-column*, *freq-column*)	creates a new column whose values equal (*numeric-column* * *frequency-column*)

Function Name	Description
ASORT (*column*, RETAIN=ALL)	sorts all of the data object's columns, in ascending order, by the values of *column*. SORT is an alias for ASORT. WARNING: if the RETAIN=ALL argument is not included, *column* alone is sorted, not the other columns, thereby losing row-wise correspondence.
DSORT (*column*, RETAIN=ALL)	sorts all of the data object's columns, in descending order, by the values of *column*. WARNING: if the RETAIN=ALL argument is not included, column alone is sorted, not the other columns, thereby losing row-wise correspondence.
NUMERATE(*column*)	returns a column that contains the ordinal position of each observation in the input data set (similar to an OBS column).

The following code shows some example uses of the GTL functions:

```
/* arrange bars in descending order of response values */
 barchartparm x=region y=eval(dsort(amount,retain=all));

/* Label outliers with their position in the data set.
   It does not matter which column is used for NUMERATE(). */
boxplot x=age y=weight / datalabel=eval(numerate(age));

/* Add information about the column being processed.
   The column name is passed by a dynamic. */
entrytitle "Distribution for " eval(colname(DYNVAR));
```

GTL Summary Statistic Functions

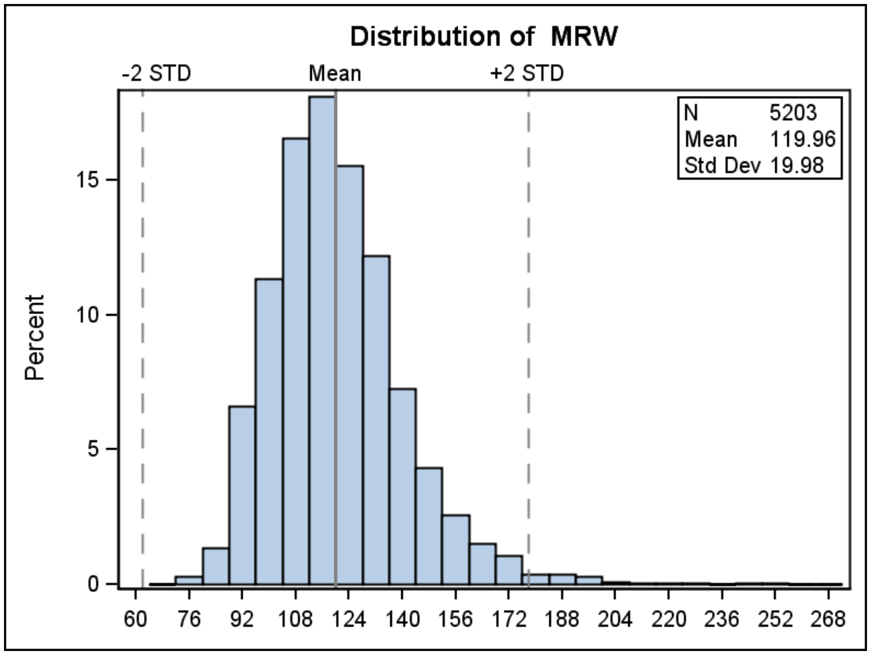
The following functions return a numeric constant, based on a summary operation that is performed on a numeric column. The results of these functions are the same as if the corresponding statistics were requested with PROC SUMMARY. These functions take a single argument, which resolves to the name of a numeric column. They take precedence over similar multi-argument DATA step functions.

number = EVAL(function-name(numeric-column))	
Function Name	**Description**
CSS	Corrected sum of squares
CV	Coefficient of variation
KURTOSIS	Kurtosis
LCLM	One-sided confidence limit below the mean
MAX	Largest (maximum) value
MEAN	Mean

number = EVAL(function-name(numeric-column))	
Function Name	**Description**
MEDIAN	Median (50th percentile)
MIN	Smallest (minimum) value
N	Number of non-missing values
NMISS	Number of missing values
P1	1st percentile
P5	5th percentile
P25	25th percentile
P50	50th percentile
P75	75th percentile
P90	90th percentile
P95	95th percentile
P99	99th percentile
PROBT	p-value for Student's t statistic
Q1	First quartile
Q3	Third quartile
QRANGE	Interquartile range
RANGE	Range
SKEWNESS	Skewness
STDDEV	Standard deviation
STDERR	Standard error of the mean
SUM	Sum
SUMWGT	Sum of weights
T	Student's t statistic
UCLM	One-sided confidence limit above the mean
USS	Uncorrected sum of squares

number = EVAL(function-name(numeric-column))	
Function Name	**Description**
VAR	Variance

The following example uses GTL summary statistic functions to dynamically construct reference lines and a table of statistics for a numeric variable, which is supplied at runtime.




```
proc template;
 define statgraph expression;
  dynamic NUMVAR "required";
  begingraph;
   entrytitle "Distribution of  " eval(colname(NUMVAR));
   layout overlay /  xaxisopts=(display=(ticks tickvalues line));
    histogram NUMVAR;

     /* create reference lines at computed positions */
    referenceline x=eval(mean(NUMVAR)+2*std(NUMVAR)) /
      lineattrs=(pattern=dash) curvelabel="+2 STD";
    referenceline x=eval(mean(NUMVAR)) /
      lineattrs=(thickness=2px) curvelabel="Mean";
    referenceline x=eval(mean(NUMVAR)-2*std(NUMVAR)) /
      lineattrs=(pattern=dash) curvelabel="-2 STD";

 /* create inset */
    layout gridded / columns=2 order=rowmajor
      autoalign=(topleft topright) border=true;
     entry halign=left "N";
     entry halign=left eval(strip(put(n(NUMVAR),12.0)));
     entry halign=left "Mean";
     entry halign=left eval(strip(put(mean(NUMVAR),12.2)));
     entry halign=left "Std Dev";
```

```
      entry halign=left eval(strip(put(stddev(NUMVAR),12.2)));
     endlayout;
    endlayout;
  endgraph;
 end;
run;

proc sgrender data=sashelp.heart template=expression;
  dynamic numvar="MRW";
run;
```

Conditional Logic

GTL supports conditional logic that enables you to include or exclude one or more GTL statements at runtime:

IF (*condition*)

GTL statement(s);

ELSE

GTL statement(s);

ENDIF ;

The IF statement requires an ENDIF statement, which delimits the IF block. The IF block can be placed anywhere within the BEGINGRAPH / ENDGRAPH block.

The *condition* is an expression that evaluates to a numeric constant, where all numeric constants other than 0 and MISSING are true. The IF block is evaluated with an implied EVAL(*condition*), so it is not necessary to include an EVAL as part of the *condition*.

Here are some examples:

```
/* test a computed value */
if (weekday(today()) in (1 7))
   entrytitle "Run during the weekend";
else
   entrytitle "Run during the work week";
endif;

/* test for the value of a numeric dynamic */
if ( ADDREF > 0 )
   referenceline y=1;
   referenceline y=0;
   referenceline y=-1;
endif;

/* test for the value of a character dynamic */
if ( upcase(ADDREF) =: "Y")
   referenceline y=1;
   referenceline y=0;
   referenceline y=-1;
endif;

/* test whether a dynamic is initialized */
if (exists(ADDREF))
   referenceline y=1;
```

```
     referenceline y=0;
     referenceline y=-1;
   endif;
```

The GTL conditional logic is used only for determining which statements to render. It is not used to control what is in the data object. In the following example, the data object contains columns for DATE, AMOUNT, and LOG10(AMOUNT), but only one scatter plot is created.

```
if ( LOGFLAG )
  scatterplot x=date y=amount;
else
  scatterplot x=date y=eval(log10(amount));
endif;
```

For the conditional logic in GTL, it is seldom necessary to test for the existence of option values that are set by columns or dynamics. Consider the following statement:

```
scatterplot x=date y=amount / group=GROUPVAR;
```

This SCATTERPLOT statement is equivalent to the following code because option values that are set by columns that do not exist, or by dynamics that are uninitialized, simply "drop out" at runtime and do not produce errors or warnings:

```
if ( exists(GROUPVAR) )
  scatterplot x=date y=amount / group=GROUPVAR;
else
  scatterplot x=date y=amount;
endif;
```

The GTL code that is specified in the conditional block must contain complete statements and / or complete blocks of statements. For example, the following IF block produces a compile error because there are more LAYOUT statements than ENDLAYOUT statements:

```
/* produces a compile error */
if ( exists(SQUAREPLOT) )
  layout overlayequated / equatetype=square;
else
  layout overlay;
endif;

    scatterplot x=XVAR y=YVAR;
endlayout;
```

The following logic is the correct conditional construct:

```
if ( exists(SQUAREPLOT) )
  layout overlayequated / equatetype=square;
    scatterplot x=XVAR y=YVAR;
  endlayout;
else
  layout overlay;
    scatterplot x=XVAR y=YVAR;
  endlayout;
endif;
```

GTL does not provide ELSE IF syntax, but you can create a nested IF/ ELSE block as follows:

IF (*condition*)

GTL statement(s);

ELSE
  IF (*condition*)
    GTL statement(s);
  ELSE
    GTL statement(s);
  ENDIF ;
ENDIF ;

The following example creates a generalized histogram that conditionally shows the variable label and combinations of fitted distribution curves:

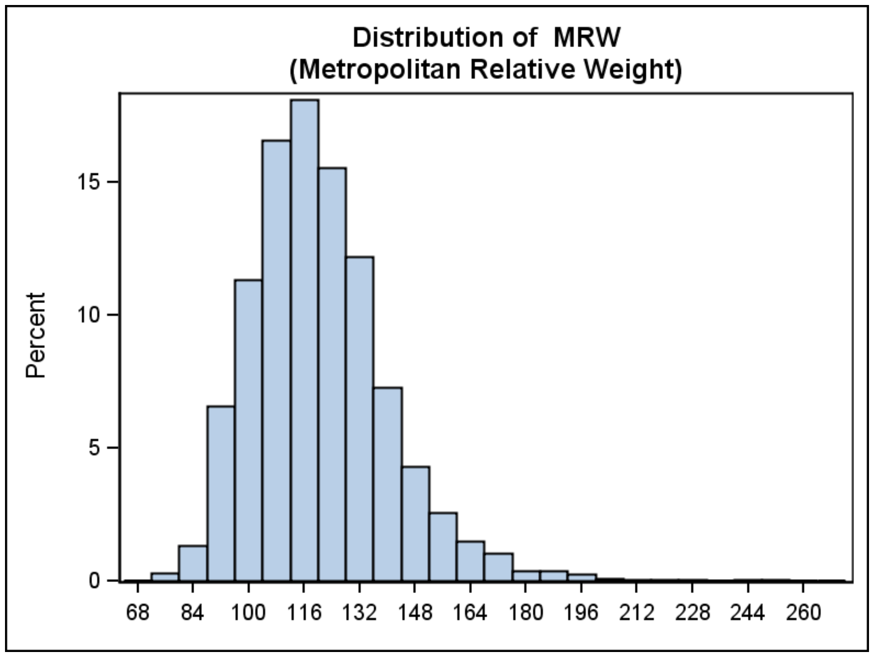
```
proc template;
  define statgraph conditional;
    dynamic NUMVAR "required" SCALE CURVE;
    begingraph;
      entrytitle "Distribution of  " eval(colname(NUMVAR));

      if ( colname(NUMVAR) ne collabel(NUMVAR) )
        entrytitle "(" eval(collabel(NUMVAR)) ")";
      endif;

      layout overlay / xaxisopts=(display=(ticks tickvalues line));
        histogram NUMVAR / scale=SCALE;

        if ( upcase(CURVE) in ("ALL" "NORMAL" ) )
          densityplot NUMVAR / normal() name="N"
            lineattrs=GraphData1 legendlabel="Normal Distribution";
        endif;

        if ( upcase(CURVE) in ("ALL" "KDE" "KERNEL") )
          densityplot NUMVAR / kernel() name="K"
            lineattrs=GraphData2 legendlabel="Kernel Density Estimate";
        endif;
        discretelegend "N" "K";
      endlayout;

    endgraph;
  end;
run;
```

- The DYNAMIC statement identifies the dynamic variables.
- The first IF block specifies an ENTRYTITLE statement that is conditionally executed if the column name differs from the column label.
- The next two IF blocks evaluate the value of the dynamic variable CURVE. If CURVE is not used, the code in the conditional blocks is not executed. If CURVE is initialized to one of the strings "all" or "normal" in any letter case, then the first DENSITYPLOT statement is executed. If CURVE is initialized to one of the strings "all", "kde", or "kernel" in any letter case, then the second DENSITYPLOT statement is executed. Thus, the results of the conditional logic determine whether zero, one, or two density plots are generated in the graph.
- Constructing the legend does not require conditional logic because any referenced plot names that do not exist are not used.

After submitting the template code, we can execute the template with various combinations of dynamic values.

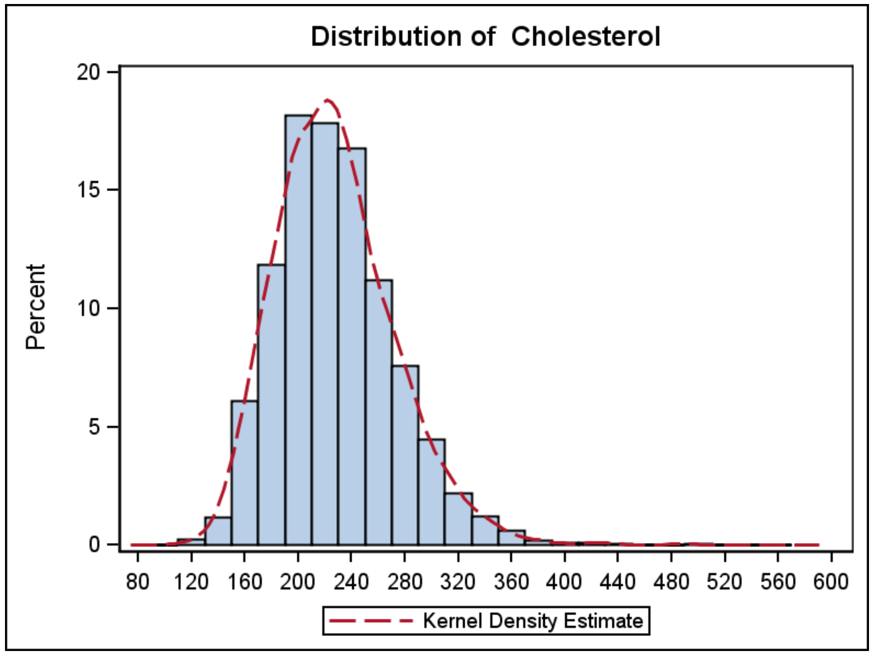
In this first execution, the NUMVAR dynamic is initialized with a column that has a defined label, so two title lines are generated. The first title line displays the column name, and the second title line displays the column label. The CURVE dynamic is not initialized, so the template does not generate a density plot.

```
proc sgrender data=sashelp.heart template=conditional;
  dynamic numvar="mrw";
run;
```



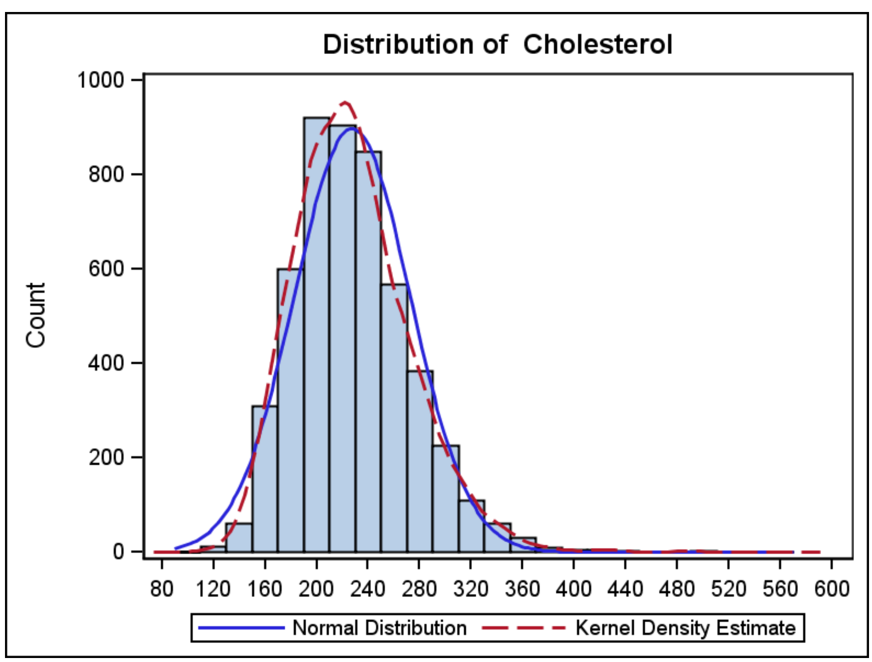
In this next execution of the template, the NUMVAR dynamic is initialized with a column that does not have a label, so only a single title line is displayed in the graph. The CURVE dynamic is initialized with the value "kde", so in addition to the histogram, the template generates a kernel density estimate.

```
proc sgrender data=sashelp.heart template=conditional;
  dynamic numvar="cholesterol" curve="kde";
run;
```



In this final execution of the template, the CURVE dynamic is initialized with the value "all", so in addition to the histogram, the template generates a normal density estimate and a kernel density estimate.

```
proc sgrender data=sashelp.heart template=conditional;
  dynamic numvar="cholesterol" scale="count" curve="all";
run;
```



The value of the SCALE dynamic does not need to be verified. If it is not one of COUNT, PERCENT, or PROPORTION (not case sensitive), the default scale is used with no warning or error.

Chapter 16
Adding Insets to a Graph

Uses for Insets in a Graph

Insets are commonly strings or tables of text that are displayed in the plot area to communicate relevant statistics, parameters, or other information relating to a graph. The information presented in an inset might come from

- text that appears in the template definition
- values that are computed with expressions within the template
- values that are passed externally to the inset by dynamics or macro variables
- columns that are assigned to an INSET= option on statements that support the option.

Inset information is often specified on ENTRY statements. However, the SCATTERPLOTMATRIX statement and the classification panel layouts (DATALATTICE and DATAPANEL layouts) provide options (for example, INSET=) that enable you to construct and locate insets in multi-cell layouts, without using ENTRY statements.

This chapter shows several techniques for adding insets to a graph. It assumes that you are familiar with the concepts and techniques presented in Chapter 7, "Adding and Changing Text in a Graph," on page 101 and Chapter 9, "Using a Simple Multi-cell Layout," on page 143.

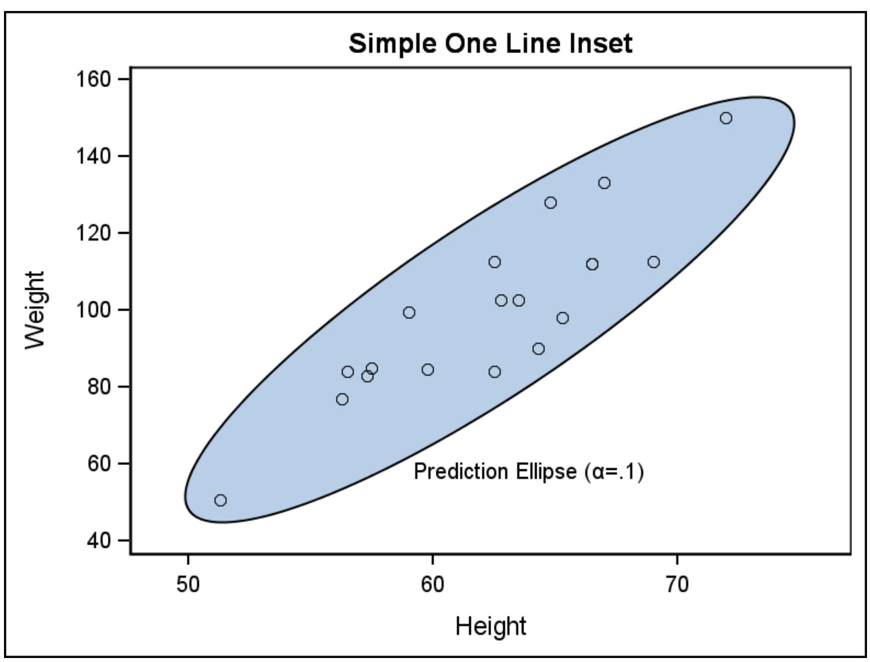
Creating a Simple Inset with an ENTRY Statement

You can use an ENTRY statement to create a simple inset within most layout blocks.

If you create the insets within a 2D overlay-type layout, you can use each ENTRY statement's AUTOALIGN= option or HALIGN= and VALIGN= options to position the text within the plot area. The HALIGN= and VALIGN= options position the text in an absolute position (such as HALIGN=LEFT and VALIGN=TOP). The AUTOALIGN= option is used for dynamic positioning that is based on placement of the graphical components in the plot area.

For example, to add an inset to an overlay of a scatter plot and an ellipse, you would like for the text to appear where it does not collide with markers or the ellipse, if at all possible. The AUTOALIGN=AUTO setting places the text in an area with the least congestion.

```
begingraph;
  entrytitle "Simple One Line Inset";
  layout overlay;
    ellipse x=height y=weight / alpha=.1 type=predicted display=all;
    scatterplot x=height y=weight;
    entry "Prediction Ellipse (" {unicode alpha} "=.1)" /
      autoalign=auto ;
  endlayout;
endgraph;
```



Note: The AUTO setting for the AUTOALIGN= option evaluates only the data points of scatter plots to determine the ENTRY position. When other plot types are present, their data representations are not evaluated and the ENTRY text might overlap a graphic element in the plot area.

Creating an Inset as a Table of Text

Perhaps the most common use for an inset is to display a table of statistics within the graph. This section shows how to construct that type of basic table. Later examples will show how to make the contents of the table more dynamic and how to integrate the table into the graph.

The basic technique for constructing the table is to place several ENTRY statements in a LAYOUT GRIDDED block. Each ENTRY statement becomes a cell of the grid. ENTRY statement options and layout options are used to further organize the table.

Suppose you want to create the following table of text:

N	5203
Mean	119.96
Std Dev	19.98

The simplest technique for creating the table is to construct a one-column, three-row table. The following example uses three ENTRY statements: one for each row in the table. The statistic name is left-justified in each row, and the statistic value is right-justified:

```
layout gridded / columns=1  border=true;
  entry halign=left "N"       halign=right "5203" ;
  entry halign=left "Mean"    halign=right "119.96" ;
  entry halign=left "Std Dev" halign=right "19.98" ;
endlayout;
```

Another technique is to create the table with two columns and three rows. This approach places each statistic name and statistic value in its own cell. Although this technique requires six ENTRY statements, it is a more flexible arrangement because each column alignment can be set independently. The following example left-justifies the text for each ENTRY statement:

N	5203
Mean	119.96
Std Dev	19.98

```
layout gridded / columns=2  order=rowmajor border=true;
  /* row 1 */
    entry halign=left "N";
    entry halign=left "5203";
  /* row 2 */
    entry halign=left "Mean";
    entry halign=left "119.96";
  /* row 3 */
    entry halign=left "Std Dev";
```

```
    entry halign=left "19.98";
endlayout;
```

ORDER=ROWMAJOR means that cells are populated horizontally, starting from column 1, followed by column 2, and then advancing to the next row. You should order the ENTRY statements as shown. To add additional rows in the table, just add additional pairs of ENTRY statements.

Of course, the LAYOUT GRIDDED statement enables you to organize cells by column, so you can achieve this same effect with ORDER=COLUMNMAJOR. The following code fragment populates the cells vertically down the columns by populating the first cell in row 1, followed by the first cell in row 2, followed by the first cell in row 3, and then advancing to the next column.

```
layout gridded / rows=3 order=columnmajor border=true;
  /* column 1 */
    entry halign=left "N";
    entry halign=left "Mean";
    entry halign=left "Std Dev";
  /* column 2 */
    entry halign=left "5203";
    entry halign=left "119.96";
    entry halign=left "19.98";
endlayout;
```

In both cases, an HALIGN=LEFT prefix option was added to each ENTRY statement to left-justify its text (the default is HALIGN=CENTER). Note that the column widths in the table are determined by the longest text string in each column on a per column basis.

The following example illustrates how to change the column justification and add extra space between the columns with the COLUMNGUTTER= option. Borders have been added to the ENTRY statements to show the text boundaries and alignment. Although it is not used in this example, the LAYOUT GRIDDED statement also provides a ROWGUTTER= option to add space between all rows.

N	5203
Mean	119.96
Std Dev	19.98

```
layout gridded / rows=3 order=columnmajor
                 columngutter=5px border=true;
  /* column 1 */
    entry halign=left "N"       / border=true;
    entry halign=left "Mean"    / border=true;
    entry halign=left "Std Dev" / border=true;
  /* column 2 */
    entry halign=right "5203"   / border=true;
    entry halign=right "119.96" / border=true;
    entry halign=right "19.98"  / border=true;
endlayout;
```

With the borders turned on in the layout, you should notice that there is spacing that appears on the left and right of the ENTRY text. The space is called padding, and it can be explicitly set with the PAD= option in the ENTRY statement. The default padding (in pixels) for ENTRY statements is

```
PAD=(TOP=0  BOTTOM=0  LEFT=3  RIGHT=3)
```

You can adjust that padding as desired.

To embellish the basic inset table with a spanning title, nest one GRIDDED layout within another GRIDDED layout. In the following example, notice that the outer GRIDDED layout has one column and two rows (the nested GRIDDED layout is treated as one cell).

Stat Table

N	5203
Mean	119.96
Std Dev	19.98

```
layout gridded / columns=1;
  entry textattrs=(weight=bold) "Stat Table";
  layout gridded / rows=3 order=columnmajor border=true;
  /* column 1 */
    entry halign=left "N";
    entry halign=left "Mean";
    entry halign=left "Std Dev";
  /* column 2 */
    entry halign=left "5203";
    entry halign=left "119.96";
    entry halign=left "19.98";
  endlayout;
endlayout;
```

Positioning an Inset

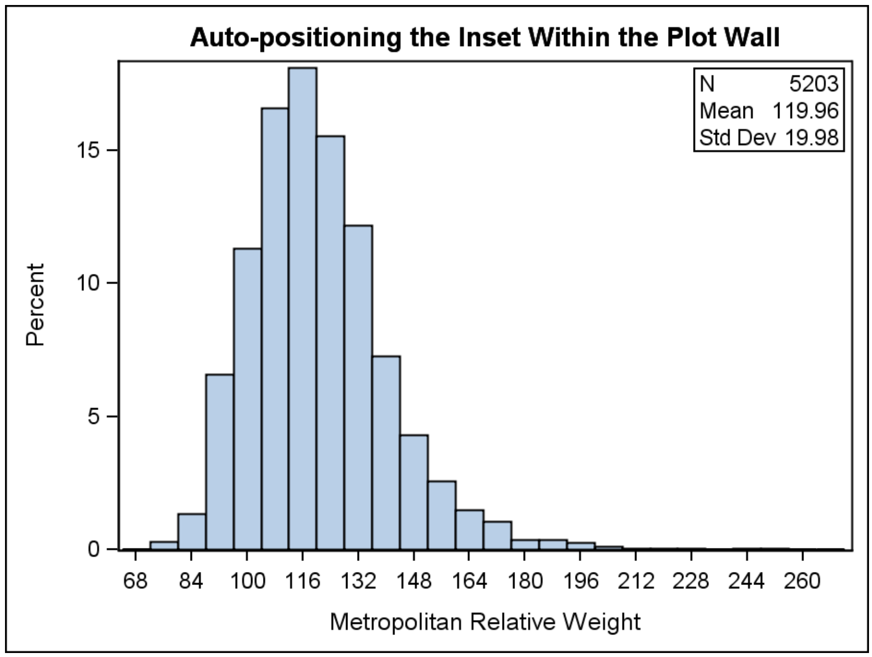
If a table of text is used as an inset within a 2D overlay-type layout, you can position the table within the parent layout with options on the LAYOUT GRIDDED statement. You can use the AUTOALIGN= option to automatically position the inset to avoid collision with scatter points, lines, bars, and other plot components.

Alternatively, you can use the HALIGN= and VALIGN= options to position the table absolutely. The combined values provide nine possible fixed positions. The disadvantage of using the HALIGN= and VALIGN= options is that they do not attempt to avoid collision with other plot components.

The following example uses the AUTOALIGN= option to restrict the table position to one of the upper corners of the plot wall.

```
proc template;
define statgraph  ginset3a;
 begingraph;
  entrytitle "Auto-positioning the Inset Within the Plot Wall";
  layout overlay;
   histogram mrw;
   layout gridded / columns=1 border=true autoalign=(topleft topright) ;
     entry  halign=left "N"       halign=right "5203";
     entry  halign=left "Mean"    halign=right "119.96";
     entry  halign=left "Std Dev" halign=right "19.98";
   endlayout;
```

```
    endlayout;
   endgraph;
  end;
  run;

  proc sgrender data=sashelp.heart template=ginset3a;
  run;
```



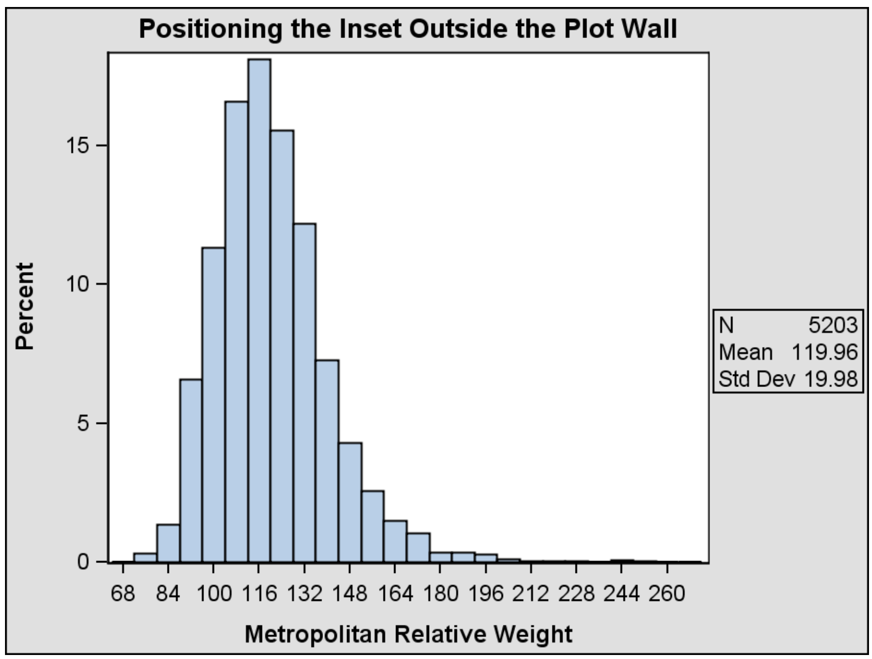
In this particular case there was not enough space to display the inset in the top left position, so the next position was used because it has no collision. With a different set of data, the inset might appear in the top left position. If both positions resulted in a collision, the position with the least collision would be used. You can specify an ordered list of up to nine positions for the AUTOALIGN list: TOPLEFT, TOP, TOPRIGHT, LEFT, CENTER, RIGHT, BOTTOMLEFT, BOTTOM, and BOTTOMRIGHT. For a scatter plot where "open" space is not predictable, you can specify AUTOALIGN=AUTO, which selects a position that minimizes collision with the scatter markers.

Note: The AUTO setting for the AUTOALIGN= option works best when the layout contains only scatter plots. When other plot types are present, the ENTRY text might overlap a graphic element in the plot area.

Outside Insets. An inset does not have to be placed inside the plot wall. This next example positions an inset in the sidebar of a LATTICE layout.

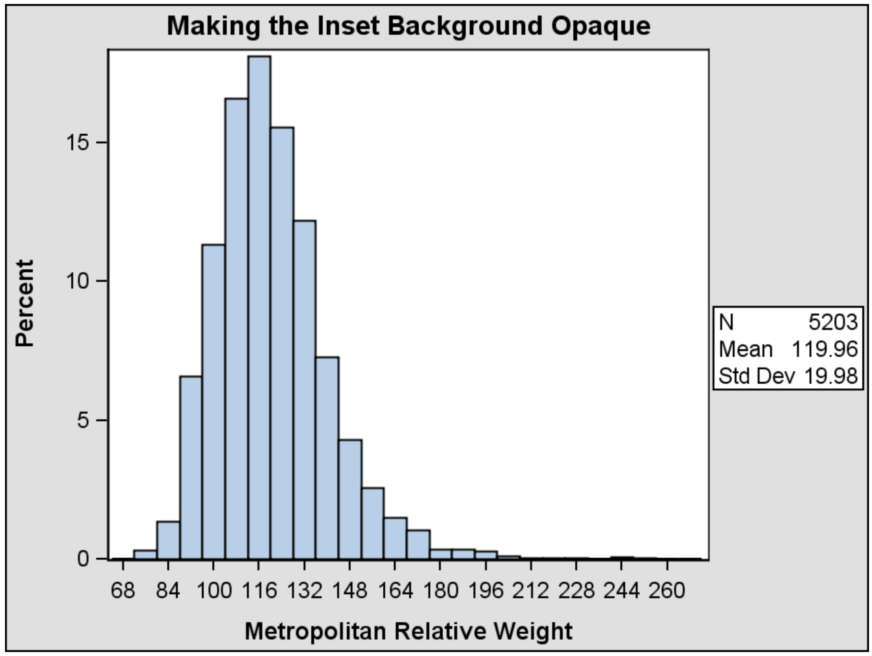
```
proc template;
 define statgraph  ginset3b;
  begingraph / pad=2px;
    entrytitle "Positioning the Inset Outside the Plot Wall";
    layout lattice;
      layout overlay;
        histogram mrw;
      endlayout;
      sidebar / align=right;
        layout overlay / pad=(left=2px);
          layout gridded / columns=1  border=true;
            entry  halign=left "N"       halign=right "5203";
```

```
            entry  halign=left "Mean"    halign=right "119.96";
            entry  halign=left "Std Dev" halign=right "19.98";
          endlayout;
            endlayout;
      endsidebar;
    endlayout;
  endgraph;
 end;
run;

ods listing style=default;
proc sgrender data=sashelp.heart template=ginset3b;
run;
```



By default, the background of ENTRY statements and a GRIDDED layout are transparent. So if the current style defines a background color and the inset does not appear in the plot wall, the style's background color will be seen through the inset. You can make the background of the insert opaque and set its background color to match the plot wall color, as shown in the following code fragment:

```
sidebar / align=right;
  layout overlay / pad=(left=2px);
    layout gridded / columns=1  border=true
           opaque=true  backgroundcolor=graphWalls:color ;
      entry  halign=left "N"       halign=right "5203";
      entry  halign=left "Mean"    halign=right "119.96";
      entry  halign=left "Std Dev" halign=right "19.98";
    endlayout;
  endlayout;
endsidebar;
```



Creating an Inset with Values that are Computed in the Template

The examples presented so far have "hard coded" the statistic values in the compiled template. Hard-coding the statistic values requires you to change and recompile the template code whenever the column values change or you want to use different columns for the analysis. A more flexible way to present a statistics table is to compute its content as follows:

- use GTL functions to calculate any required statistics
- use dynamic variables as placeholders for column names in the template
- at runtime, initialize the dynamic variables so that they resolve to the names of columns in the data object that is used to provide data values for the graph.

GTL supplies several functions that you can use to calculate the statistics, including functions that match the statistic keywords used by PROC SUMMARY. GTL functions are always specified within an EVAL function. To declare dynamic variables, you use the DYNAMIC statement.

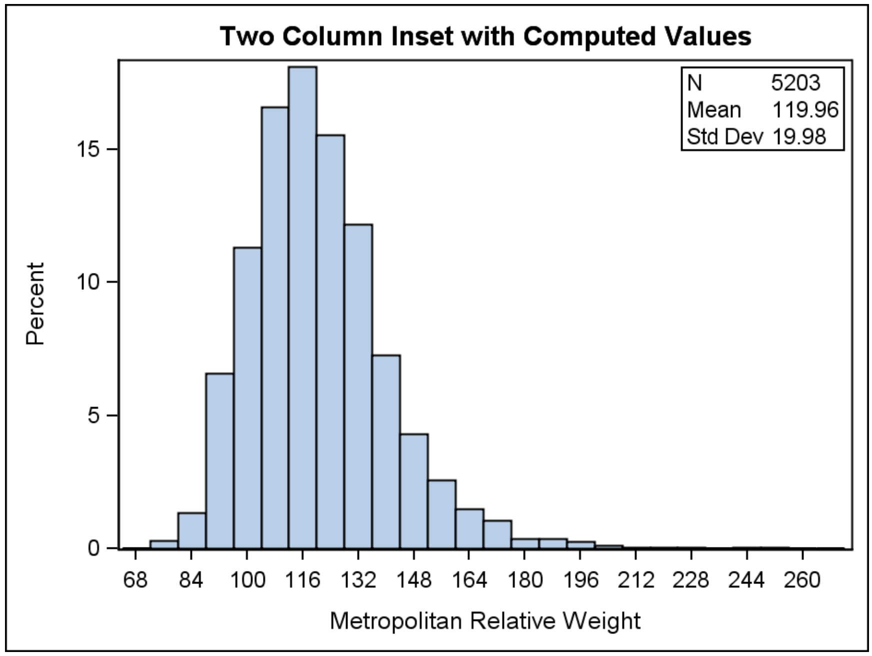
The following example uses the DYNAMIC statement to declare a dynamic variable named VAR, which is used in the functions N, MEAN, and STDDEV to calculate the statistics that are displayed in the statistics table:

```
proc template;
 define statgraph  ginset4a;
  dynamic VAR;
  begingraph;
    entrytitle "Two Column Inset with Computed Values";
    layout overlay;
      histogram VAR ;
      layout gridded / rows=3 order=columnmajor border=true
```

```
                  autoalign=(topleft topright);
        /* column 1 */
         entry halign=left "N";
         entry halign=left "Mean";
         entry halign=left "Std Dev";
        /* column 2 */
         entry halign=left eval(strip(put(n(VAR),12.0))) ;
         entry halign=left eval(strip(put(mean(VAR),12.2))) ;
         entry halign=left eval(strip(put(stddev(VAR),12.2))) ;
       endlayout;
     endlayout;
   endgraph;
  end;
run;

proc sgrender data=sashelp.heart template=ginset4a;
  dynamic VAR="mrw";
run;
```

- Dynamic VAR is first referenced in the HISTOGRAM statement, where it is used to represent the variable that will provide numeric values for the histogram.
- Dynamic VAR is again referenced on each of the ENTRY statements that specify the statistic values to use in the statistics table. Each of the ENTRY statements uses an EVAL function to specify functions to calculate the statistic.
- On each of the ENTRY statements, the STRIP function strips leading and trailing blanks from the returned values. The PUT function on the first ENTRY statement returns the statistics value with format 12.0, and the next two PUT statements return values with format 12.2. The N, MEAN, and STDDEV functions return the number of observations, mean, and standard deviation of variable VAR.
- In the SGRENDER procedure, the DYNAMIC statement initializes dynamic VAR so that it resolves at runtime to column MRW from the SASHELP.HEART data set. Because the dynamic will resolve to a column name, the value that is assigned to it is enclosed in quotation marks. (Values for dynamics that resolve to column names or strings should be quoted. Numeric values should not be quoted.)

See Chapter 15, “Using Conditional Logic and Expressions,” on page 259 for more information about the functions that can be used in the EVAL function. See Chapter 14, “Using Dynamics and Macro Variables to Make Flexible Templates,” on page 251 for more information on using dynamics and macro variables in GTL templates.



Creating an Inset from Values that are Passed to the Template

Overview

When the statistic that you want to display in an inset cannot be computed within the template, you can create an output data set from a procedure and then use dynamics or macro variables to "import" the computed values at runtime.

The following discussion explains how to create and call a macro that can pass a data set name and variable name to a previously compiled GTL template. To follow this discussion, you should understand the topics that are discussed in Chapter 14, “Using Dynamics and Macro Variables to Make Flexible Templates,” on page 251.

For this example, we will create a macro named HISTOGRAM that takes two arguments.

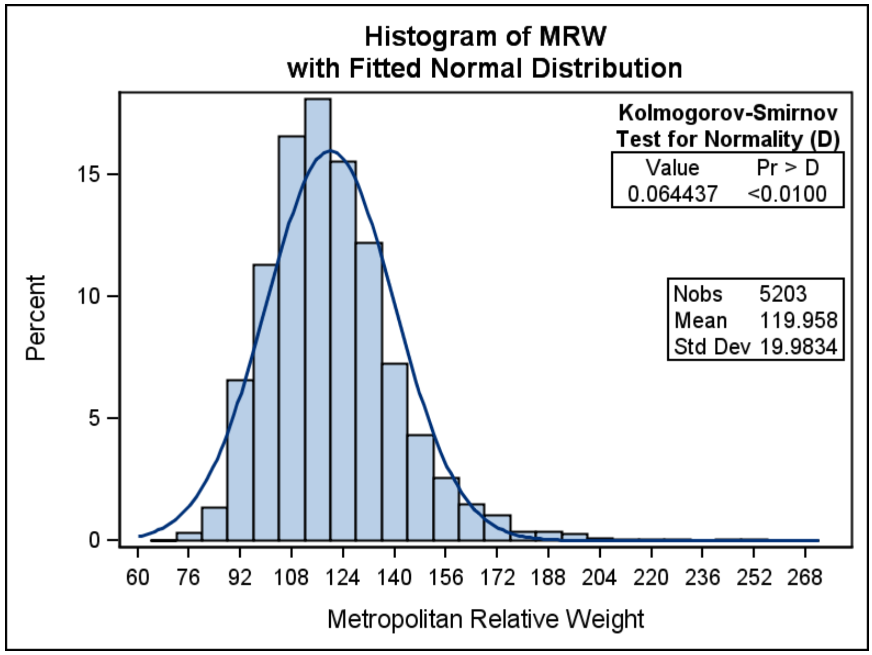
- The first argument, DSN, passes a data set name.
- The second argument, VAR, passes a variable name.

When invoked, the macro generates a histogram and model fit plot for the analysis variable VAR. The graph also displays two insets that show the related statistics.

The following call to the HISTOGRAM macro uses the data set SASHELP.HEART to generate a histogram for variable MRW (Metrapolitan Relative Weight for subjects in a heart study):

```
%histogram(sashelp.heart, mrw)
```

Figure 16.1 *Passing Parameter Values to a Template*



For more information on creating and calling macros, see the *SAS Macro Language: Reference*.

Creating a Template that Uses Macro Variables

This section creates a GTL template that can generate the histogram and model fit plot that is shown in Figure 16.1 on page 281.The template definition uses the MVAR statement to define macro variables that will provide runtime values and labels for the graph insets. The MVAR statement also defines a macro variable named VAR, which will be used as the column argument for the histogram and overlaid normal density plot.

For this example, the inset statistics are calculated in the macro body, and the value for macro variable VAR is passed as a parameter on the macro call.

Here is the GTL code for a template that we will name GINSET:

```
proc template;
define statgraph  ginset;
  MVAR VAR NOBS MEAN STD TEST TESTLABEL STAT PTYPE PVALUE ;
  begingraph;
    entrytitle "Histogram of " eval(colname(VAR));
    entrytitle "with Fitted Normal Distribution";
    layout overlay;
      histogram VAR ;
      densityplot VAR / normal();

      /* inset for normality test */
      layout gridded / columns=1  opaque=true
             autoalign=(topright topleft);
        entry TEST / textattrs=(weight=bold);
        entry "Test for Normality " TESTLABEL /
             textattrs=(weight=bold);
        layout gridded / columns=2   border=true;
          entry "Value"; entry PTYPE ;
```

```
          entry STAT;    entry  PVALUE  ;
        endlayout;
      endlayout;

      /* inset for descriptive statistics */
      layout gridded / columns=2 border=true
             opaque=true autoalign=(right left);
        entry halign=left "Nobs";    entry halign=left NOBS ;
        entry halign=left "Mean";    entry halign=left MEAN ;
        entry halign=left "Std Dev"; entry halign=left STD ;
      endlayout;
    endlayout;
  endgraph;
end;
run;
```

- The MVAR statement declares the macro variables that will be referenced in the template.
- The ENTRYTITLE statement specifies macro variable VAR as the argument on the COLNAME function, which returns the case-sensitive name of the column. Thus, the variable name that you pass on the macro call will be displayed in the graph title.
- The HISTOGRAM and DENSITYPLOT statements specify macro variable VAR as their column arguments. Again, the variable name that you pass on the macro call will determine that column name.
- The first LAYOUT GRIDDED block constructs a table to use as an inset. The inset identifies the normality test that is used in the analysis, and it displays the related probability statistic.

  The first two ENTRY statements in the layout block specify a title for the inset. Macro variable TEST, which will be initialized by the code in the macro body, identifies the normality test that is applied to the data. As we'll see later when we create the macro, either of two normality tests will be used, depending on the number of observations that are read from the data. Macro variable TESTLABEL provides either of two test labels, depending on which test is used at runtime.

  The nested LAYOUT GRIDDED statement defines a two-column table for the statistics table that is displayed in the first inset. Macro variable STAT in the first column provides the normality value, and macro variables PTYPE and PVALUE provide the probability statistics. These macro variables will be initialized by the code in the macro body.
- The last LAYOUT GRIDDED statement constructs a two-column inset that shows descriptive statistics for the analysis variable. Macro variables NOBS, MEAN, and STD will be calculated by the code in the macro body and will resolve to the number of observations in the data, the mean value, and the standard deviation.

Defining a Macro to Initialize the Variables and Generate the Graph

In "Creating a Template that Uses Macro Variables" on page 281 we created template GINSET, which declared the following macro variables:

TEST	Identifies the normality test that is applied to the data.
TESTLABEL	Provides the label that is associated with the applied normality test.

STAT	Provides the normality statistic that is calculated by the applied normality test.
PTYPE and PVALUE	Provide the probability (type and value) for the applied normality test.
NOBS, MEAN, and STD	Provide the number of observations, mean, and standard deviation for the analysis variable.

To initialize these macro variables, we will now create a macro that calculates values for them and also specifies an SGRENDER procedure that uses template GINSET. The macro needs two parameters: one for passing a SAS data set name, and a second for passing the name of a column in that data set.

The following macro code uses PROC UNIVARIATE to create two output data sets. A DATA step then reads the output data sets, creates the required macro variables, and assigns values to those macro variables in a local symbol table. When the macro runs the SGRENDER procedure, the values of the macro variables are imported into the GINSET template to produce a graph with insets, similar to the graph in shown in Figure 16.1 on page 281. As mentioned earlier, the normality test that is performed on the analysis variable will be based on the number of observations in that analysis variable.

Note: To make the following macro more robust, it could be designed to validate the parameters.

```
%macro histogram(dsn,var);

/* compute tests for normality */
ods output TestsForNormality=norm;
proc univariate data=&dsn normaltest;
 var &var;
 output out=stats n=n mean=mean std=std;
run;

%local nobs mean std test testlabel stat ptype pvalue;

data _null_;
  set stats(keep=n mean std);
  call symputx("nobs",n);
  call symput("mean",strip(put(mean,12.3)));
  call symput("std",strip(put(std,12.4)));
  if n > 2000 then /* use Shapiro-Wilk */
    set norm(where=(TestLab="D"));
  else /* use Kolmogorov-Smirnov */
    set norm(where=(TestLab="W"));
  call symput("testlabel","("||trim(testlab)||")");
  call symput("test",strip(test));
  call symput("ptype",strip(ptype));
  call symput("stat",strip(put(stat,best8.)));
  call symput("pvalue",psign||put(pvalue,pvalue6.4));
run;

proc sgrender data=&dsn template=ginset;
run;

%mend;
```

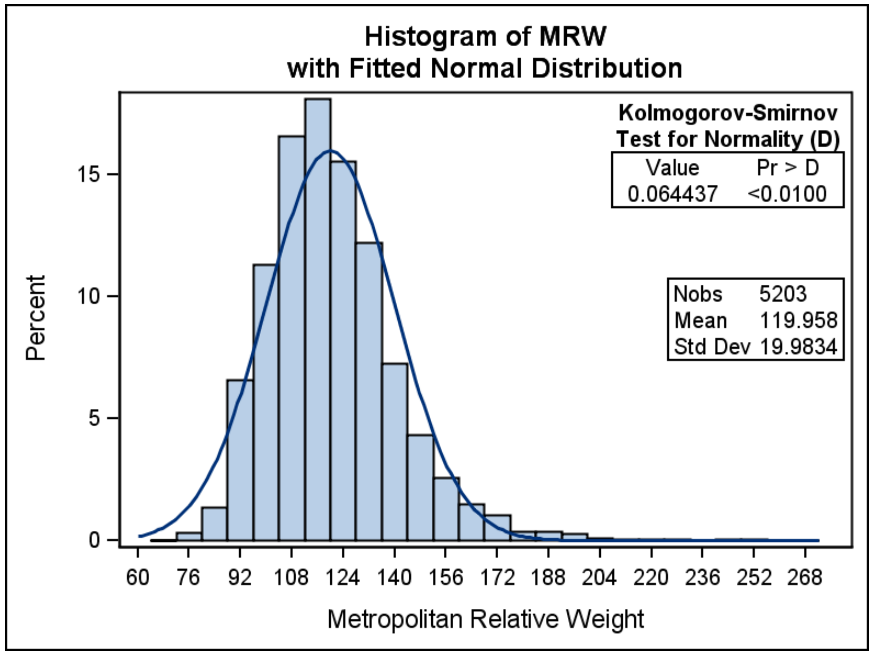
- The %MACRO statement declares a macro named HISTOGRAM that takes two parameters: DSN (for the data set name) and VAR (for the column name).
- The ODS OUTPUT statement produces a SAS data set named NORM from the TestsForNormality output object that will be generated by the UNIVARIATE procedure (next statement). For more information on the ODS OUTPUT statement, see the *SAS Output Delivery System: User's Guide*.
- Deriving the input data set name from the DSN parameter and the analysis variable name from the VAR parameter, the UNIVARIATE procedure calculates the number of observations, mean, and standard deviation for the analysis variable. It writes the values for these statistics to an output data set named STATS, storing the values in variables named N, MEAN, and STD.
- The %LOCAL statement creates a set of local macro variables to add to the local symbol table.
- The DATA step reads variables N, MEAN, and STD from the STATS data set.
- The first three CALL SYMPUT routines use the data input variables to assign labels and values to the local macro variables N, MEAN, and STD. On each CALL SYMPUT, the first argument identifies the macro variable to receive the value, and the second argument identifies the data input variable that contains the value to assign to the macro variable in the symbol.
- The IF/ELSE structure determines which normality test values to read from the NORM data set that was created by the ODS OUTPUT statement. If there are fewer than 2000 observations, the Shapiro-Wilk test values are used; otherwise, the Kolmogorov-Smirnov values are used.
- The remaining CALL SYMPUT routines assign values to the rest of the macro variables, using the values from variables in the NORM data set.

Executing the Macro

To execute the HISTOGRAM macro, we must pass it a data set name and the name of a numeric column in the data.

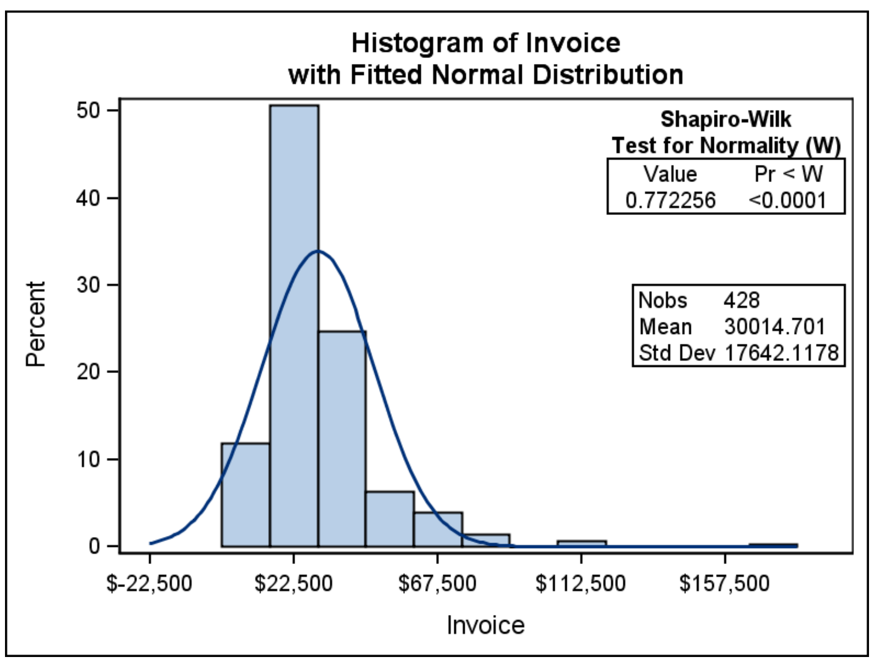
The following macro call passes the data set name SASHELP.HEART and the column name MRW. Because variable MRW has more than 2000 observations, the Kolmogorov-Smirnov test is used in the analysis.

```
%histogram(sashelp.heart, mrw)
```



This next macro call passes the data set name SASHELP. CARS and the column name INVOICE. Because variable INVOICE has 2000 or fewer observations, the Shapiro-Wilk test is used in the analysis.

```
%histogram(sashelp.cars,invoice)
```



Adding Insets to a SCATTERPLOTMATRIX Graph

The SCATTERPLOTMATRIX statement provides the following options for displaying insets in the cells of the graph matrix (see the documentation for the

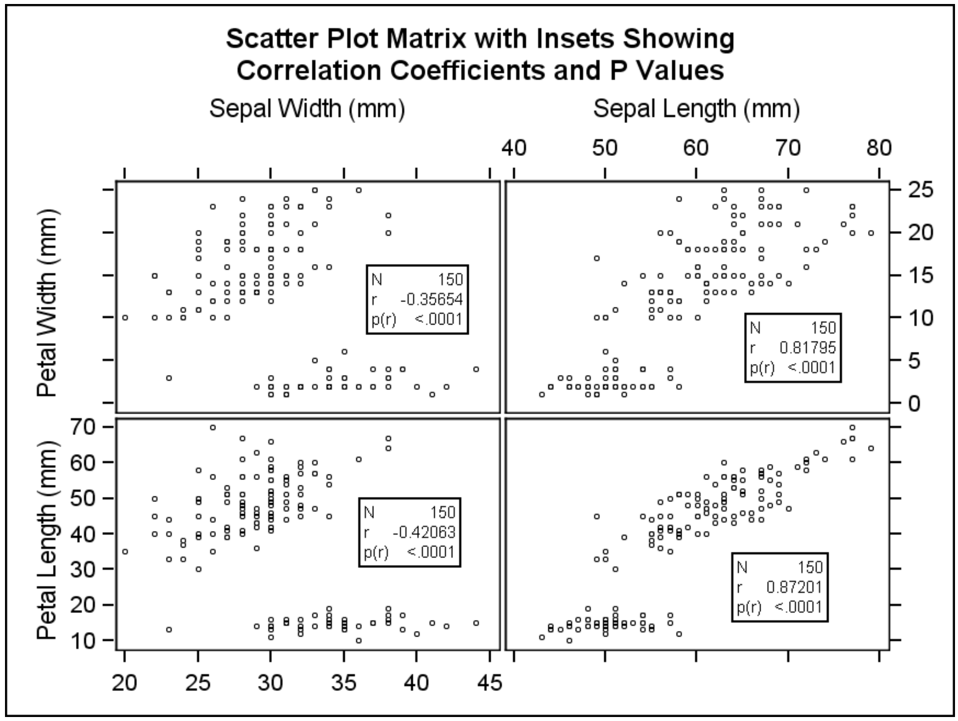
SCATTERPLOTMATRIX statement in the *SAS/GRAPH Graph Template Language Reference* for complete details about these options):

INSET= (*info-options*)	Determines what information is displayed in an inset. Accepts one, two, or all three of the following keywords: NOBS Number of observations PEARSON Pearson product-moment correlation PEARSONPVAL Probability value for the Pearson product-moment correlation This option must be used to determine which inset information is displayed in each cell. If this option is not used, the related CORROPTS= and INSETOPTS= options are ignored.
CORROPTS= (*correlation-options*)	Controls statistical options for computing correlations. These options are similar to PROC CORR options. Accepts one or more of the following keywords: EXCLNPWGT= specifies whether observations with non-positive weight values are excluded from the analysis. Accepts TRUE (the default) or FALSE. NOMISS= specifies whether observations with missing values are excluded from the analysis. Accepts TRUE (the default) or FALSE. WEIGHT= specifies a weighting variable to use in the calculation of Pearson weighted product-moment correlation. The observations with missing weights are excluded from the analysis. Accepts the name of a numeric column. VARDEF= specifies the variance divisor in the calculation of variances and covariances. Accepts one of the keywords DF (Degrees of Freedom, the default, N - 1), N (number of observations), WDF (sum of weights minus 1), WEIGHT (sum of weights).

INSETOPTS= (*appearance-options*)	Controls the inset placement and other appearance features. AUTOALIGN= specifies whether the inset is automatically aligned within the layout. Accepts keywords NONE (no auto-alignment, the default), AUTO (available only with scatter plots, attempts to center the inset in the area that is farthest from any surrounding markers), or a location list in parentheses that contains one or more keywords that identify the preferred alignment (TOPLEFT TOP TOPRIGHT LEFT CENTER RIGHT BOTTOMLEFT BOTTOM BOTTOMRIGHT). BACKGROUNDCOLOR= specifies the color of the inset background. Accepts a style reference or a color specification. BORDER= specifies whether a border is displayed around the inset. Accepts TRUE or FALSE (the default). HALIGN= specifies the horizontal alignment of the inset. Accepts keywords LEFT (the default), CENTER. or RIGHT. OPAQUE= specifies whether the inset background is opaque (TRUE) or transparent (FALSE, the default). TEXTATTRS= specifies the text properties of the entire inset. VALIGN= specifies the vertical alignment of the inset. Accepts keywords TOP (the default), CENTER. or BOTTOM.

The following example uses all three of these options to display an inset in the cells of a graph that is generated with the SCATTERPLOTMATRIX statement:

```
proc template;
 define statgraph spminset;
   begingraph;
     entrytitle "Scatter Plot Matrix with Insets Showing";
     entrytitle "Correlation Coefficients and P Values";
     layout gridded;
       scatterplotmatrix sepalwidth sepallength /
         rowvars=(petalwidth petallength)
         inset=(nobs pearson pearsonpval)
         insetopts=(autoalign=auto border=true opaque=true)
         corropts=(nomiss=true vardef=df)
         markerattrs=(size=5px);
     endlayout;
   endgraph;
 end;
run;

proc sgrender data=sashelp.iris template=spminset;
run;
```



Notice that the inset position might change from cell to cell to avoid obscuring point markers.

Adding Insets to Classification Panels

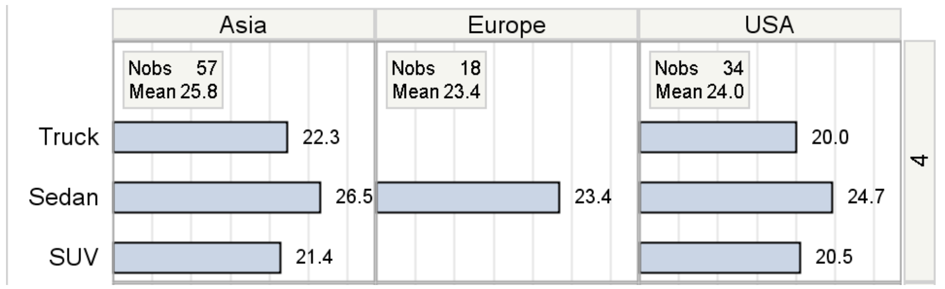
This section requires familiarity with Chapter 11, "Using Classification Panels," on page 185. You should skip this section if you are not familiar with the general coding for classification panels.

The DATALATTICE and DATAPANEL layouts provide INSET= and INSETOPTS= options for displaying insets in classification panels. The INSETOPTS= option supports the same placement and appearance features as those documented for the SCATTERPLOTMATRIX statement in "Adding Insets to a SCATTERPLOTMATRIX Graph" on page 285 However, unlike the SCATTERPLOTMATRIX statement, the DATALATTICE and DATAPANEL layouts do not have predefined information available. Thus, for the INSET= option, you must create the columns for the information that you want to display in the inset and integrate it with the input data before the graph is rendered. Then, on the INSET= option, you specify the name(s) of the column(s) that contain the desired information.

For example, the following template code uses INSET=(NOBS MEAN) to reference input data columns that are named NOBS and MEAN. When the graph is rendered, the values that are stored in these columns will be displayed in the inset.

In the inset display in this example, one row is displayed for each column that is listed on INSET=, and each row has two columns. The left column shows the column name (column label, if it is defined in the data), and the right column contains the column value for that particular cell of the panel. The number of rows of data for these columns should match the number of cells in the classification panel and the sequence in which the cells are populated.

The following template code defines a template named PANEL. The template "makes room" for the insets in each panel by adding a maximum row axis offset. In this case, OFFSETMAX=0.4 is sufficient, but the setting will vary case-by-case. This is what the first row of the classification panel with insets will look like:




```
proc template;
 define statgraph panel;
  begingraph;
    entrytitle "Average City MPG for Vehicles";
    entrytitle "by Origin, Cylinders and VehicleType";
    layout datalattice columnvar=origin rowvar=cylinders /
        columndatarange=unionall rowdatarange=unionall
        headerlabeldisplay=value
        headerbackgroundcolor=GraphAltBlock:color
        inset=(cellN cellMean)
        insetopts=(border=true
                   opaque=true backgroundcolor=GraphAltBlock:color)
        rowaxisopts=( offsetmax=.4 offsetmin=.1 display=(tickvalues) )
        columnaxisopts=(display=(label tickvalues)
                        linearopts=( tickvaluepriority=true
                          tickvaluesequence=(start=5 end=30 increment=5))
                        griddisplay=on offsetmin=0 offsetmax=.1);
      layout prototype;
        barchart x=type y=mean / orient=horizontal
                                 barwidth=.5 barlabel=true;
      endlayout;
    endlayout;
  endgraph;
 end;
run;
```

When this template is used, the input data must contain separate columns for the following:

classification variables	**columnvar=origin rowvar=cylinders**
inset information	**inset=(cellN cellMean)**
bar chart	**x=type Y=mean**

The data for this example is from the SASHELP.CARS data set. To calculate the number of observations and mean for the observations, we can use PROC SUMMARY.

The following PROC SUMMARY step calculates the number of observations and the mean of MPG_CITY for each of the classification interactions listed in the TYPES statement. CYLINDERS*ORIGIN is the crossing needed for the cell summaries, and CYLINDER*ORIGIN*TYPE is the crossing needed by each cell's bar chart.

The COMPLETETYPES option creates summary observations even when the frequency of the classification interactions is zero. Additionally, the code creates subsets in the input data to restrict the number of bars in each bar chart to at most three, and to reduce the number cells in the classification panel. There are three values of ORIGIN (Asia, Europe, and USA) and three values of CYLINDERS (4, 6, and 8).

For the insets to display accurate data, we must ensure that the order of the observations in the data corresponds to the column order for the CLASS statement of PROC SUMMARY. Because the panel cells are populated across one row before proceeding to the next row, the values of the panel's row variable (CYLINDERS) determines the panel order and must be specified first in the SUMMARY procedure's CLASS statement so that the values of CYLINDERS also determine the order for the statistics calculations.

```
/* compute the barchart data and inset information */

proc summary data=sashelp.cars completetypes;
  where type in ("Sedan" "Truck" "SUV") and
        cylinders in (4 6 8);
  class cylinders origin type;
  var mpg_city;
  output out=mileage mean=Mean n=Nobs / noinherit;
  types cylinders*origin cylinders*origin*type;
run;
```

The SAS log displays the following note when the procedure code is submitted:

```
NOTE: There were 337 observations read from the data set SASHELP.CARS.
      WHERE type in ('SUV', 'Sedan', 'Truck') and cylinders in (4, 6, 8);
NOTE: The data set WORK.MILEAGE has 36 observations and 6 variables.
```

Display 16.1 *Confirm the Order of Data Observations*

VIEWTABLE: mileage

	Cylinders	Origin	Type	_TYPE_	Mean	Nobs
1	4	Asia		6	25.842105263	57
2	4	Europe		6	23.444444444	18
3	4	USA		6	24	34
4	6	Asia		6	18.3	60
5	6	Europe		6	18.763157895	38
6	6	USA		6	18.475409836	61
7	8	Asia		6	15.2	10
8	8	Europe		6	16.166666667	24
9	8	USA		6	15.514285714	35
10	4	Asia	SUV	7	21.4	5
11	4	Asia	Sedan	7	26.510204082	49
12	4	Asia	Truck	7	22.333333333	3
13	4	Europe	SUV	7		0

Display 16.1 on page 290 shows the order of observations in the interim data set named MILEAGE. Notice that the first nine observations (where _TYPE_ equals 6) are the cell summaries. The remaining 27 observations (where _TYPE_ equals 7) are for each cell's bar chart.

To create separate columns for the inset, we need to store the _TYPE_= 6 observations in new columns. The following DATA step writes the inset information to another data set named OVERALL.

```
data mileage
     overall(keep=origin cylinders mean nobs
```

```
            rename=(origin=cellOrigin cylinders=cellCyl
                    mean=cellMean nobs=cellNobs ));
   set mileage; by _type_;
   if _type_ eq 6 then output overall;
   else output mileage;
 run;
```

The SAS log displays the following note when the code is submitted:

```
NOTE: There were 36 observations read from the data set WORK.MILEAGE.
NOTE: The data set WORK.MILEAGE has 27 observations and 5 variables.
NOTE: The data set WORK.OVERALL has 9 observations and 4 variables.
```

Finally, we create a new data set named SUMMARY, which merges the MILEAGE and OVERALL data sets. Note that this is a non-match merge (no BY statement), and that all columns in the two tables have unique names to prevent overwriting any data values.

```
data summary;
   merge mileage overall;
   label Mean="MPG (City)";
   format mean cellMean 4.1;
run;
```

```
NOTE: There were 27 observations read from the data set WORK.MILEAGE.
NOTE: There were 9 observations read from the data set WORK.OVERALL.
NOTE: The data set WORK.SUMMARY has 27 observations and 9 variables.
```

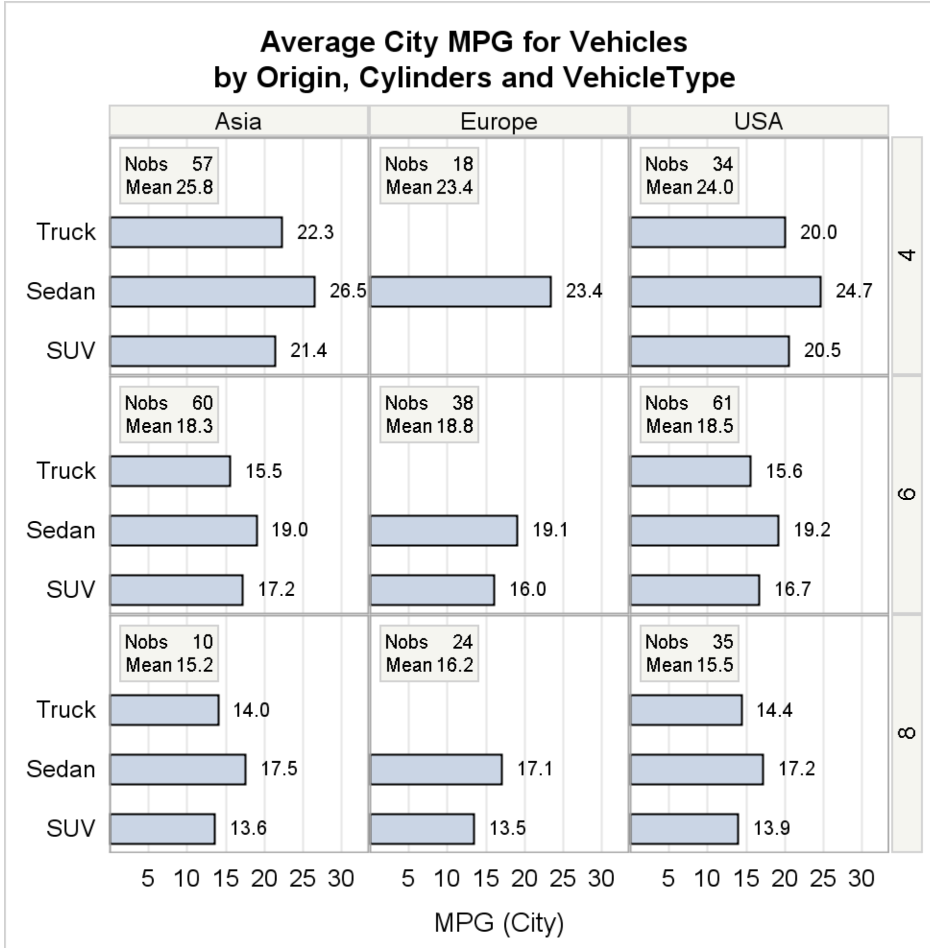
Display 16.2 *Modified Input Data Set with Additional Columns*

VIEWTABLE: summary

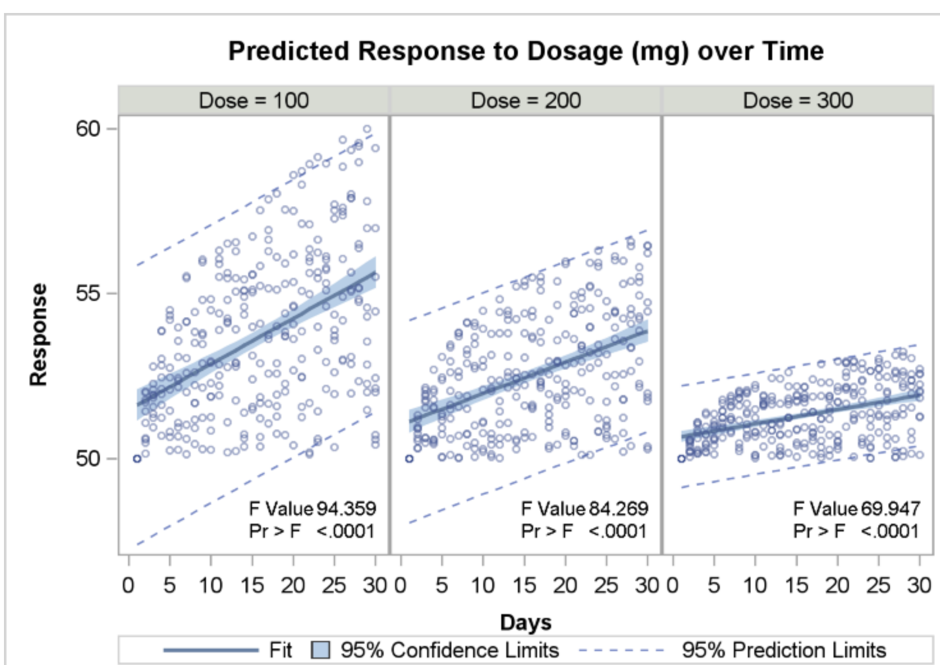
	Cylinders	Origin	Type	Mean	Nobs	cellCyl	cellOrigin	cellMean	cellNobs
1	4	Asia	SUV	21.4	5	4	Asia	25.8	57
2	4	Asia	Sedan	26.5	49	4	Europe	23.4	18
3	4	Asia	Truck	22.3	3	4	USA	24.0	34
4	4	Europe	SUV	.	0	6	Asia	18.3	60
5	4	Europe	Sedan	23.4	18	6	Europe	18.8	38
6	4	Europe	Truck	.	0	6	USA	18.5	61
7	4	USA	SUV	20.5	2	8	Asia	15.2	10
8	4	USA	Sedan	24.7	29	8	Europe	16.2	24
9	4	USA	Truck	20.0	3	8	USA	15.5	35
10	6	Asia	SUV	17.2	15	.		.	.
11	6	Asia	Sedan	19.0	41	.		.	.
12	6	Asia	Truck	15.5	4	.		.	.
13	6	Europe	SUV	16.0	4	.		.	.

The SUMMARY data set can now be used to render a graph from template PANEL:

```
ods listing style=statistical;
proc sgrender data=summary template=panel;
run;
```



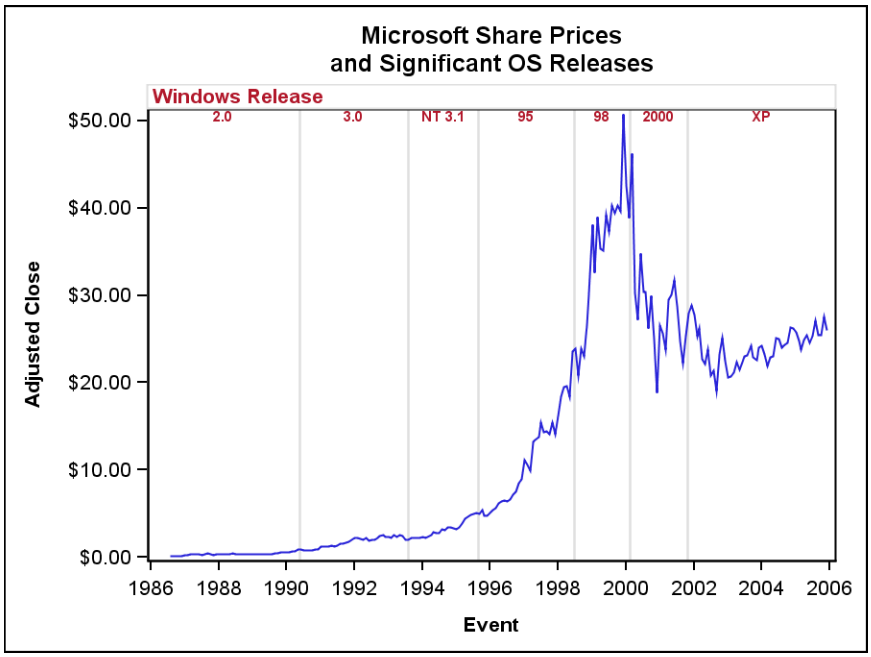
The following figure shows another example of adding insets to a classification panel. The complete code for this output is presented inChapter 11, “Using Classification Panels,” on page 185.



Creating an Axis-Aligned Inset with a Block Plot

Sometimes you want an inset to provide information on values along an axis. In the following example, "events" have been defined over time and the inset information at the top of the plot provides information about these events.

The example uses a BLOCKPLOT statement, which creates one or more strips of rectangular blocks containing text values. The width of each block corresponds to specified numeric intervals along the X-axis.



The following template code defines a template named BLOCKPLOT1, which is used to create this graph. In the template code, the block plot is overlaid with a series plot to create an axis-aligned inset. Notice that the BLOCKPLOT statement requires two input columns: on for the X= argument and another for the BLOCK= argument. The BLOCK= transition points control the boundary of each block and the text that is displayed. The range of the X= values between two consecutive block transition points determine the width of each block.

```
proc template;
  define statgraph blockplot1;
    begingraph;
      entrytitle "Microsoft Share Prices";
      entrytitle "and Significant OS Releases";
      layout overlay;
        blockplot x=event block=release / display=(outline values label)
          valuevalign=top valuehalign=center labelposition=top
          valueattrs=GraphDataText(weight=bold
                                   color=GraphData2:contrastcolor)
          labelattrs=GraphValueText(weight=bold
                                    color=GraphData2:contrastcolor)
          outlineattrs=(color=GraphGridLines:color);
        seriesplot x=date y=adjClose / lineattrs=GraphData1;
```

```
      endlayout;
    endgraph;
  end;
run;
```

The BLOCKPLOT statement supports many options for controlling the content, position, and appearance of the blocks and text information.

DISPLAY= (<OUTLINE> <FILL> <VALUES> <LABEL>)
: specifies the features to display

VALUEVALIGN= TOP | CENTER | BOTTOM
: specifies the vertical position of the text values within the blocks

VALUEHALIGN=LEFT | CENTER | RIGHT | START
: specifies the horizontal position of the text values within the blocks

LABELPOSITION= LEFT | RIGHT | TOP | BOTTOM
: specifies a position for the block label that applies to the block values

VALUEATTRS=*style-element*
: specifies font properties for block the values

LABELATTRS=*style-element*
: specifies font properties for the block label

The input data that is used with the BLOCKPLOT1 template must contain data for both plots. The simplest way to construct the appropriate data is to create separate X= variables for the block plot (EVENT) and the scatter plot (DATE).

VIEWTABLE: events

	Stock	Date	AdjClose	Event	Release
1	Microsoft	01DEC05	$25.96	09DEC87	2.0
2	Microsoft	01NOV05	$27.48	22MAY90	3.0
3	Microsoft	03OCT05	$25.44	01AUG93	NT 3.1
4	Microsoft	01SEP05	$25.47	24AUG95	95
5	Microsoft	01AUG05	$27.10	25JUN98	98
6	Microsoft	01JUL05	$25.28	17FEB00	2000
7	Microsoft	01JUN05	$24.52	25OCT01	XP
8	Microsoft	02MAY05	$25.46	.	
9	Microsoft	01APR05	$24.89	.	
10	Microsoft	01MAR05	$23.78	.	
11	Microsoft	01FEB05	$24.75	.	
12	Microsoft	03JAN05	$25.77	.	

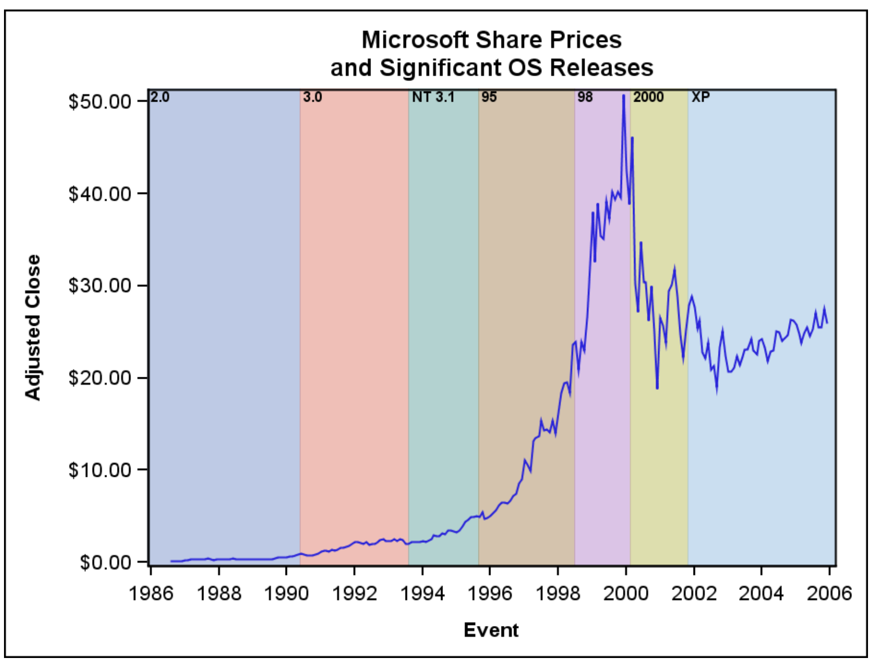
```
/* data for block plot - ordered by event */
data MSevents;
  input Event date9. Release $7.;
  label Release="Windows Release";
  format Event date.;
datalines;
09dec1987 2.0
22may1990 3.0
01aug1993 NT 3.1
24aug1995 95
25jun1998 98
17feb2000 2000
25oct2001 XP
run;
```

```
/* non-match merge of input data */
data events;
  merge sashelp.stocks(keep=stock date adjClose
                       where=(stock="Microsoft"))
        MSevents;
run;

proc sgrender data=events template=blockplot1;
run;
```

The next example shows a different way to present the same information. Here the outlines are removed and the blocks are filled with colors. The example uses the following BLOCKPLOT options:

FILLTYPE= MULTICOLOR | ALTERNATE
: specifies how the blocks are filled

DATATRANSPARENCY= *number*
: specifies the degree of the transparency of the block fill and outline. The range for *number* is from 0 (opaque) to 1 (entirely transparent).



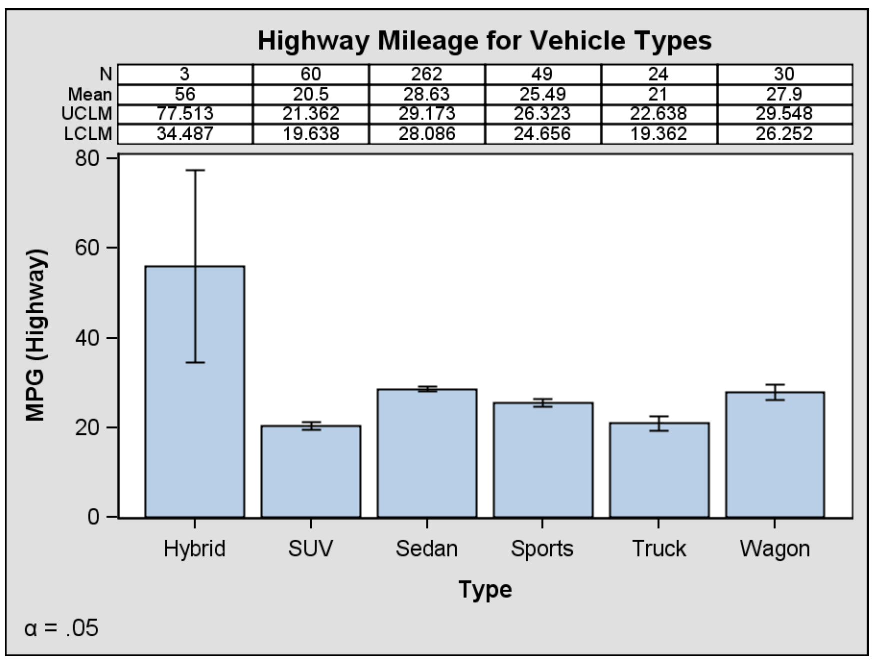
In this example, the FILLATTRS=MULTICOLOR setting ensures that the colors will be obtained from the GraphData1 to GraphDataN style elements of the current style. Transparency is added to fade the colors. The block label "Windows Release" is suppressed, and the horizontal alignment of the block values is shifted to the left.

```
proc template;
  define statgraph blockplot1a;
    begingraph;
      entrytitle "Microsoft Share Prices";
      entrytitle "and Significant OS Releases";
      layout overlay;
        blockplot x=event block=release / display=(fill values)
            valuevalign=top  valuehalign=left
            valueattrs=GraphDataText(weight=bold)
            filltype=multicolor
```

```
            datatransparency=.5;
          seriesplot x=date y=adjClose / lineattrs=GraphData1;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=events template=blockplot1a;
run;
```

The BLOCKPLOT statement can also create a table of inset information where the columns are centered on discrete values along the X-axis and the rows represent different statistics for each value of the X= variable. This technique for displaying inset information is possible for plots with a discrete X-axis, such as box plots and bar charts. The BLOCKPLOT statement supports a CLASS=*variable* option that creates a separate block plot for each unique value of the CLASS= variable. Notice that in this example, the block plot is not located inside the OVERLAY layout but in its own cell of a LATTICE layout.

Figure 16.2 *Inset Displayed as a Multi-row Table*



To create this graph, some data set up is necessary. First, we can use PROC SUMMARY to create the summarized input data for the block plot and the bar chart:

```
/* Create summarized data with desired statistics */
proc summary data=sashelp.cars nway alpha=.05;
  class type;
  var mpg_highway;
  output out=stats(drop=_FREQ_ _TYPE_) n=N mean=Mean uclm=UCLM lclm=LCLM;
run;
```

The columns for TYPE, MEAN, UCLM, and LCLM will be used by a BARCHARTPARM statement.

VIEWTABLE: stats

	Type	N	Mean	UCLM	LCLM
1	Hybrid	3	56	77.513263649	34.486736351
2	SUV	60	20.5	21.362039446	19.637960554
3	Sedan	262	28.62977	29.173242208	28.086299776
4	Sports	49	25.4898	26.323355793	24.656236043
5	Truck	24	21	22.637785892	19.362214108
6	Wagon	30	27.9	29.547750098	26.252249902

However, the columns that are required for the BLOCKPLOT statement are not the same as those for the BARCHARTPARM statement. The information must first be transposed.

```
/* Transpose data for use with BLOCKPLOT */
proc transpose data=stats
      out=blockstats(drop=_label_
                     rename=(type=type2 _name_=statname col1=stat));
  by type;
  var n mean uclm lclm;
run;
```

The SAS log displays the following note when the procedure code is submitted:

```
NOTE: There were 6 observations read from the data set WORK.STATS.
NOTE: The data set WORK.BLOCKSTATS has 24 observations and 3 variables.
```

VIEWTABLE: blockstats

	type2	statname	stat
1	Hybrid	N	3
2	Hybrid	Mean	56
3	Hybrid	UCLM	77.513263649
4	Hybrid	LCLM	34.486736351
5	SUV	N	60
6	SUV	Mean	20.5
7	SUV	UCLM	21.362039446
8	SUV	LCLM	19.637960554
9	Sedan	N	262
10	Sedan	Mean	28.629770992
11	Sedan	UCLM	29.173242208
12	Sedan	LCLM	28.086299776

Finally, the data for the BARCHARTPARM and BLOCKPLOT statements must be non-match merged into one input data set. Note that the TYPE and TYPE2 variables must be distinct variables.

```
/* Combine summary data for BARCHARTPARM with tabular data for BLOCKPLOT */
data all;
  merge stats blockstats;
run;
```

```
NOTE: There were 6 observations read from the data set WORK.STATS.
NOTE: There were 24 observations read from the data set WORK.BLOCKSTATS.
NOTE: The data set WORK.ALL has 24 observations and 8 variables.
```

VIEWTABLE: all

	Type	N	Mean	UCLM	LCLM	type2	statname	stat
1	Hybrid	3	56	77.513263649	34.486736351	Hybrid	N	3
2	SUV	60	20.5	21.362039446	19.637960554	Hybrid	Mean	56
3	Sedan	262	28.629770992	29.173242208	28.086299776	Hybrid	UCLM	77.513263649
4	Sports	49	25.489795918	26.323355793	24.656236043	Hybrid	LCLM	34.486736351
5	Truck	24	21	22.637785892	19.362214108	SUV	N	60
6	Wagon	30	27.9	29.547750098	26.252249902	SUV	Mean	20.5
7		.	.	.	.	SUV	UCLM	21.362039446
8		.	.	.	.	SUV	LCLM	19.637960554
9		.	.	.	.	Sedan	N	262
10		.	.	.	.	Sedan	Mean	28.629770992
11		.	.	.	.	Sedan	UCLM	29.173242208
12		.	.	.	.	Sedan	LCLM	28.086299776
13		.	.	.	.	Sports	N	49

The template for this graph uses a BLOCKPLOT statement with X=TYPE2 and BLOCK=STAT. By default, if there are adjacent repeated values for the BLOCK= column, a new block does not begin until the BLOCK value changes. The CLASS=STATNAME setting creates a row (block plot) for each value of the TYPE2 variable. By default, the values of the CLASS= variable appear as row labels external to the block plot.

The ROWWEIGHTS = option for the LATTICE layout governs the relative amount of vertical space that is allotted to the BLOCKPLOT (15%) and the BARCHARTPARM (85%). This would have to be changed if you have a much larger or smaller number of rows in the statistics table.

```
proc template;
  define statgraph blockplot2;
    begingraph;
      entrytitle "Highway Mileage for Vehicle Types";
      entryfootnote halign=left {unicode alpha} " = .05";
      layout lattice / columns=1 rowweights=(.15 .85);

        blockplot x=type2  block=stat /  class=statname
           includemissingclass=false
           display=(values label outline) valuehalign=center
           labelattrs=GraphDataText valueattrs=GraphDataText;

        barchartparm x=type y=mean / errorlower=lclm errorupper=uclm;

      endlayout;
    endgraph;
  end;
run;

ods listing style=default;
proc sgrender data=all template=blockplot2;
run;
```

Chapter 17
Managing the Graph Appearance with Styles

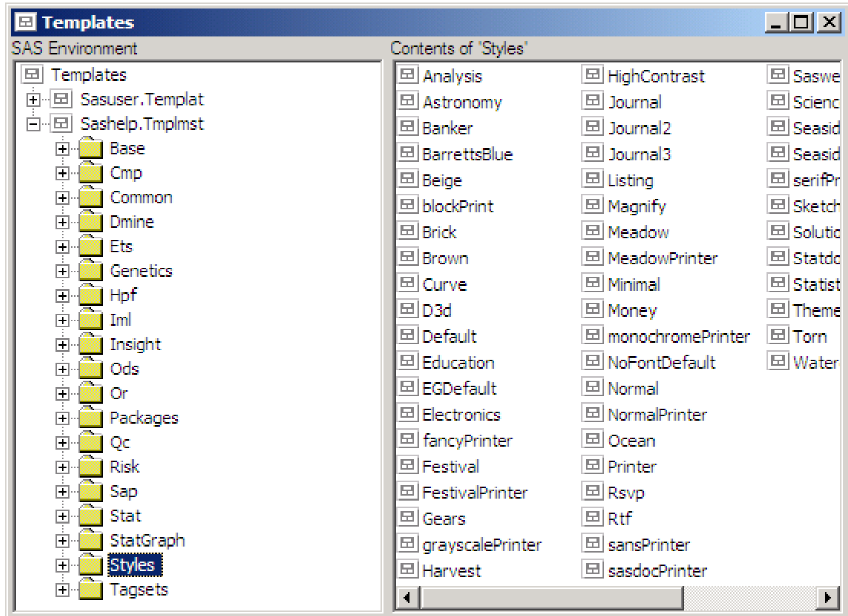
ODS Style Templates

It is often useful to create graphs with specific visual characteristics that do not have to be hard coded into the GTL for every graph that you create. For example, you might want to modify settings for the following graph features:

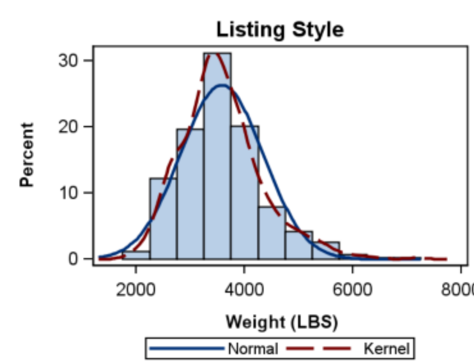
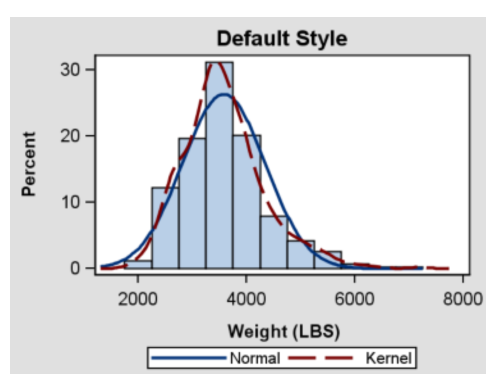
- font or font sizes
- line or marker properties
- colors
- display features for box plots, histograms, contours, and other chart types
- a combination of features that are related to a publication or corporate presentation scheme.

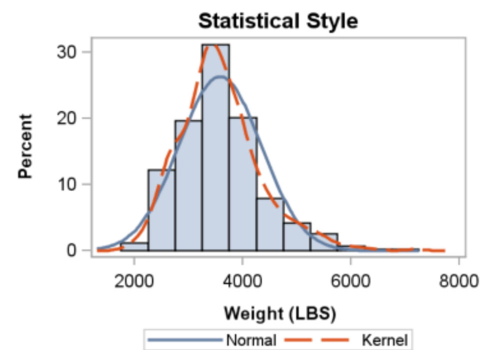
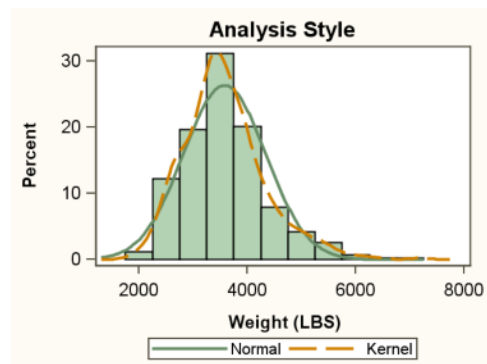
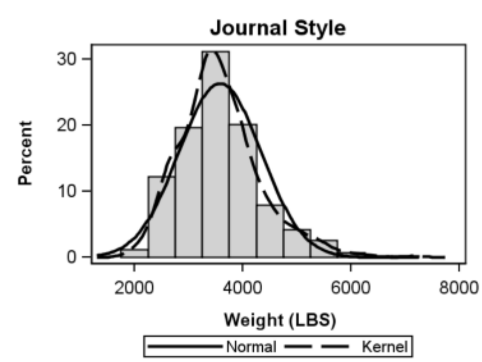
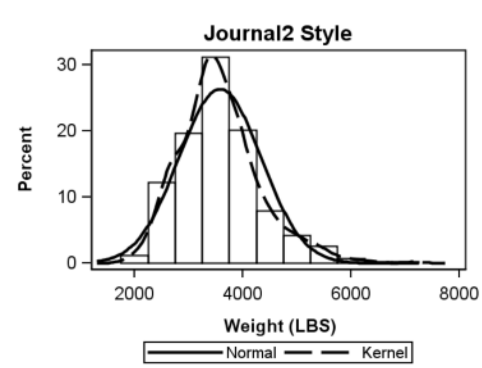
Because the default properties of nearly all GTL appearance-related options are obtained from the current style (see Chapter 6, “Managing Graph Appearance: General Principles,” on page 83), modifying an existing style is often the best way to enforce a certain look-and-feel across many graphs.

Similar to graphics templates, ODS style templates are stored in SAS item stores. All SAS-supplied styles are located in the STYLES directory of the SASHELP.TMPLMST item store. Templates can be viewed from the Templates window (ODSTEMPLATE command). The template source can be viewed by opening any template.



Although an ODS style can be constructed from scratch, you will find it much simpler to identify a style that is fairly close to what you want and make limited changes to it. The following styles are recommended starting points.

Style	**Example**
LISTING • white background • white wall • sans-serif fonts • color used for lines, markers, and filled areas • colors and other features are the same as the DEFAULT style	
DEFAULT • gray background • white wall • sans-serif fonts	

Style	Example
STATISTICAL • white background • white wall • sans-serif fonts • contrasting color scheme of blues, reds, greens for markers, lines, and filled areas	
ANALYSIS • light tan background • white wall • sans-serif fonts • muted color scheme of tans, greens, yellows, oranges and browns for lines, markers, and filled areas	
JOURNAL • white background • white wall • sans-serif fonts • gray-scale color scheme for markers, lines, and filled areas	
JOURNAL2 • white background • white wall • sans-serif fonts • black-only scheme for markers and lines • no filled areas—a minimal ink style	

Changing Fonts in a Style Template

Notice that all of the recommended styles use Sans-Serif fonts. The following example shows how to create a custom style that uses Serif fonts instead. The example uses the STATISTICAL style as a starting point (parent) for the custom style.

The following PROC TEMPLATE code shows the beginning of the style definition for the STATISTICAL style, which is delivered with the SAS System. Only the code to be modified is shown.

```
proc template;
define style Styles.Statistical;
parent = styles.default;

style fonts /
 'TitleFont2'=("<sans-serif>, <MTsans-serif>, Helvetica,Helv",2,bold)
 'TitleFont'=("<sans-serif>, <MTsans-serif>, Helvetica,Helv",3,bold)
 'StrongFont'=("<sans-serif>, <MTsans-serif>, Helvetica,Helv",2,bold)
 'EmphasisFont'=("<sans-serif>, <MTsans-serif>, Helvetica,Helv",2,italic)
 'FixedFont'=("<monospace>, Courier",2)
 'BatchFixedFont'=("SAS Monospace, <monospace>, Courier, monospace",2)
 'FixedHeadingFont'=("<monospace>, Courier, monospace",2)
 'FixedStrongFont'=("<monospace>, Courier, monospace",2,bold)
 'FixedEmphasisFont'=("<monospace>, Courier, monospace",2,italic)
 'headingEmphasisFont'=("<sans-serif>, <MTsans-serif>, Helvetica,
       Helv",2,bold italic)
 'headingFont'=("<sans-serif>, <MTsans-serif>, Helvetica, Helv",2,bold)
 'docFont'=("<sans-serif>, <MTsans-serif>, Helvetica, Helv",2);

style GraphFonts /
 'GraphDataFont'=("<sans-serif>, <MTsans-serif>",7pt)
 'GraphUnicodeFont'=("<MTsans-serif-unicode>",9pt)
 'GraphValueFont'=("<sans-serif>, <MTsans-serif>",9pt)
 'GraphLabelFont'=("<sans-serif>, <MTsans-serif>",10pt)
 'GraphFootnoteFont'=("<sans-serif>, <MTsans-serif>",10pt,italic)
 'GraphTitleFont'=("<sans-serif>, <MTsans-serif>",11pt,bold)
 'GraphAnnoFont'=("<sans-serif>, <MTsans-serif>",10pt);

  /* more code */

end;
run;
```

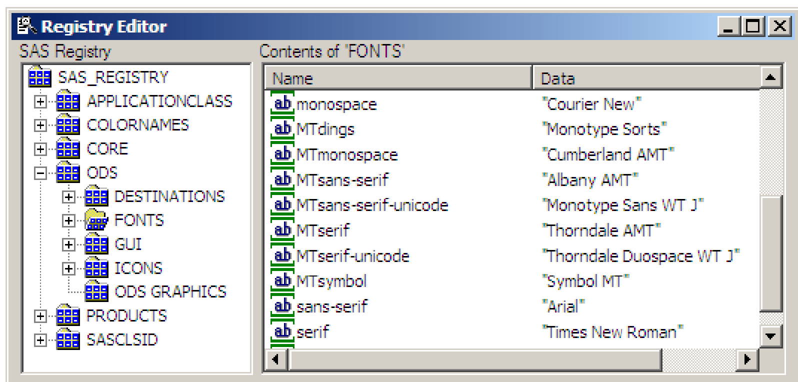
We will make the following changes:

- Assign a name to a new style that identifies STATISTICAL as its parent style. It is recommended that you create a new style of a different name so that access to the existing style is not blocked. See discussion under "Controlling ODS Search Paths" on page 304.
- Change the Fonts style element (affects tables) so that it uses Serif fonts.
- Change the GraphFonts style element (affects graphs) so that it uses Serif fonts.

Two style elements govern all fonts in a style: the Fonts element governs tables, and the GraphFonts element governs graphs. When changing fonts in a style, be sure to make consistent changes to both elements. In this case, we want to change from a sans-serif font to a serif font. You can also change font size, weight, and style.

In style templates, the name of a font family normally appears as a quoted string. However, ODS also supports an indirect reference to a font family. When a font name appears between less than and greater than symbols, such as <*sans-serif*>, it means that the font family *sans-serif* is defined in the SAS Registry. For the Windows Release of SAS, here are some of the registry keys and values that are stored under **ODS** ⇨ **Fonts**:

Display 17.1 *Registry Keys and Values for Fonts in the Windows Release of SAS*



The registry definition of MTsans-serif and MTserif refer to TrueType fonts that are shipped with SAS and are similar to "Arial" and "Times New Roman." These "MT" fonts (short for Monotype) can be used on any computer where SAS is installed. A specific font family such as "Verdana" could be used instead. Note that fonts are normally listed in a "most-specific" to "most generic" order so that reasonable substitution can be made when a font cannot be located on the current computer.

Notice that several graph fonts affect different parts of a graph. The following table shows some but not all features that are affected by the graph fonts:

GraphTitleFont	Used for all titles of the graph. Typically the largest font.
GraphFootnoteFont	Used for all footnotes. Typically smaller than the titles. Sometime footnotes are italicized.
GraphLabelFont	Used for axis labels and legend titles. Generally smaller than titles.
GraphValueFont	Used for axis tick values and legend entries. Generally smaller than labels.
GraphDataFont	Used for text where minimum size is necessary (such as point labels).
GraphUnicodeFont	Used for adding special glyphs (for example, α , ±, €) to text in the graph (see Chapter 7, "Adding and Changing Text in a Graph," on page 101).
GraphAnnoFont	Default font for text added as annotation in the ODS Graphics Editor.

The *SAS Output Delivery System: User's Guide* provides information and examples of all predefined style elements and attributes.

In our example, we will name our modified style template *SerifStatistical* , change all occurrences of sans-serif to serif, change all occurrences of MTsans-serif to MTserif, and change Helvetica and Helv (sans-serif fonts) to Times (a serif font):

```
proc template;
 define style Styles.SerifStatistical ;
  parent = styles.statistical;
   style fonts /
    'TitleFont2'=(" <serif> , <MTserif> , Times ",2,bold)
    'TitleFont'=(" <serif> , <MTserif> , Times ",3,bold)
    'StrongFont'=(" <serif> , <MTserif> , Times ",2,bold)
    'EmphasisFont'=(" <serif> , <MTserif> , Times ",2,italic)
    'FixedFont'=("<monospace>, Courier",2)
    'BatchFixedFont'=("SAS Monospace, <monospace>, Courier, monospace",2)
    'FixedHeadingFont'=("<monospace>, Courier, monospace",2)
    'FixedStrongFont'=("<monospace>, Courier, monospace",2,bold)
    'FixedEmphasisFont'=("<monospace>, Courier, monospace",2,italic)
    'headingEmphasisFont'=(" <serif> , <MTserif> , Times ",
             2,bold italic)
    'headingFont'=(" <serif> , <MTserif> , Times ",2,bold)
    'docFont'=(" <serif> , <MTserif> , Times ",2);

   style GraphFonts /
    'GraphTitleFont'=(" <serif> , <MTserif> ",11pt,bold)
    'GraphFootnoteFont'=(" <serif> , <MTserif> ",10pt,italic)
    'GraphLabelFont'=(" <serif> , <MTserif> ",10pt)
    'GraphValueFont'=(" <serif> , <MTserif> ",9pt)
    'GraphDataFont'=(" <serif> , <MTserif> ",7pt)
    'GraphUnicodeFont'=(" <MTserif-unicode> ",9pt)
    'GraphAnnoFont'=(" <serif> , <MTserif> ",10pt);
 end;
run;
```

Note: By assigning the parent to STYLES.STATISTICAL, we need to change only two style elements. The rest of the elements are inherited.

Controlling ODS Search Paths

Before you submit your modified style definition, you should consider whether this style is for your use only or whether you want to share it with others. Your decision will determine where you store the style.

The ODS PATH statement determines the read and write locations for SAS item store templates.

```
ods path show;
```

```
Current ODS PATH list is:
1. SASUSER.TEMPLAT(UPDATE)
2. SASHELP.TMPLMST(READ)
```

By default, modified templates are stored in SASUSER.TEMPLAT, which is appropriate for your personal use. To store a modified template in this default location, you will see the following note in the SAS Log after submitting the PROC TEMPLATE code:

```
NOTE: STYLE 'Styles.SerifStatistical' has been saved to:
SASUSER.TEMPLAT
```

You can then run your program with the new style:

```
ods rtf style=serifStatistical ;
ods graphics on;
proc reg data=sashelp.class;
  model weight=height;
quit;
ods rtf close;
```

To save a modified template to a location where others can access it, you cannot use the default SASUSER.TEMPLAT location. Rather, store the template in a different library, using the ODS PATH statement to set the search path:

```
libname common "u:\ODS_templates";

ods path common.dept(update)
         sasuser.templat(update)
         sashelp.tmpmst(read);
```

This ODS PATH statement establishes a new search path. The first itemstore (common.dept) can be updated and will contain the new template (Styles.SerifStatistical). It is important to include SASHELP.TMPLMST in the path because the inherited parent style (Styles.Default) is in SASHELP.

After setting this new search path, you will see the following note in the SAS Log when you submit the PROC TEMPLATE code:

```
NOTE: STYLE 'Styles.SerifStatistical' has been saved to:
COMMON.DEPT
```

For others to access this style definition, everyone will have to precede their programs with the following code:

```
libname common "u:\ODS_templates" access=readonly;
ods path sasuser.templat(update)
         common.dept(read)
         sashelp.tmpmst(read);
```

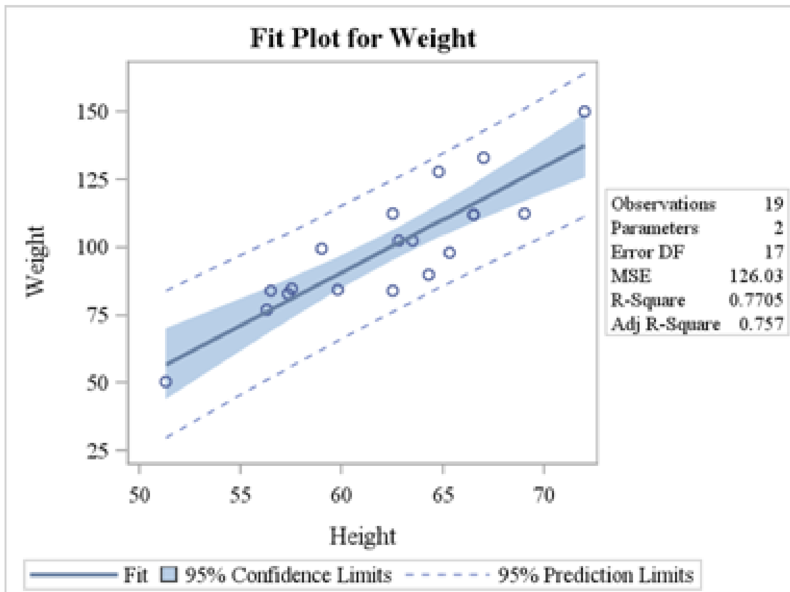
They can then run their program with the new style:

```
ods rtf style=serifStatistical;
ods graphics on;
proc reg data=sashelp.class;
  model weight=height;
quit;
ods rtf close;
```

The following figure shows a table and graph from the output.

Model: MODEL1
Dependent Variable: Weight

Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	-143.02692	32.27459	-4.43	0.0004
Height	1	3.89903	0.51609	7.55	<.0001



Changing Boxplot Display

The SAS System defines many graphical style elements. Some have a very narrow scope, such as those that control the display of box plots. Using these style elements as a starting point, you can change the style attribute values to achieve a very different appearance for your box plots. The same is true for histograms, contours, and some other plot types.

Using the DEFAULT style for an example, here is a portion of the style definition for elements that are related to box plots:

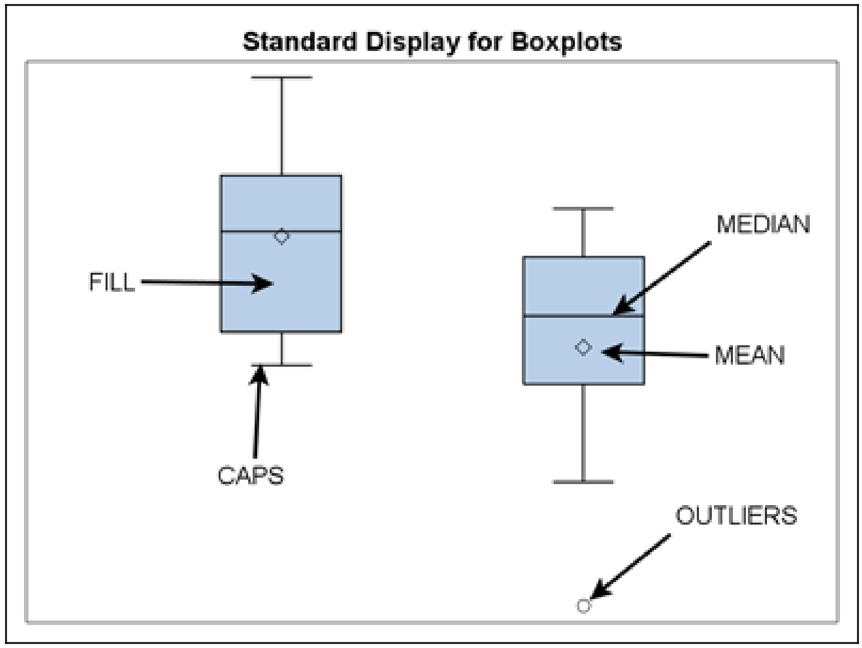
```
proc template;
  define style Styles.Default;
  ...
  style GraphBox /
     capstyle = "serif"
     connect = "mean"
     displayopts = "fill caps median mean outliers";
  style GraphBoxMean / ... ;
  style GraphBoxMedian / ... ;
  style GraphBoxOutlier / ... ;
  style GraphBoxWhisker / ... ;
```

```
   ...
   end;
run;
```

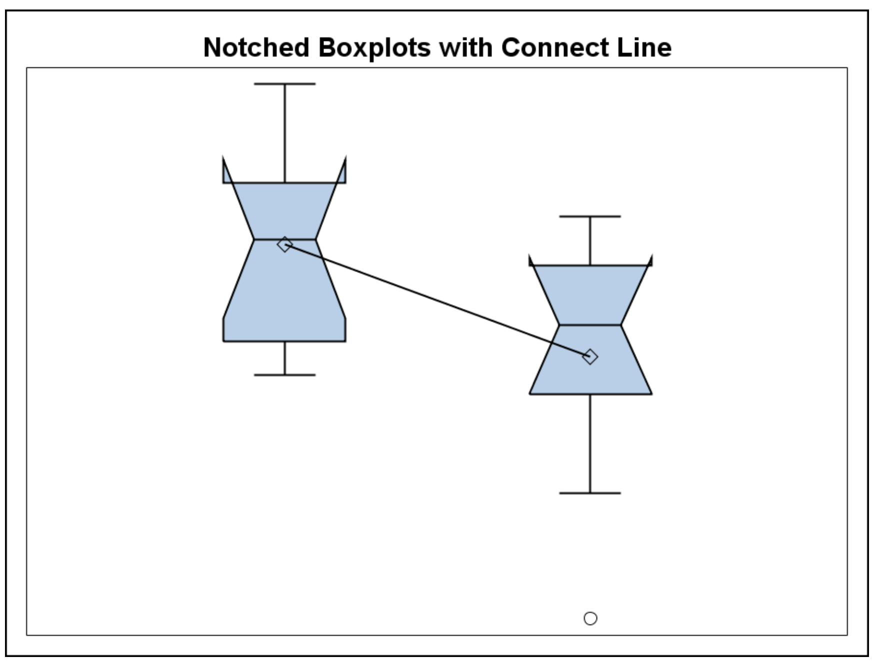
Style Element	Purpose
GraphBox	general box plot properties (see the next table)
GraphBoxMean	marker properties of mean marker (MARKERSYMBOL= , MARKERSIZE=, CONTRASTCOLOR=)
GraphBoxMedian	line properties of the median line (LINESTYLE=, LINETHICKNESS=, CONTRASTCOLOR=)
GraphBoxOutlier	marker properties of outliers (MARKERSYMBOL=, MARKERSIZE=, CONTRASTCOLOR=)
GraphBoxWhisker	line properties of whiskers and caps (LINESTYLE=, LINETHICKNESS=, CONTRASTCOLOR=)

Attributes and Values for the GraphBox Style Element		
Attribute	**Value(s)**	**Description**
CONNECT=	"MEAN" | "MEDIAN" | "Q1" | "Q3" | "MIN" | "MAX"	statistic to connect with line
CAPSTYLE=	"SERIF" | "LINE" | "BRACKET"	shape at ends of whiskers
DISPLAYOPTS=	"<CAPS>	show caps at end of whiskers
	<FILL>	show filled boxes
	<MEAN>	show a marker for the mean
	<MEDIAN>	show a line for the median
	<OUTLIERS>	show markers for the outliers
	<CONNECT>	show line connecting same statistic on multiple boxes
	<NOTCHES>"	show notched boxes

The Displayopts attribute of GraphBox lists the general features that will be displayed. The following diagram shows the standard display for box plots, as defined by the DEFAULT style. The keywords that are related to the appearance features are annotated:

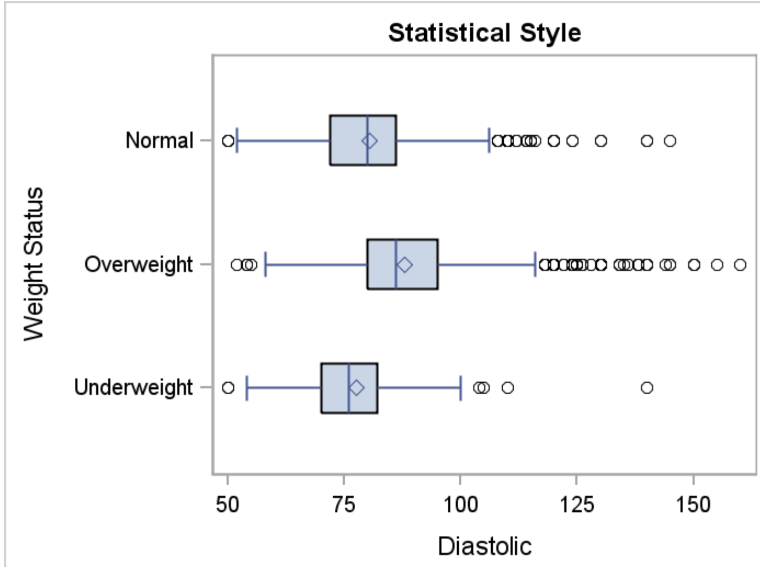


The two display options that are not the default are CONNECT (show connect lines) and NOTCHES.



The STATISTICAL style is derived from the DEFAULT style and inherits the GraphBox element from the parent DEFAULT style. The following code generates a box plot for the STATISTICAL style:

```
title "Statistical Style";
ods listing style=statistical;

proc sgplot data=sashelp.heart;
  hbox diastolic /
       category=weight_status;
run;
```



For this example, we want to change the following attributes on the default box plot:

- By default, serif caps are displayed at the end of the fences. We want to remove those caps from the fence lines.
- By default, the boxes are filled. We want to display empty, notched boxes.
- By default, the mean values are represented by hollow diamonds. We want to display filled diamonds and slightly reduce their size.
- By default, the marker symbols for the outliers are hollow black circles. We want to change the size and shape of the marker symbols, and again reduce their size.

To make these changes, we can derive a new style from the STATISTICAL style and set the attributes that we want to change. Any attribute settings that we do not change will be inherited from the parent STATISTICAL style. The following style definition will effect the desired changes:

```
proc template;
  define style Styles.Boxplot;
  parent = styles.statistical;
    style GraphBox from GraphBox /
      capstyle = " line "
      displayopts = " caps median mean outliers notches ";
    style GraphBoxMean from GraphBoxMean /
      markersymbol=" diamondfilled "
      contrastcolor=GraphColors('gcdata1")
      markersize = 5px;
    style GraphOutlier from GraphOutlier /
      markersize = 5px
      markersymbol = " x "
      contrastcolor = GraphColors(" gcdata2 ");
    end;
run;
```

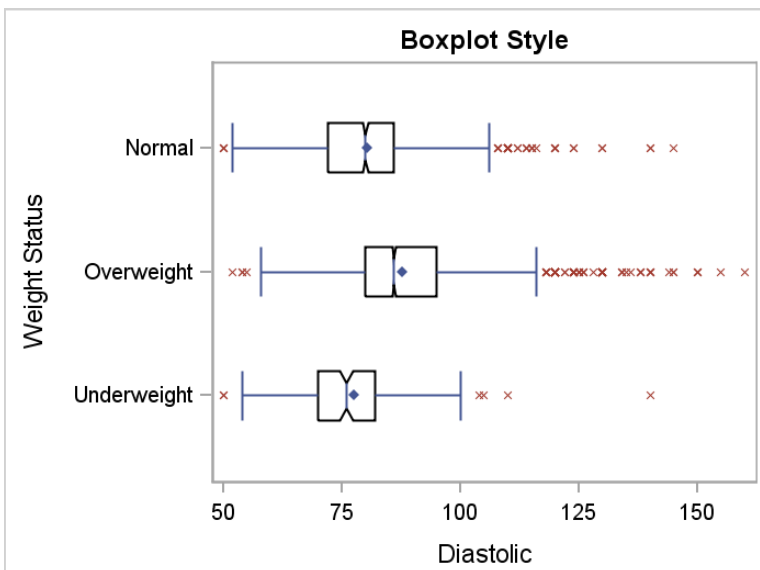
- The DEFINE STYLE statement assigns the name BOXPLOT to our new style, and sets the STATISTICAL style as the parent style.
- On the GraphBox style element, the CAPSTYLE= attribute is set to LINE, which removes the serif caps from the end of the fences. The DISPLAYOPTS= attribute drops

the FILL value from the display list and adds the NOTCHES value; these changes determine that the graph will display empty, notched boxes.

- On the GraphBoxMean style element, the marker symbol is changed to a filled diamond and the marker size is reduced to 5 pixels (the default is 9 pixels). The CONTRASTCOLOR= attribute is set to GCDATA1 (the default is GCDATA).
- On the GraphBoxOutlier style element, the marker symbol is changed to an X and the marker size is reduced to 5 pixels (the default is 7 pixels). The CONTRASTCOLOR= attribute is set to GCDATA2 (the default is GCOUTLIER).

The following code generates a box plot for the BOXPLOT style:

```
title "Boxplot Style";
ods listing style=styles.boxplot;

proc sgplot data=sashelp.heart;
  hbox diastolic /
       category=weight_status;
run;
```



When making such style changes remember that you are affecting all box plot displays for all procedures that produce box plots when this style is in effect. It is possible to change the box plot appearance for specific procedures, but to do this, a specific graph template must be modified, not a style template.

For a comprehensive description of the style elements affecting ODS graphics, see the section for the style elements affecting template-based graphics in the Appendix for ODS Style Elements of the *SAS Output Delivery System: User's Guide*.

Chapter 18
Executing Graph Templates

Techniques for Executing Templates

Compiled graph templates can be executed using either the PROC SGRENDER statement or a DATA step. Both techniques offer the same functionality but differ in their syntax. The SGRENDER syntax is simpler, but any required data manipulations must be completed before the PROC SGRENDER statement is used. The DATA step syntax is more complex, but it can integrate data manipulations with the graph execution.

Both PROC SGRENDER and a DATA step can be used to

- specify the input template
- specify the input data set
- associate a label with input variable(s) using a LABEL statement
- associate a format with input variable(s) using a FORMAT statement
- filter input data using a WHERE statement or WHERE= input data set option
- assign values to dynamic variables for substitution in the template
- name the output data object
- label the output data object.

The following sections show how to use both SGRENDER and a DATA step to generate graphs from compiled GTL templates.

Minimal Required Syntax

Consider the following simple GTL template definition:

```
proc template;
define statgraph mygraphs.scatter;
  begingraph;
    layout overlay;
      scatterplot X=height Y=weight;
    endlayout;
  endgraph;
end;
run;
```

Both PROC SGRENDER and the DATA step can be used to execute this template. Both techniques minimally require you to specify the input data source and the template name. Behind the scenes in both cases, an ODS data object is populated and bound to the template. The data object is then passed to a graph renderer, which processes the data and graph request to produce an output image.

The PROC SGRENDER syntax is simple. It uses the DATA= option to specify the data source and the TEMPLATE= option to specify the template to use for rendering the graph:

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

The DATA step syntax is slightly more complex. To execute a GTL template, the DATA step FILE and PUT statements provide syntax that is specific to ODS. You must minimally specify the following:

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

- The DATA step uses keyword _NULL_ for the data set name so that the DATA step executes without writing observations or variables to an output data set. The input data source is defined with a SET statement. This approach is appropriate in the current example, but the input data source can be defined with any appropriate DATA step syntax (INPUT with DATALINES, INPUT with INFILE, SET, MERGE, UPDATE, and so on).
- FILE PRINT ODS directs output to ODS. PRINT is a reserved fileref that is required when executing a GTL template. It directs output that is produced by any PUT statements to the same file as output that is produced by SAS procedures. The TEMPLATE= specification is required to specify the input template name.
- The PUT _ODS_ statement, also required, writes the necessary variables to the output object for each execution of the DATA step.

Note: The necessary variables for the output data object are the ones defined by the graph template (in this case, HEIGHT and WEIGHT), not the input data source. As with other DATA step or procedure processing, if you know exactly which variables the template uses, you can restrict the input variables with DROP= or KEEP= input data set options for slightly more efficient processing.

Managing the Input Data

Filtering the Input Data

If you do not need all of the variables or all of the data values from the input data source, you can use WHERE statements or input SAS dataset options (for example, OBS= or WHERE=) to control the observations that are processed. The filtering techniques can be used whether the GTL template is executed with PROC SGRENDER or with a DATA step.

In the following example, the first PROC SGRENDER uses a WHERE statement to select only female observations for the graph. The second PROC SGRENDER uses the OBS= input dataset option to limit the number of observations used in the graph.

```
/* plot only observations for females */
proc sgrender data=sashelp.class template=mygraphs.scatter;
   where sex="F";
run;

/* test the template */
proc sgrender data=sashelp.class( obs=5 )
              template=mygraphs.scatter;
run;
```

Performing Data Transformations

When using PROC SGRENDER, any required data transformations or computations must take place before a template is executed. The transformations therefore require an intermediate step. For example, the following code performs data transformations on the HEIGHT and WEIGHT variables that are in the data set SASHELP.CLASS. The transformations are stored in a temporary data set named CLASS, which is then used on PROC SGRENDER to produce a graph:

```
data class;
  set sashelp.class;
  height=height*2.54;
  weight=weight*.45;
  label height="Height in CM" weight="Weight in KG";
run;
proc sgrender data=class template=mygraphs.scatter;
run;
```

When executing a template with a DATA step, the same DATA step that builds the data object can perform any required data transformations or computations. An intermediate data set is not needed. This next example produces the same graph that the previous example produced with PROC SGRENDER:

```
data _null_;
  set sashelp.class;
  height=height*2.54;
  weight=weight*.45;
  label height="Height in CM" weight="Weight in KG";
  file print ods=(template="mygraphs.scatter");
  put _ods_;
run;
```

Initializing Template Dynamics and Macro Variables

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

You can create new macro variables or use the automatic macro variables that are defined in SAS, such as the system date and time value (SYSDATE). Both types of macro variables must be declared before they can be referenced. Whereas automatic macro variables do not require initialization, you must initialize any macro variables that you create with the variable declarations. The macro variable values are obtained from the current symbol table (local or global), so SAS resolves their values according to the context in which they are used.

The following template declares the dynamic variables XVAR and YVAR, and the macro variables STUDY and SYSDATE:

```
proc template;
define statgraph mygraphs.regfit;
  dynamic XVAR YVAR;
  mvar STUDY SYSDATE;
  begingraph;
    entrytitle "Regression fit for Model " YVAR " = " XVAR ;
    entryfootnote halign=left STUDY halign=right SYSDATE ;
    layout overlay;
       scatterplot X=XVAR Y=YVAR ;
       regressionplot X=XVAR Y=YVAR ;
    endlayout;
  endgraph;
end;
run;
```

- The DYNAMIC statement declares dynamic variables XVAR and YVAR. On the statements that later execute this template, you must initialize these dynamics by assigning them to variables from the input data source so that they have values at runtime.
- The ENTRYTITLE statement concatenates dynamic variables XVAR and YVAR into a string that will be displayed as the graph title. At runtime, the dynamics will be replaced by the names of the variables that are assigned to the dynamics when they are initialized.
- The SCATTERPLOT and REGRESSIONPLOT statements each reference the dynamics on their X= and Y= arguments. At runtime for both plots, the variable that has been assigned to XVAR will provide X values for the plot, and the variable that has been assigned to YVAR will provide Y values.
- The MVAR statement declares the macro variable STUDY. Because STUDY is not a SAS automatic macro variable, it will be created for use in this template. On the statements that later execute this template, you must initialize a value for STUDY.

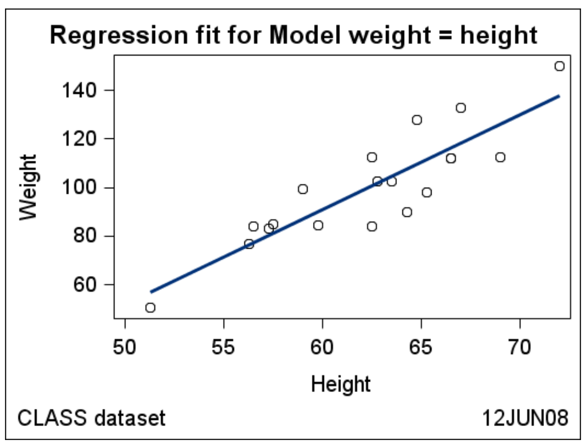
  The MVAR statement also declares the automatic macro variable SYSDATE. At runtime, the current system date and time will be substituted for this variable.
- The ENTRYFOOTNOTE statement references both of the macro variables STUDY and SYSDATE. The value that you assign to STUDY will be displayed as a left-justified

footnote, and the runtime value of SYSDATE will be displayed as a right-justified footnote.

As with all GTL templates, the MYGRAPHS.REGFIT template can be executed with either a PROC SGRENDER statement or a DATA step. Either way, any dynamics and/or new macro variables that are declared in the template must be initialized to provide runtime values for them. The following example executes the template with PROC SGRENDER:

```
%let study=CLASS dataset;
proc sgrender data=sashelp.class template=mygraphs.regfit;
  dynamic xvar="height" yvar="weight";
run;
```

- The %LET statement assigns string value "CLASS data set" to the STUDY macro variable.
- PROC SGRENDER uses the DYNAMIC statement to initialize the dynamic variables XVAR and YVAR. XVAR is assigned to the input variable HEIGHT, and YVAR is assigned to the input variable WEIGHT.



The DATA step uses the DYNAMIC= suboption of the ODS= option to initialize dynamics. Macro variables can be initialized from the existing symbol table. You can update the symbol table during DATA step execution with a CALL SYMPUT or CALL SYMPUTX routine. The following example executes the MYGRAPHS.REGFIT template with a DATA step:

```
data _null_;
  if _n_=1 then call symput("study","CLASS dataset") ;
  set sashelp.class;
  file print ods=( template="mygraphs.regfit"
                  dynamic=( xvar="height" yvar="weight" ) );
  put _ods_;
run;
```

- The CALL SYMPUT routine initializes the macro variable STUDY with the string value "CLASS dataset." The macro variable only needs to be initialized once, so the IF statement limits the initialization to the first observation (_N_ = 1).
- The DYNAMIC= suboption initializes the dynamic variables XVAR and YVAR. XVAR is assigned to the input variable HEIGHT, and YVAR is assigned to the input variable WEIGHT.

For a more complete discussion of this topic and additional examples, see Chapter 14, “Using Dynamics and Macro Variables to Make Flexible Templates,” on page 251.

Managing the Output Data Object

Setting Labels and Formats for the Output Columns

By default, the columns in the output data object derive variable attributes (name, type, label, and format) from the input variables. However, using the LABEL and FORMAT statements, you can change the label and format of the corresponding output object column.

The LABEL and FORMAT statements are available on PROC SGRENDER and on the DATA step. The following example assigns labels to the HEIGHT and WEIGHT variables that are used in the MYGRAPHS.SCATTER template. It also assigns a format to the WEIGHT variable.

```
proc sgrender data=sashelp.class template=mygraphs.scatter;
  label height="Height in Inches" weight="Weight in Pounds";
  format weight 3.;
run;
```

Setting a Name and Label for the Output Data Object

When the output data object is created, it is assigned a name and a label. The following table shows the default names and labels, depending on whether the corresponding GTL template is executed with a PROC SGRENDER statement or a DATA step:

Option	Default Name with PROC SGRENDER	Default Name with DATA Step
OBJECT= *name*	SGRENDER	FilePrint`n` (each execution of the DATA step increments the object name : FilePrint1, FilePrint2, and so on)
OBJECTLABEL="*string*"	The SGRENDER Procedure	same as object name

Using either PROC SGRENDER or a DATA step, you can use the OBJECT= option to set a name for the output data object. You can use the OBJECTLABEL= option to set a descriptive label for the data object. The following example sets the object name and label on PROC SGRENDER:

```
/* set object name and label on PROC SGRENDER */
proc sgrender data=sashelp.class template=mygraphs.scatter
              object=Scatter1
              objectlabel="Scatter Plot 1" ;
run;
```

This next example sets the object name and label on a DATA step:

```
/* set object name and label on a DATA step */
data _null_;
  set sashelp.class;
  file print ods=( template="mygraphs.scatter"
                   object=Scatter2
                   objectlabel="Scatter Plot 2" );
```

```
   put _ods_;
run;
```

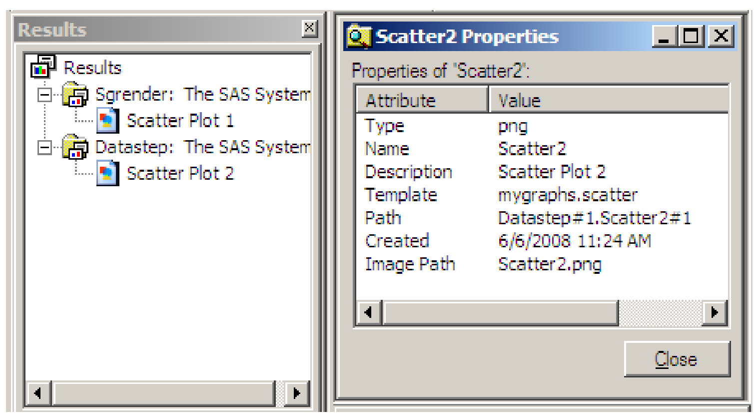
Viewing the Data Object Name and Label in the Results Window

When a GTL template is executed, an ODS data object is populated and bound to the template. The data object is assigned a name, and that name can be used to reference the object on various ODS statements, such as ODS SELECT, ODS EXCLUDE, and ODS OUTPUT. The data object is also assigned a label.

Object names and labels appear in the Results window. To view them,

1. Open the Results window if it is not already open (choose **View** ⇨ **Results**).
2. Right-click on the graph and choose **Properties** to view the object properties.

   The following figure shows the output objects that were created in "Setting a Name and Label for the Output Data Object" on page 316. The Results window shows the two objects that were created, and the Scatter2 Properties window shows the properties for the second object, which was named *Scatter2*.



Setting a Name for the Output Image File

By default, the output image file is assigned the same name as the output data object. You can use the IMAGENAME= option in the ODS GRAPHICS statement to assign an alternative name to the output image file. For example, the following code assigns the filename *regfit_heightweight* to the output image file:

```
ods graphics / imagename="regfit_heightweight";

proc sgrender data=sashelp.class template=mygraphs.regfit;
   dynamic xvar="height" yvar="weight";
run;
```

Converting the Output Data Object to a SAS Data Set

A data object can be converted to a SAS data set with the ODS OUTPUT statement. Generally, you identify the data object to convert, and assign it a data set name.

When a GTL template is executed with PROC SGRENDER, the output data object is always named SGRENDER. Thus, you can identify the data object by that name in the

ODS OUTPUT statement. The following example converts the data object to a SAS data set named REGFIT1:

```
ods output sgrender=regfit1;

proc sgrender data=sashelp.class template=mygraphs.regfit;
   dynamic xvar="height" yvar="weight";
run;
```

Because the output object name from the DATA step changes with each execution, it is handy to use the OBJECT= option on the DATA step FILE statement to set the object name so that it is easy to identify for the conversion.

The following example assigns the name DATAOBJ to the data object and uses that name to convert the data object to a SAS data set named REGFIT2:

```
ods output dataobj=regfit2 ;

data _null_;
  if _n_=1 then call symput("study","CLASS dataset");
  set sashelp.class;
  file print ods=( template="mygraphs.regfit"
                   dynamic=( xvar="height" yvar="weight" )
                   object=dataobj );
  put _ods_;
run;
```

Chapter 19
Managing Graphical Output

Introduction

Whenever you run a program that creates ODS Graphics output, several details are handled by default. Among them are the following:

- output file characteristics (file path and filename)
- image characteristics (format, name, DPI, size)
- ODS style used
- when anti-aliasing is used
- whether fonts and markers are scaled when graph size is changed
- whether the graph that is created can be edited
- whether data tips are produced.

In addition to the actual template code, you have a great deal of control over the environment in which ODS graphs are produced. Knowing what options are available and how to adjust these options gives you the maximum control in producing the best possible graphs for your needs.

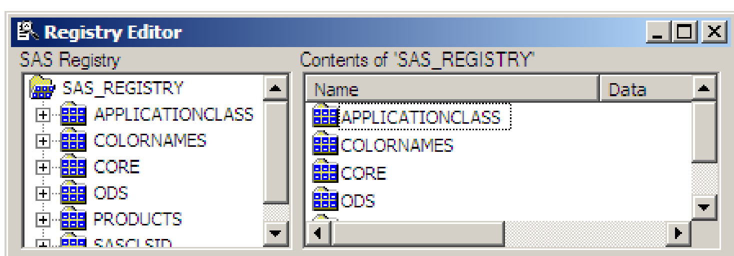
Three areas work in conjunction with each other to control all aspects of graph creation:

SAS Registry	Provides a repository of defaults for many options that affect ODS Graphics
ODS Destination statement	Provides options specific to destinations, such as HTML, PDF, and RTF
ODS GRAPHICS statement	Provides many global options that affect ODS graphics

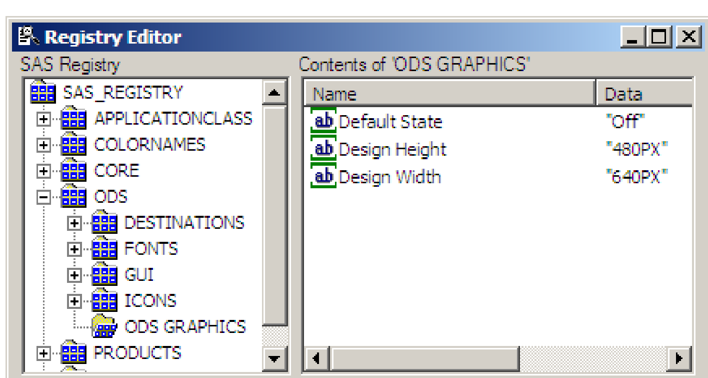
You often need to add options to both the ODS destination statement and the ODS GRAPHICS statement to get the desired output. Resetting SAS registry keys serves to configure your default ODS Graphics environment.

SAS Registry Settings for ODS Graphics

The SAS Registry is a special SAS itemstore file that is stored in your SASUSER storage location. It contains the default settings for many SAS products and their features. You can browse or edit this hierarchical file with PROC REGISTRY, or with the Registry Editor window. The window can be accessed with the global REGEDIT command from a Display Manager session. When you issue the command, the main Registry Editor window opens:



If you expand the ODS folder, you will see a sub-folder for ODS GRAPHICS, which contains three registry keys.



Registry Key	**Use**
Default State	Determines whether the ODS Graphics environment is active by default

Registry Key	Use
Design Height	Determines the default height of a graph that is generated with GTL
Design Width	Determines the default width of a graph that is generated with GTL

If you were to change the Default State from Off to On, it would make the ODS Environment active in every SAS session. This implies that if you run a procedure that normally requires you to activate the ODS Graphics environment with the ODS GRAPHICS ON; statement, you would not have to issue this statement—ODS graphs would be automatically produced every time you run an ODS graphics-enabled procedure such as UNIVARIATE, ARIMA, or REG.

Note: The SAS/GRAPH procedures such as SGRENDER, SGPLOT, SGPANEL, and SGSCATTER only produce template-based graphics. They internally activate the ODS Graphics environment if it is not active and are unaffected by the Default State key value.

The Design Height and Design Width keys control the default graph size for all graph templates. The 640px by 480px size represents a 4/3 aspect ratio. If you change these values, any new or existing graph templates are affected unless you explicitly set a DESIGNWIDTH= or DESIGNHEIGHT= option in the BEGINGRAPH statement in the graph template definition. For details, see “Controlling Graph Size” on page 326.

ODS Destination Statement Options Affecting ODS Graphics

Each ODS destination has options that govern aspects of your ODS Graphics output. The following table shows the options for the most commonly used destinations.

Table 19.1 *ODS Destination Options that Affect ODS Graphics*

ODS Destination	Options for ODS Graphics	Description
LISTING		Creates a stand-alone image. The default image format is PNG.
	GPATH="*directory-spec*"	Indicates the directory where images are created. The default is the current working directory.
	IMAGE_DPI=*number*	Specifies the image resolution in dots per inch for output images. IMAGE_DPI=100 is the default.
	STYLE= *style-definition*	Specifies the style to use. STYLE=LISTING is the default.
	SGE= OFF | ON	SGE=ON enables creation of corresponding SGE file(s), which can be edited with the ODS Graphics Editor. The SGE file(s) have the same name as the image file(s). SGE=OFF is the default.
PDF		Creates embedded image(s) in a PDF document. The default image format is PNG.
	DPI=*number*	Specifies the image resolution in dots per inch for output images. DPI=200 is the default.
	STYLE= *style-definition*	Specifies the style to use. STYLE=PRINTER is the default.
RTF		Creates embedded image(s) in RTF document. The default image format is PNG.
	IMAGE_DPI=*number*	Specifies the image resolution in dots per inch for output images. IMAGE_DPI=100 is the default.
	STYLE= *style-definition*	Specifies the style to use. STYLE=RTF is the default.

ODS Destination	Options for ODS Graphics	Description
HTML		Creates stand-alone image(s) and the HTML page. Image(s) are referenced in the HTML page. The default image format is PNG.
	GPATH="*directory-spec*"	Indicates the directory where images are created. If not specified, the PATH= "*directory-spec*" is used.
	IMAGE_DPI=*number*	Specifies the image resolution in dots per inch for output images. IMAGE_DPI=100 is the default.
	STYLE= *style-definition*	Specifies the style to use. STYLE=DEFAULT is the default.

ODS GRAPHICS Statement Options

The ODS GRAPHICS statement is the primary statement that controls the runtime environment for producing template-based graphs. In a sense, is it similar to the GOPTIONS statement for GRSEG-based graphs, but completely independent of that statement. The GOPTIONS statement does not affect template-based graphical output and the ODS GRAPHICS does not affect GRSEG-based graphs.

All options for the ODS GRAPHICS statement are global to a SAS session, unless

- the graphics environment is disabled with the ODS GRAPHICS OFF; statement.
- the RESET or RESET= option of the ODS GRAPHICS statement is used to return the default state to all options or a specific option.

The following table shows some of the available options. For a complete and more detailed explanation of all available options, see the documentation of the ODS GRAPHICS statement in *SAS Output Delivery System: User's Guide*.

Table 19.2 *Partial Listing of ODS GRAPHICS Statement Options*

Task	Option
Specify the threshold for allowing anti-aliasing.	ANTIALIASMAX= *positive-integer* The default is 600 markers and/or lines.
Specify whether graph rendering uses anti-aliasing.	ANTIALIAS= ON | OFF The default is ON.
Specify whether to draw a border around any graph.	BORDER= ON | OFF The default is ON.

Task	Option
Specify the height of any graph.	HEIGHT= *dimension* Supported dimension units include SPX (special pixels), PX (pixels), IN (inches), CM (centimeters), and MM (millimeters). This option overrides the design height specified by the template definition. The default unit is SPX. It is recommended you always provide a unit such as PX, IN, CM, or MM with the dimension value.
Specify the image format used to generate image files.	IMAGEFMT= STATIC | *image-format* Supported formats include PNG, GIF, JPEG, WMF, TIFF, PDF, PS, and others. The keyword STATIC is the default, which means to automatically select the best format, based on the output destination.
Specify whether data tips are generated.	IMAGEMAP= OFF | ON The default is OFF.
Specify the base image filename.	IMAGENAME= "*file-name*" (no path information) The default is to use the invoking procedure name as the base name.
Control whether legend(s) are drawn.	MAXLEGENDAREA=**n** Specifies an integer that is interpreted as the maximum percentage that a legend can occupy in the overall graphics area. The default integer is 20.
Reset one or more ODS GRAPHICS options to its default. RESET by itself is the same as RESET=ALL.	RESET | RESET= *option* The *option* can be ALL, HEIGHT, WIDTH, INDEX, and other *options*. By default, each time you run a procedure, new images are created and numbered incrementally using a base name, such as SGRender, SGRender1, SGRender2, and so on. RESET will reset to the base name without the increment number. This is handy if you run a PROC several times and are interested only in the images from the last run (the previous ones will be overwritten). This option is positional, so it typically comes first.
Specify whether the content of any graph is scaled proportionally.	SCALE = ON | OFF The default is ON.
Specify the maximum number of distinct mouse-over areas allowed before data tips are disabled.	TIPMAX=*n* The default number is 500.

Task	Option
Specify the width of any graph.	WIDTH= *dimension* Supported dimension units include SPX (special pixels), PX (pixels), IN (inches), CM (centimeters), and MM (millimeters). This option overrides the design width specified by the template definition. The default unit is SPX. It is recommended you always provide a unit such as PX, IN, CM, or MM with the dimension value.

Common Tasks

The following sections show the coding that is necessary to accomplish several common tasks for managing ODS graphics output.

Controlling the Image Name and Image Format

To control the image name and image format for ODS Graphics output, use the IMAGENAME= and IMAGEFMT= options in the ODS GRAPHICS statement.

The following example creates a GIF image named REGPLOT:

```
ods graphics / imagename="regplot" imagefmt=gif;
```

Note: Unless you have a special requirement for changing the image format, it not recommended that you do so. The default PNG (Portable Network Graphics) format is far superior to other formats, such as GIF, in support for transparency and a large number of colors. PNG images require much less disk storage space than JPEG or TIFF formats.

The assigned name REGPLOT is treated as a "root" name and the first output created is named REGPLOT. Subsequent graphs are named REGPLOT1, REGPLOT2, and so on, with an increasing index counter. All graphs in this example will be GIF images.

If you are developing a template and its takes several submissions to get the desired output, you might want to use RESET or RESET= *option* to force each output to replace itself:

```
ods graphics / reset=index ... ;
```

This specification causes all subsequent images to be created with the default or current image name.

The options in the ODS GRAPHICS statement are global to the SAS session. These options are in effect until you

- disable the ODS graphics environment by submitting the statement **`ods graphics off;`**
- reset all options to their defaults submitting the statement **`ods graphics / reset;`**
- reset specific options submitting the statement **`ods graphics / reset=`*`option`*`;`**

Controlling the Image's Output Location

To control the image location (path) for ODS Graphics output, use the PATH= or GPATH= option on the ODS destination statement.

```
ods listing gpath="C:\ODSgraphs";

ods html gpath="C:\ODSgraphs";
```

For the HTML destination, the PATH= option is used to indicate whether the HTML page is stored. If GPATH= is not used, images are stored at the PATH= storage location. Use PATH= and GPATH= together when you want to store images in a different storage location. The (URL= NONE | **`url-spec`**) sub-option specifies a Uniform Resource Locator for the PATH= or the GPATH= options.

For example, the following program will create an HTML page named **`u:\public_html\report.html`**:

```
ods graphics / reset imagename="graph";
ods listing close;
ods html style=statistical
         path="u:\public_html"
         gpath="u:\public_html" (url=none)
         file="report.html";

proc sgrender data=sashelp.heart template=modelfit
              des="Regression Fit plot";
run;
proc sgrender data=sashelp.heart template=distribution
              des="Distribution of Cholesterol";
   dynamic var="Cholesterol";
run;
ods html close;
ods listing;
```

The graphs produced are named graph.png and graph1.png and are stored in u:\public_html\. The (URL=NONE) suboption prevents any path or URL information from being included in the SRC=" " attribute of the <IMG> tag. This creates relative references to the images in the html source:

<img alt="Regression Fit" src="graph.png" style=" height: 480px; width: 640px;" border="0">

<img alt="Distribution of Cholesterol" src="graph1.png" style=" height: 480px; width: 640px;" border="0">

For ODS destinations such as RTF or PDF, the image is embedded in the document that is created by that destination.

Controlling Graph Size

Overview

By default, the size of the graph that you create with ODS Graphics is governed by the following:

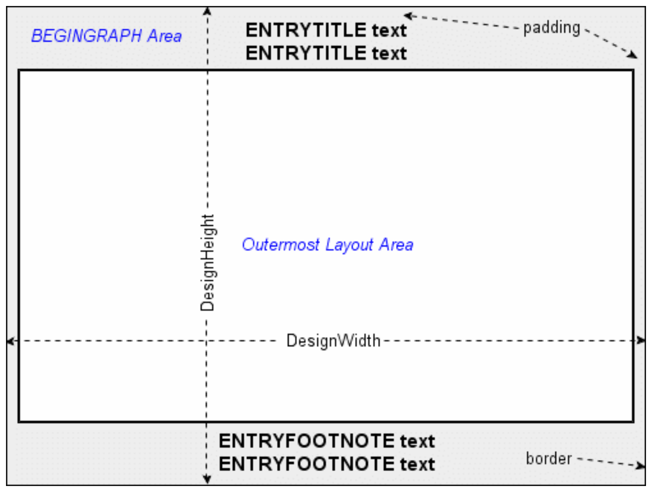
- settings for ODS Graphics in the SAS Registry
- the size indicated by the DESIGNWIDTH= and DESIGNHEIGHT= options of the BEGINGRAPH statement
- the WIDTH= and HEIGHT= options of the ODS GRAPHICS statement.

BEGINGRAPH Statement

When creating a graphics template, you often want to control the design width and design height, especially for multi-cell graphs.

```
BEGINGRAPH / DESIGNWIDTH= dimension   DESIGNHEIGHT= dimension;
```

In addition to specifying sizes in several units, you can also refer to the current registry settings with the constants DEFAULTDESIGNWIDTH and DEFAULTDESIGNHEIGHT.



In the following example, the intent is to produce a square graph (equal height and width) in order to reduce unused graphical area. The design width is set to the default internal height (DEFAULTDESIGNHEIGHT).

```
proc template;
  define statgraph squareplot;
    dynamic title xvar yvar;
    begingraph / designwidth=defaultDesignHeight;
      entrytitle title;
      layout overlayequated / equatetype=square;
         scatterplot x=xvar y=yvar;
         regressionplot x=xvar y=yvar;
      endlayout;
    endgraph;
  end;
run;
```

If this template were executed with the following SGRENDER specification, a 480px by 480px graph would be created:

```
proc sgrender data=mydata template="squareplot";
  dynamic title="Square Plot" xvar="time1" yvar="time2";
run;
```

If a 550px width or height were set on an ODS GRAPHICS statement before the template is executed with the SGRENDER procedure, a 550px by 550px graph would be created, maintaining the 1:1 aspect ratio:

```
ods graphics / width=550px;
proc sgrender data=mydata template="squareplot";
  dynamic title="Square Plot" xvar="time1" yvar="time2";
run;
```

```
/* Setting a 550px height would create the same size graph */
ods graphics / height=550px;
proc sgrender data=mydata template="squareplot";
  dynamic title="Square Plot" xvar="time1" yvar="time2";
run;
```

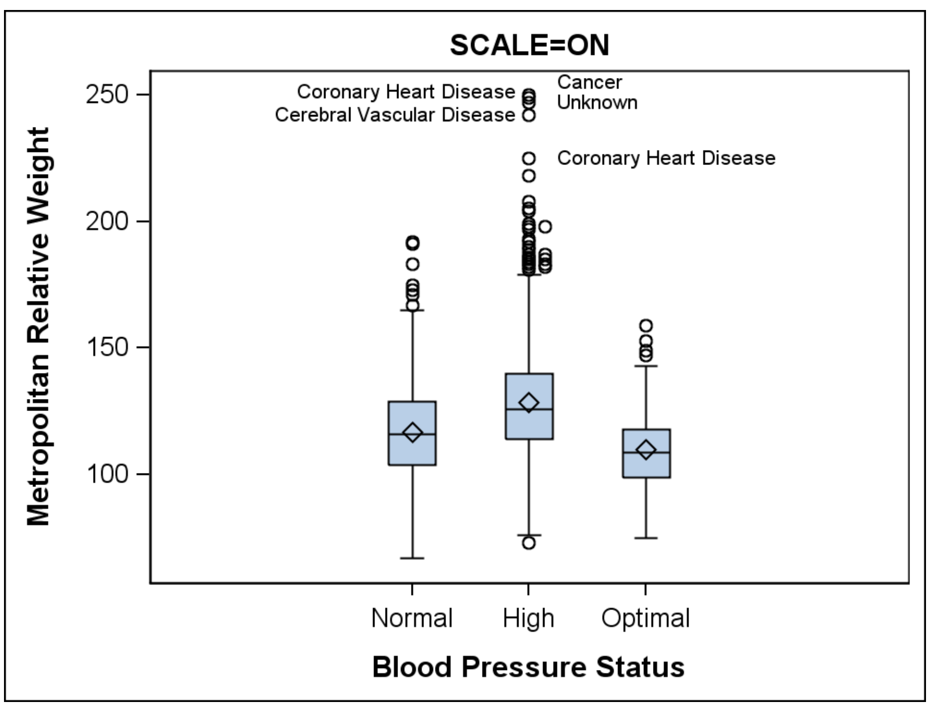
When no DESIGNWIDTH= or DESIGNHEIGHT= option is specified in the BEGINGRAPH statement, graphs are rendered with the registry defaults, unless changed by the ODS GRAPHICS statement HEIGHT= or WIDTH= options.

Examples for sizing multi-cell graphs are discussed in Chapter 9, "Using a Simple Multi-cell Layout," on page 143, Chapter 10, "Using an Advanced Multi-cell Layout," on page 155, and Chapter 11, "Using Classification Panels," on page 185.

Understanding Graph Scaling

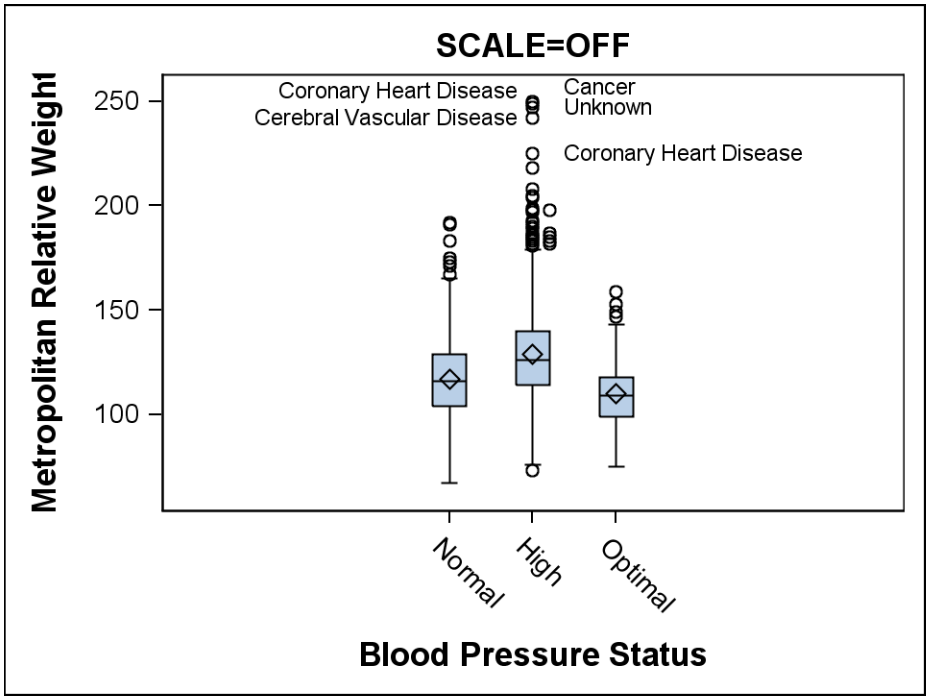
ODS graphics uses style information to control the appearance of the graph. Style definitions contain information about fonts, color, lines, and markers, and they also contain settings such as font size and marker size. When a graph is rendered at a size larger or smaller than its design size, scaling takes place by default.

```
ods graphics / width=480px height=360px scale=on ;
```



If you turn off scaling, the font sizes, marker sizes, and so on revert to the sizes that are defined in the style. To accommodate the larger font sizes for the titles, footnotes, axis labels, tick values, and data labels, the wall area and contained graphical components automatically shrink.

```
ods graphics / width=480px height=360px scale=off ;
```



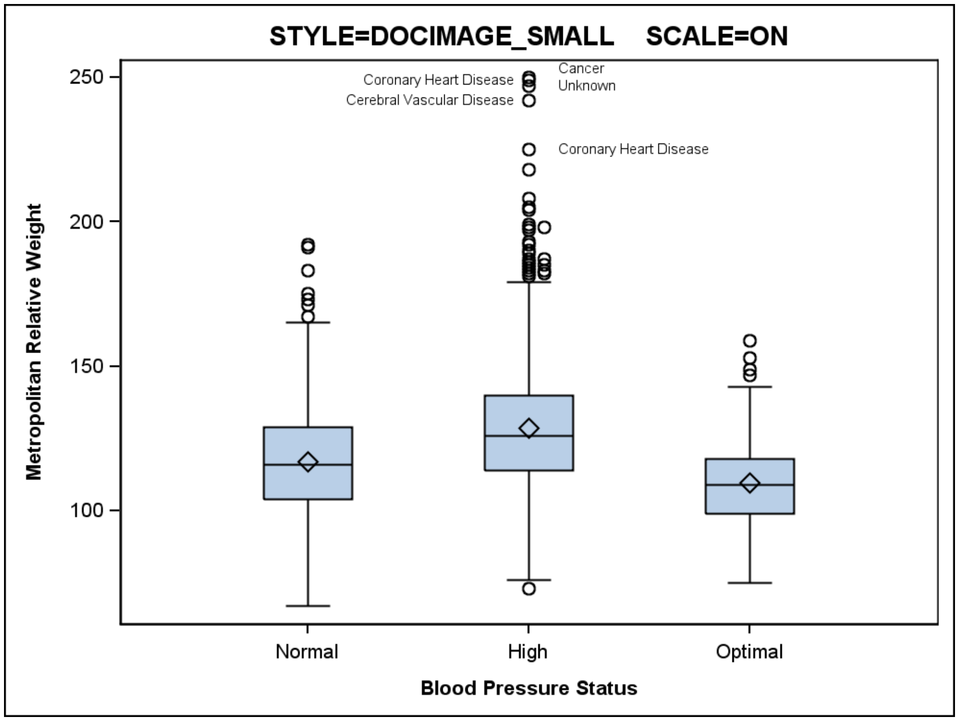
In general, having the fonts scale up or down as the graph size increases or decreases is desirable. However, in some cases you might want greater control of the font sizes.

The examples in this document were created with different styles that varied only in the font sizes that they used. In some cases, smaller graphs look better when rendered in a smaller set of fonts. The style examples below use the LISTING style as a parent, but you could use any style as the parent. The DOCIMAGE style keeps fonts close to the default sizes and weights, while the DOCIMAGE_SMALL style reduces the font sizes by a few points. See Chapter 17, “Managing the Graph Appearance with Styles,” on page 299 for a discussion of defining you own styles and what parts of the graph are affected by various style elements.

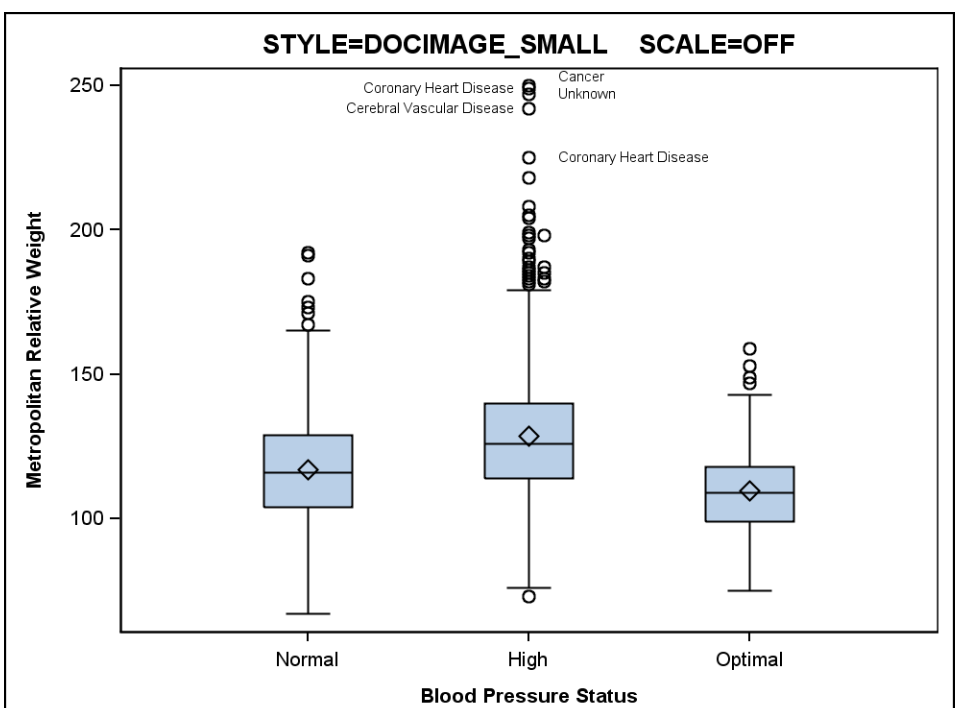
```
proc template;
  define style docimage;
    parent=styles.listing;
    style GraphFonts from GraphFonts
         "Fonts used in graph styles" /
         'GraphDataFont' = ("<sans-serif>, <MTsans-serif>",8pt)
         'GraphUnicodeFont' = ("<MTsans-serif-unicode>",10pt)
         'GraphValueFont' = ("<sans-serif>, <MTsans-serif>",10pt)
         'GraphLabelFont' = ("<sans-serif>, <MTsans-serif>",12pt,bold)
         'GraphFootnoteFont' = ("<sans-serif>, <MTsans-serif>",10pt)
         'GraphTitleFont' = ("<sans-serif>, <MTsans-serif>",12pt,bold);
  end;

  define style docimage_small;
    parent=styles.listing;
    style GraphFonts from GraphFonts
         "Fonts used in graph styles" /
         'GraphDataFont' = ("<sans-serif>, <MTsans-serif>",6pt)
         'GraphUnicodeFont' = ("<MTsans-serif-unicode>",8pt)
         'GraphValueFont' = ("<sans-serif>, <MTsans-serif>",8pt)
         'GraphLabelFont' = ("<sans-serif>, <MTsans-serif>",8pt,bold)
         'GraphFootnoteFont' = ("<sans-serif>, <MTsans-serif>",8pt)
```

```
            'GraphTitleFont' = ("<sans-serif>, <MTsans-serif>",10pt,bold);
      end;
   run;
```

The previous two graphs were created the DOCIMAGE style. These next two graphs were created with the DOCIMAGE_SMALL style.



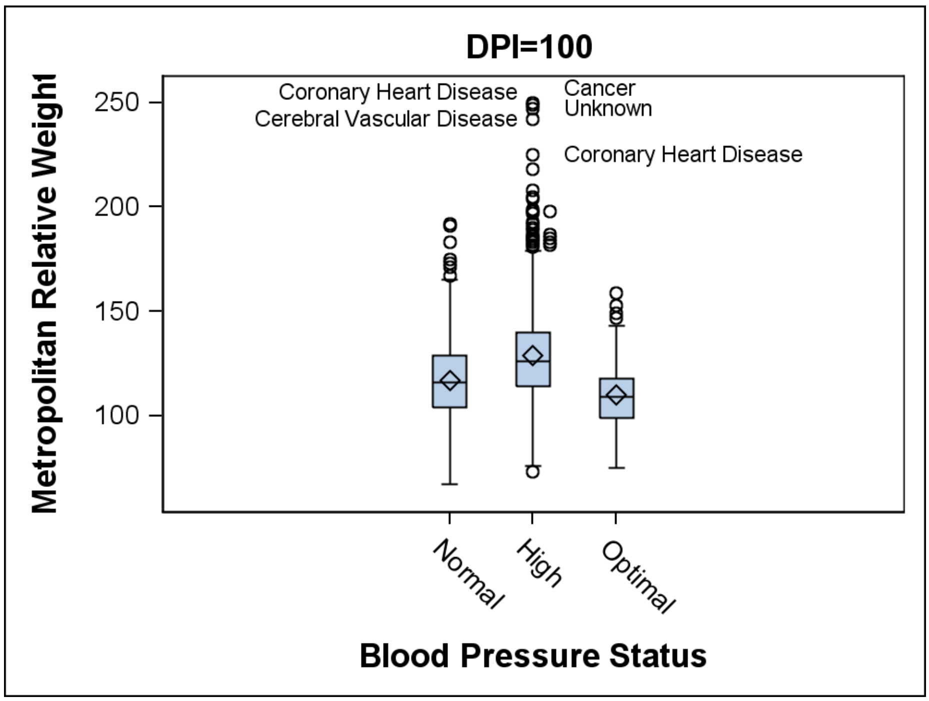
In both of these graphs that use the DOCIMAGE_SMALL style, the text in the graph is still legible whether scaling is on or off. Also, more space is available to the graphical elements in the output.



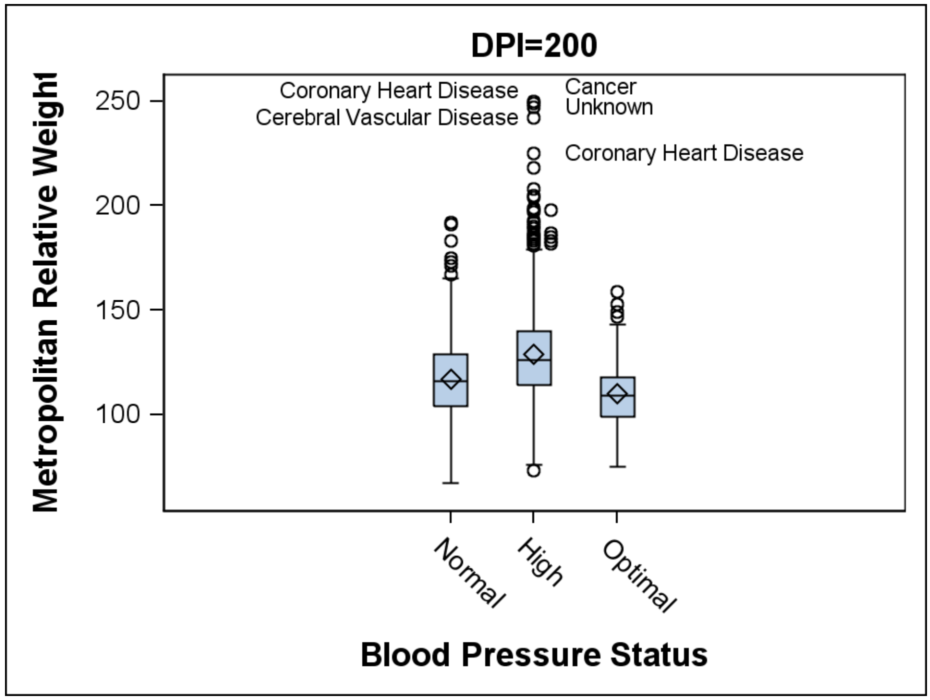
Controlling DPI

All ODS destinations use a default DPI (dots per inch) setting when creating ODS Graphics output. By default, LISTING and HTML use 100 dpi, while RTF and PDF use 200 dpi. Graphs that are rendered at higher DPI have greater resolution and larger file size. Although DPI can be set to large values such as 1200, from a practical standpoint, settings larger than 300dpi are seldom necessary for most applications. Also, setting an unrealistically large DPI like 1200 could cause an out-of-memory condition. Note that the ODS option for setting DPI is not the same for all destinations. For the LISTING and HTML destinations, use the IMAGE_DPI= option. For the RTF and PDF destinations, use the DPI= option.

```
ods graphics / width=480px height=360px scale=off;
ods listing image_dpi=100 style=docimage_small;
```




```
ods graphics / width=480px height=360px scale=off;
ods listing image_dpi=200 style=docimage_small;
```



In these examples, the text in the 200 dpi graph is slightly more legible. Markers and lines are also more legible.

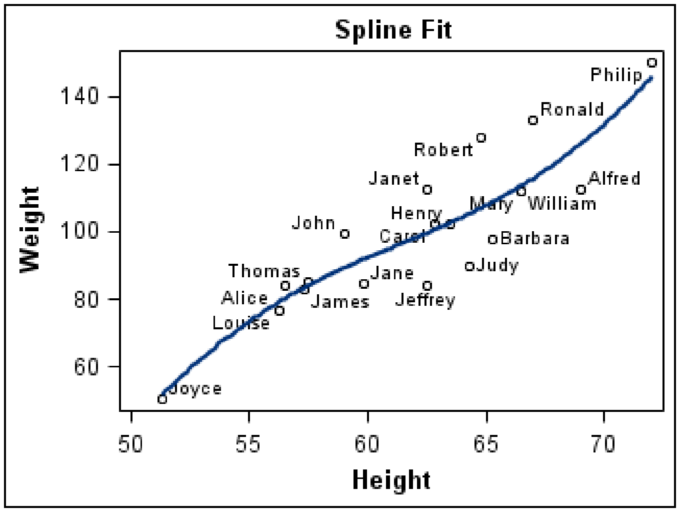
Controlling Anti-Aliasing

Anti-aliasing is a graphical rendering technique that improves the readability of text and the crispness of the graphical primitives, such as the markers and lines. By default, ODS Graphics uses anti-aliasing.

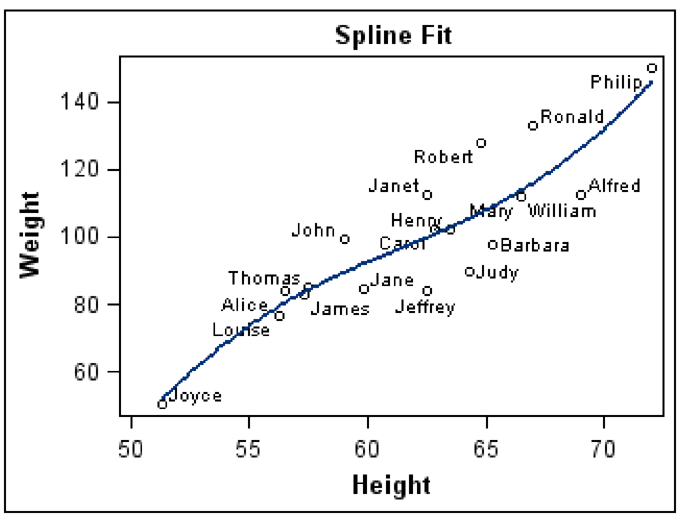
Note: Titles, footnotes, entry text, axis labels, tick values, and legend text is always anti-aliased. Graphical components related to the data, such as markers, lines, and data labels, are affected by the ANTIALIAS= and ANTIALIASMAX= options, as discussed in this section.

To see how much the graph quality is improved with anti-aliasing, you can turn this feature on and off with the ANTITALIAS= option in the ODS GRAPHICS statement.

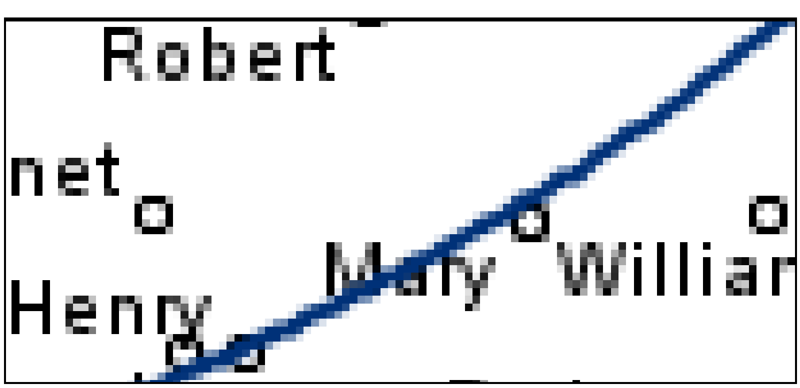
```
ods graphics / antialias=on;
ods listing image_dpi=100;
proc sgrender data=sashelp.class template=fitline;
run;
```



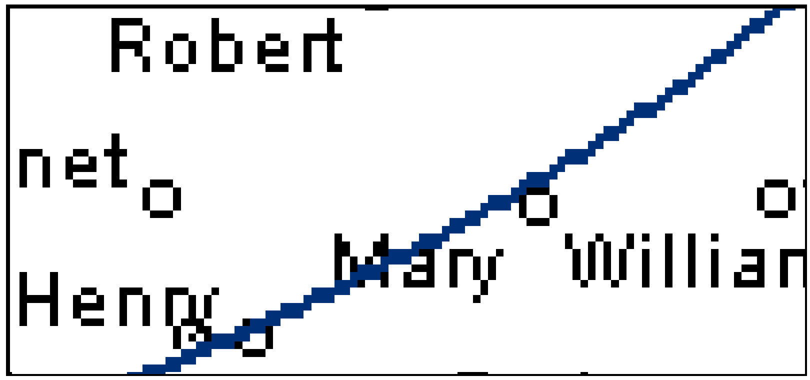

```
ods graphics / antialias=off ;
ods listing image_dpi=100;
proc sgrender data=sashelp.class template=fitline;
run;
```



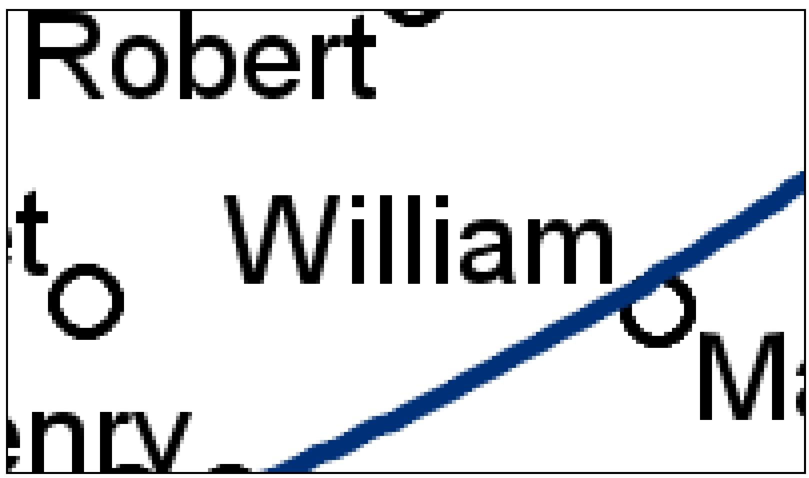
The following image shows a zoomed-in view of a portion of the anti-aliased image (100dpi). Notice that the text, markers, and line appear fuzzy because of the anti-aliasing algorithm.



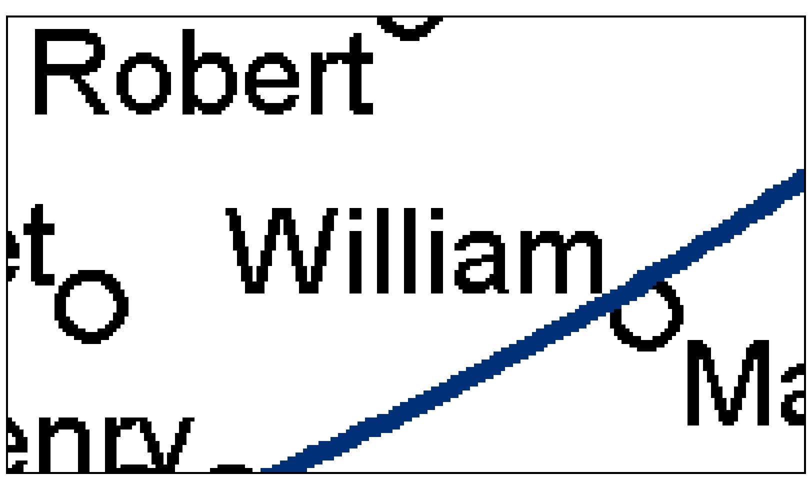
This next image shows a zoomed-in view of the image (100dpi) that has anti-aliasing turned off. Notice that the text, markers, and line are not fuzzy but have a jagged appearance.



If the image is created at 300dpi, the combination of anti-aliasing and higher resolution produces a very high quality image.



The non-anti-aliased image at 300dpi, is good but still has jagged edges.



To perform anti-aliasing requires additional computer resources (CPU, memory, and execution time). Graphs that have a lot of markers, lines, and text use even more resources. Filled or gradient 3D surface plots might require even more resources.

Setting a higher DPI increases anti-aliasing resources. At some point, ODS Graphics deems that anti-aliasing requires too many resources and it turns the feature off. When this happens, you will get a non-anti-aliased rendered graph and a message in the SAS log similar to the following :

```
NOTE: Marker and line antialiasing has been disabled because the threshold has
      been reached. You can set ANTIALIASMAX=5700 in the ODS GRAPHICS
      statement to restore antialiasing.
```

If you want anti-aliasing for a graph that caused the anti-aliasing to be disabled, you must set a higher threshold (at least 5700) for anti-aliasing with the ANTIALIASMAX= option of the ODS GRAPHICS statement.

```
ods graphics / antialiasmax=5700;
```

The number that is specified on the ANTIALIASMAX= option represents the maximum number of observations in the data to be anti-aliased before anti-aliasing is disabled.

Creating a Graph that Can Be Edited

SAS provides an application called the ODS Graphics Editor that can be used to post-process ODS Graphics output. With the editor, you can edit the following features in a graph that was created using ODS Graphics:

- Change, add, or remove titles and footnotes.
- Change style, marker symbols, line patterns, axis labels, and so on.
- Highlight or explain graph content by adding annotation, such as text, lines, arrows, and circles.

For example, suppose the following template is used to create box plots in a graph and you want to indicate that the labeled outliers are far outliers (more than 3 IQR above 75th percentile).

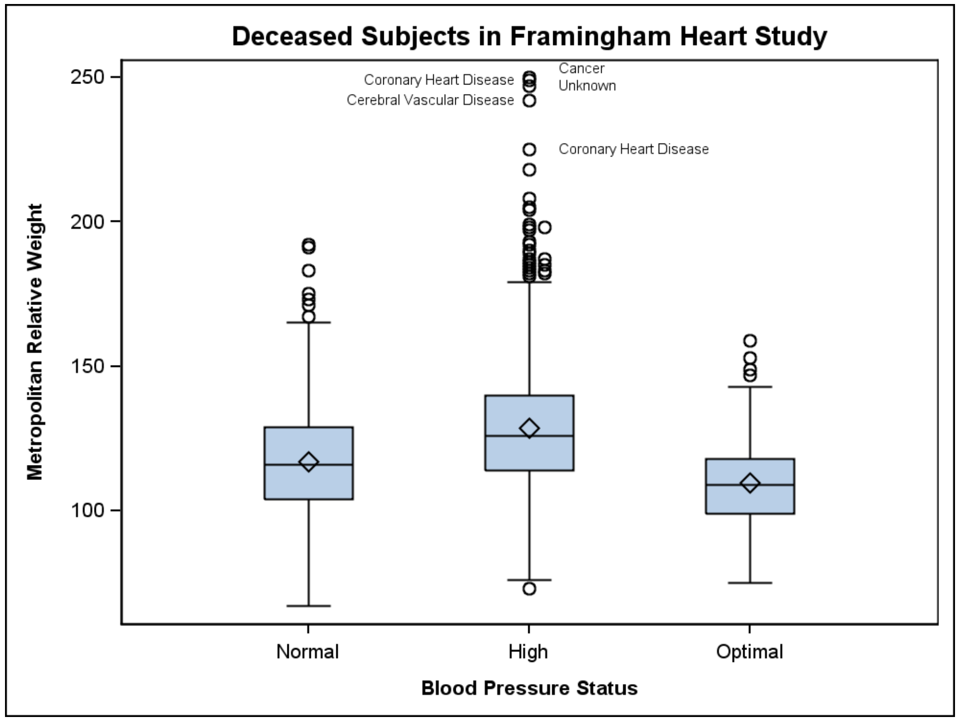
```
proc template;
  define statgraph boxplot;
    begingraph;
      entrytitle "Deceased Subjects in Framingham Heart Study";
      layout overlay;
        boxplot y=mrw x=bp_status / datalabel=deathcause
                spread=true labelfar=true;
      endlayout;
    endgraph;
  end;
run;
```

To create ODS Graphics output that can be edited, you must specify the SGE=ON option in the ODS LISTING destination statement before creating the graph:

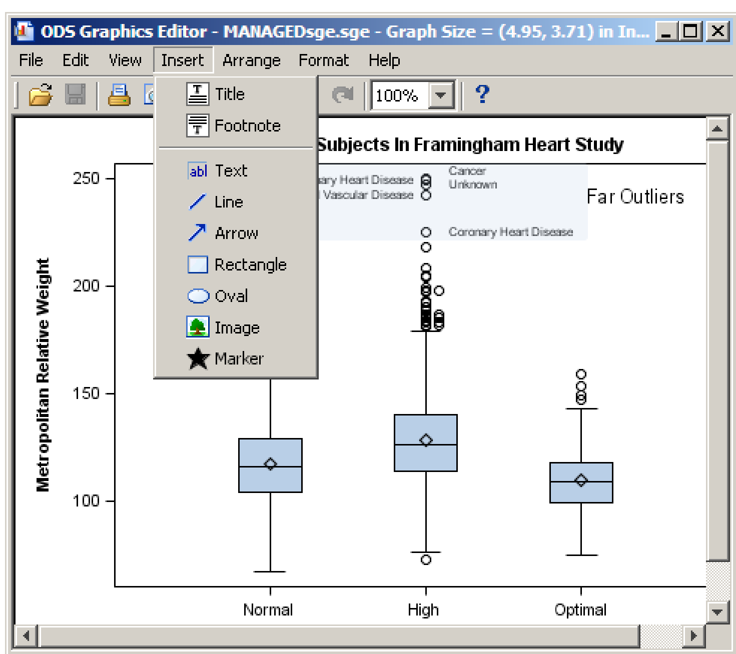
```
ods listing sge=on;

proc sgrender data=sashelp.heart template=boxplot;
  where status="Dead";
run;
```

When SGE=ON is in effect, an .SGE file is created in addition to the image file normally produced. From the Results Window, you can open the .SGE file in the ODS Graphics Editor by selecting Open in the icon. You can also open the .SGE file directly from the

Windows file system. The .SGE file is always created in the same location as the image output. Here is the image output.



The following figure shows the Graphical User Interface for the ODS Graphics Editor after some of the annotation has been completed.



You can save your annotated graph as an .SGE file or as an image file. If you save it as an .SGE file, you can open it again for further editing.

Note: Changes that are made in the ODS Graphics Editor do not affect the compiled template code.

After you are finished creating editable graphics, you should either close the ODS destination (in this case LISTING) or specify SGE=OFF to discontinue producing .SGE files and avoid the extra computational resources used to generate the extra .SGE files:

```
ods listing sge=off;
```

Creating a Graph to Include in MS Office Applications

The default height for a graph is 480 pixels. At a 100 dot per inch (DPI) setting, you can consider the default height to be 4.8 inches. If you render a graph at 480 pixels and 100 DPI, insert it into a document like an MS Office application, and then print the page, the graph height on paper will be 4.8 inches and all font sizes will look right in their point weights. You can render the graph at a higher DPI to get higher quality graphs. As long as the graph is then inserted in the document as a 4.8 inch graph, it will work as expected.

To alter the graph size or DPI for a graph that you want to include in an MS Office application, one technique that produces good results is to create a stand-alone image that is sized appropriately and has high resolution, say 200 DPI or 300 DPI.

```
ods graphics / reset width=5in imagename="fitplot" imagefmt=png
    antialias=on;
ods listing gpath="\ODSgraphs" image_dpi=200 style=analysis;

proc sgrender data= . . . template= . . .;
run;
```

This code produces a 5 inch, 200 DPI image **`\ODSgraphs\fitplot.png`**, which can be inserted into Word or PowerPoint documents. When only the WIDTH= or HEIGHT= option is specified in the ODS GRAPHICS statement, the design aspect ratio of the graph is maintained. Also, check the SAS log to ensure that anti-aliasing has not been disabled. If it has been disabled, add the ANTIALIASMAX= option (see "Controlling Anti-Aliasing" on page 332 for a discussion of anti-aliasing).

After inserting the graph into the MS Office document, you can change the picture size with good results (while maintaining aspect ratio). If you find that the text in the graph is too large or too small, recreate the graph with different font sizes using the techniques discussed in "Understanding Graph Scaling" on page 328.

To create good looking graphs for a two-column MS Word document where each column is about 3.5 inches wide, use a graph width of 3.5 inches. If the original graph has a default width of 640 pixels, you can set WIDTH=3.5IN in the ODS GRAPHICS statement to get a smaller graph with appropriately smaller fonts. In this case, the fonts will not be exactly the right point size, but they will be scaled smaller using a non-linear scaling factor.

Controlling Data Tips

Creating a Graph with Data Tips in an HTML Page

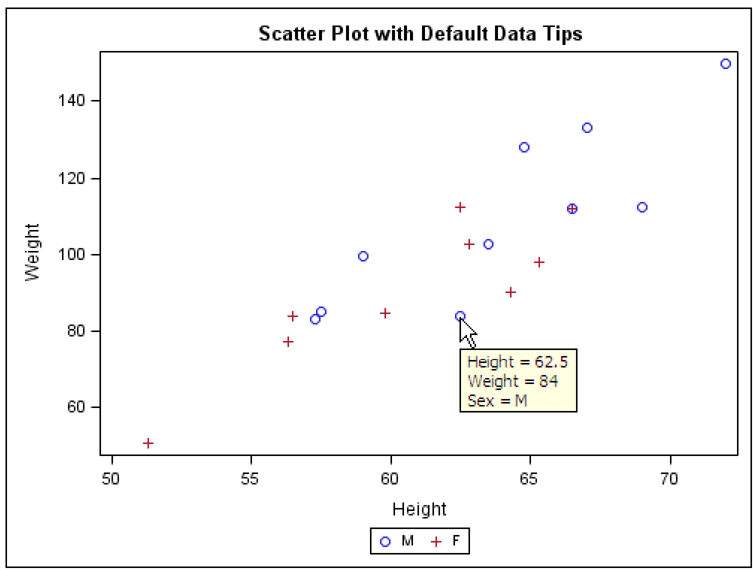
Data tips (sometimes called tooltips) can be displayed by graphs that are included in HTML pages. When data tips are provided, you can "mouse over" parts of a graph, and text balloons open to show information (typically data values) that is associated with the area where the mouse pointer rests. Nearly all plot statements in GTL create default data tip information. However, this information is not generated unless you request it with the IMAGEMAP= option in the ODS GRAPHICS statement:

```
ods html file=". . ." path=". . ." (url=none);
ods graphics / reset width=5in imagemap=on ;
```

```
proc sgrender data= . . . template= . . .;
run;

ods graphics / reset;
ods html close;
```

Using the following simple template, we can show how the default data tips look when the mouse pointer hovers over a data point:

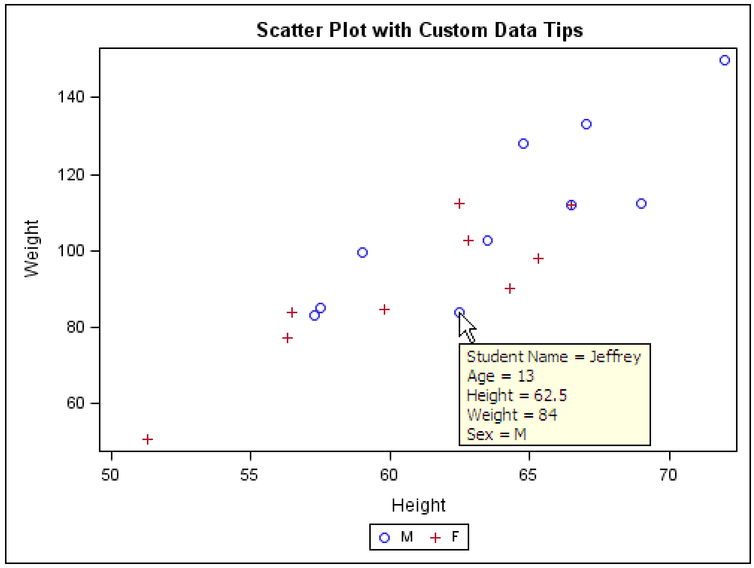
```
layout overlay;
  scatterplot x=height y=weight / group=sex name="s";
  discretelegend "s";
endlayout;
```



Creating a Graph with Custom Data Tips in an HTML Page

GTL supports plot statement syntax that enables you to suppress or customize the default data tip information. Here is an example:

```
layout overlay;
  /* scatter points have enhanced tooltips */
  scatterplot x=height y=weight / group=sex name="s"
    rolename=(tip1=name tip2=age)
    tip=(tip1 tip2 X Y GROUP)
    tiplabel=(tip1="Student Name")
    tipformat=(tip2=2.);
  discretelegend "s";
endlayout;
```



The ROLENAME=, TIP=, TIPLABEL= and TIPFORMAT= options are common to most plot statements in GTL.

ROLENAME defines one or more name / value pairs as *role-name* = *column-name*, where *column-name* is some input data column that does not participate directly in the plot. In this example, we want the NAME and AGE column values to show in the tip. Notice that the choice of role names is somewhat arbitrary. The TIP1 and TIP2 role names are added to the default role names X, Y, and GROUP.

The TIP= option defines a list of roles to be displayed, and it also determines their order their order in the display. Notice that it is not necessary to request all default roles. For example, it might be obvious from the legend that the GROUP role does not really need to be in the data tip, so in that case you would specify:

```
tip=(tip1 tip2 X Y)
```

For any role, the default tip label is 1) the data label, or 2) the name of the column that is associated with the role. If you want other label text displayed, use the TIPLABEL= option:

```
tiplabel=(tip1="Student Name" group="Group")
```

For any role, you can assign a format to the display of tip values.

Creating Shared Templates

When creating templates (especially with dynamics that generalize the usefulness of the template), you typically want to enable several people to create graphs from the template. To enable access to templates, you must store the "public" templates in a directory that is accessible to others. PROC TEMPLATE can store templates in specified SAS libraries and within specific item stores. By default, templates are stored in SASUSER.TEMPLAT , but another *library.itemstore* can be specified with the STORE= option in the DEFINE statement.

```
libname p "\\public\templates";

proc template;
  define statgraph graphs.distribution / store=p.templat ;
     ...
```

```
    end;
    define statgraph graphs.regression / store=p.templat ;
       ...
    end;
  run;
```

When this template code is submitted, you see the following notes in the SAS log:

```
NOTE: STATGRAPH 'Graphs.Distribution' has been saved to:
PUBLIC.TEMPLAT
NOTE: STATGRAPH 'Graphs.Regression' has been saved to:
PUBLIC.TEMPLAT
```

After shared templates are compiled and stored, others can access them to produce graphs.

```
libname p "\\public\templates" access=readonly;

ods path reset;
ods path (prepend) p.templat(read) ;

proc sgrender data= ... template=graphs.distribution;
  dynamic var="height";
run;
```

Manipulating the ODS search path is the best way to make the templates publicly available.

Note that this code did not replace the path but rather added an item store at the beginning of the path. This is done to allow access to all SAS-supplied production templates, which are stored in SASHELP.TMPLMST.

```
ods path show;
```

```
Current ODS PATH list is:
1. P.TEMPLAT(READ)
2. SASUSER.TEMPLAT(UPDATE)
3. SASHELP.TMPLMST(READ)
```

Appendix 1
SAS Keywords for Unicode Glyphs

Greek Letters

Keyword	Glyph	Unicode	Description
alpha	α	03B1	lowercase alpha
beta	β	03B2	lowercase beta
gamma	γ	03B3	lowercase gamma
delta	δ	03B4	lowercase delta
epsilon	ε	03B5	lowercase epsilon
zeta	ζ	03B6	lowercase zeta
eta	η	03B7	lowercase eta
theta	θ	03B8	lowercase theta
iota	ι	03B9	lowercase iota
kappa	κ	03BA	lowercase kappa
lambda	λ	03BB	lowercase lamda
mu	μ	03BC	lowercase mu
nu	ν	03BD	lowercase nu
xi	ξ	03BE	lowercase xi
omicron	ο	03BF	lowercase omicron

Keyword	Glyph	Unicode	Description
pi	π	03C0	lowercase pi
rho	ρ	03C1	lowercase rho
sigma	σ	03C3	lowercase sigma
tau	τ	03C4	lowercase tau
upsilon	υ	03C5	lowercase upsilon
phi	φ	03C6	lowercase phi
chi	χ	03C7	lowercase chi
psi	ψ	03C8	lowercase psi
omega	ω	03C9	lowercase omega
alpha_u	Α	0391	uppercase alpha
beta_u	Β	0392	uppercase beta
gamma_u	Γ	0393	uppercase gamma
delta_u	Δ	0394	uppercase delta
epsilon_u	Ε	0395	uppercase epsilon
zeta_u	Ζ	0396	uppercase zeta
eta_u	Η	0397	uppercase eta
theta_u	Θ	0398	uppercase theta
iota_u	Ι	0399	uppercase iota
kappa_u	Κ	039A	uppercase kappa
lambda_u	Λ	039B	uppercase lambda
mu_u	Μ	039C	uppercase mu
nu_u	Ν	039D	uppercase nu
xi_u	Ξ	039E	uppercase xi
omicron_u	Ο	039F	uppercase omicron
pi_u	Π	03A0	uppercase pi
rho_u	Ρ	03A1	uppercase rho

Keyword	Glyph	Unicode	Description
sigma_u	Σ	03A3	uppercase sigma
tau_u	Τ	03A4	uppercase theta
upsilon_u	Υ	03A5	uppercase upsilon
phi_u	Φ	03A6	uppercase phi
chi_u	Χ	03A7	uppercase chi
psi_u	Ψ	03A8	uppercase psi
omega_u	Ω	03A9	uppercase omega

Special Characters

Keyword	Glyph	Unicode	Description
prime	′	00B4	single prime sign
bar	‾	0305	combining overline *
bar2	═	033F	combining double overline *
tilde	˜	0303	combining tilde *
hat	ˆ	0302	

Appendix 2
SAS Graph Style Elements for GTL

Graphical Style Elements

Style Element	Recognized Attributes	Attribute Values (DEFAULT Style)
Graph Affects outer border appearance and background color.	BorderColor BorderWidth CellPadding CellSpacing	inherited inherited 0 inherited
GraphBackground Affects background of the graph.	Color Transparency	Colors('docbg') not set
GraphWalls Affects wall(s) bounded by axes.	Color Transparency FrameBorder LineThickness LineStyle ContrastColor	GraphColors('gwalls') not set on 1px 1 GraphColors('gaxis')
GraphLegendBackground Affects background color of the legend.	Color Transparency	Colors('glegend') not set
GraphTitleText Affects text font & color for title(s).	Font font-attributes* Color	GraphFonts('GraphTitleFont') not set GraphColors('gtext')
* font attributes include FontFamily, FontSize, FontStyle, FontWeight		

Style Element	Recognized Attributes	Attribute Values (DEFAULT Style)
GraphFootnoteText Affects text font & color for footnote(s).	Font font-attributes* Color	GraphFonts('GraphFootnoteFont') not set GraphColors('gtext')
GraphLabelText Affects text font & color for axis labels and legend title.	Font font-attributes* Color	GraphFonts('GraphLabelFont') not set GraphColors('glabel')
GraphValueText Affects text font & color for axis tick values and legend values.	Font font-attributes* Color	GraphFonts('GraphValueFont') not set GraphColors('gtext')
GraphDataText Affects text font and color for point / line labels.	Font font-attributes* Color	GraphFonts('GraphDataFont') GraphColors('gtext')
GraphBorderLines Affects border around graph wall, legend border, borders to complete axis frame.	ContrastColor LineThickness LineStyle	GraphColors('gborderlines') 1px 1
GraphOutlines Affects outline properties for fill areas such as bars, pie slices, boxplots, ellipses, histograms.	ContrastColor LineStyle LineThickness	GraphColors('goutlines') 1 1px
GraphAxisLines Affects X, Y and Z axis lines.	ContrastColor LineStyle LineThickness TickDisplay	GraphColors('gaxis') 1 1px "outside"
GraphGridLines Affects horizontal and vertical grid lines drawn at major tick marks.	ContrastColor LineStyle LineThickness Transparency DisplayOpts	GraphColors('ggrid') 1 1px not set "auto"
GraphReference Affects horizontal and vertical reference lines and drop lines.	ContrastColor LineStyle LineThickness	GraphColors('greferencelines') 1 1px
GraphHeaderBackground Affects background color of the legend title.	Color Transparency	Colors('gheader') not set

* font attributes include FontFamily, FontSize, FontStyle, FontWeight

Style Element	Recognized Attributes	Attribute Values (DEFAULT Style)
GraphUnicodeText Affects text font for Unicode values.	Font font-attributes* Color	GraphFonts('GraphUnicodeFont') not set GraphColors('gtext')
GraphAnnoText Affects annotation text in SGEditor.	Font font-attributes* Color	GraphFonts('GraphAnnoFont') not set GraphColors('gtext')
GraphAnnoLine Affects annotation lines in SGEditor.	ContrastColor LineStyle LineThickness	GraphColors('gcdata') 1 1px
GraphAnnoShape Affects annotation closed shapes (circles, squares, and so on) in SGEditor.	Color ContrastColor MarkerSymbol MarkerSize LineThickness LineStyle Transparency	GraphColors('gdata') GraphColors('gcdata') "starfilled" 12px 2px 1 not set
* font attributes include FontFamily, FontSize, FontStyle, FontWeight		

Style Element	Recognized Attributes	Attribute Values (DEFAULT Style)
TwoColorRamp Affects gradient contours / surfaces / markers / data labels with continuous color response.	StartColor EndColor	GraphColors('gramp2cstart') GraphColors('gramp2cend')
TwoColorAltRamp Affects line contours / markers / data labels with segmented range color response.	StartColor EndColor	GraphColors('gconramp2cstart') GraphColors('gconramp2cend')
ThreeColorRamp Affects gradient contours / surfaces / markers / data labels with continuous color response.	StartColor NeutralColor EndColor	GraphColors('gramp3cstart') GraphColors('gramp3cneutral') GraphColors('gramp3cend')
ThreeColorAltRamp Affects line contours / markers / data labels with segmented range color response.	StartColor NeutralColor EndColor	GraphColors('gconramp3start') GraphColors('gconramp3cneutral') GraphColors('gconramp3end')

Style Element	Recognized Attributes	Attribute Values (DEFAULT Style)
GraphDataDefault Affects primitives related to non-grouped data items; Color applies to filled areas; ContrastColor applies to markers and lines.	Color ContrastColor MarkerSymbol MarkerSize LineStyle LineThickness StartColor NeutralColor EndColor	GraphColors('gdata') GraphColors('gcdata') "circle" 7px 1 1px GraphColors('gramp3cstart') GraphColors('gramp3cneutral') GraphColors('gramp3cend')
GraphOutlier Affects outlier data for the graph.	ContrastColor Color MarkerSize MarkerSymbol LineStyle LineThickness	GraphColors('goutlier') GraphColors('gcoutlier') 7px "circle" 42 2px
GraphFit Affects primary fit line such as a normal density curve.	ContrastColor Color MarkerSize MarkerSymbol LineStyle LineThickness	GraphColors('gcfit') GraphColors('gfit') 7px "circle" 1 2px
GraphFit2 Affects secondary fit line such as a kernel density curve.	ContrastColor Color MarkerSize MarkerSymbol LineStyle LineThickness	GraphColors('gcfit2') GraphColors('gfit2') 7px "X" 4 2px
GraphConfidence Affects primary confidence lines and bands. Color for bands, ContrastColor for lines.	ContrastColor Color MarkerSize MarkerSymbol LineStyle LineThickness Transparency	GraphColors('gcconfidence') GraphColors('gconfidence') 7px "triangle" 1 1px not set

Style Element	**Recognized Attributes**	**Attribute Values (DEFAULT Style)**
GraphConfidence2 Affects secondary confidence lines and bands. Color for bands, ContrastColor for lines.	ContrastColor Color MarkerSize MarkerSymbol LineStyle LineThickness Transparency	GraphColors('gcconfidence2') GraphColors('gconfidence2') 7px "diamond" 4 1px not set
GraphPrediction Affects prediction lines. Color for bands, ContrastColor for lines.	ContrastColor Color MarkerSize MarkerSymbol LineStyle LineThickness Transparency	GraphColors('gcpredict') GraphColors('gpredict') 7px "plus" 4 2px not set
GraphPredictionLimits Affects prediction limits. Color for bands, ContrastColor for lines.	ContrastColor Color MarkerSize MarkerSymbol LineStyle LineThickness Transparency	GraphColors('gcpredictlim') GraphColors('gpredictlim') 7px "chain" 2 1px not set
GraphError Affects error line or error bar fill. ContrastColor for lines, Color for bar fill.	ContrastColor Color MarkerSize MarkerSymbol LineStyle LineThickness Transparency	GraphColors('gcerror') GraphColors('gerror') 7px "asterisk" 1 1px not set
GraphBand Affects display options for confidence bands.	DisplayOpts	"fill"
GraphBox Affects display options for box plots.	DisplayOpts CapStyle Connect	"fill caps mean median outliers" "serif" "mean"

Style Element	Recognized Attributes	Attribute Values (DEFAULT Style)
GraphBoxMean Affects marker for mean.	ContrastColor MarkerSize MarkerSymbol	GraphColors('gcdata') 9px "diamond"
GraphBoxMedian Affects line for median.	ContrastColor LineStyle LineThickness	GraphColors('gcdata') 1 1px
GraphBoxWhisker Affects box whiskers and serifs.	ConstrastColor LineStyle LineThickness	GraphColors('gcdata') 1 1px
GraphContour Affects display options for contours.	DisplayOpts StartColor NeutralColor EndColor	"LabeledLineGradient" GraphColors('gramp3cstart') GraphColors('gramp3cneutral') GraphColors('gramp3cend')
GraphEllipse Affects display options for confidence ellipses.	DisplayOpts	"outline"
GraphHistogram Affects display options for histograms.	DisplayOpts	"fill outline"
GraphBlock Affects fill color for block plots.	Color	GraphColors('gblock')
GraphAltBlock Affects alternate fill color for block plots.	Color	GraphColors('gablock')
GraphConnectLine Affects line for connecting boxes or bars.	ContrastColor LineStyle LineThickness	GraphColors('gcdata') 1 1px

Style Element	Style Attributes	Attribute Values (Default Style)
GraphData1 Affects primitives related to 1st grouped data items. Color applies to filled areas, ContrastColor applies to markers and lines.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata1') GraphColors('gcdata1') 'circle' 1 not set not set

Style Element	Style Attributes	Attribute Values (Default Style)
GraphData2 Affects primitives related to 2nd grouped data items.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata2') GraphColors('gcdata2') 'plus' 4 not set not set
GraphData3 Affects primitives related to 3rd grouped data items.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata3') GraphColors('gcdata3') 'x' 8 not set not set
GraphData4 Affects primitives related to 4th grouped data items.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata4') GraphColors('gcdata4') 'triangle' 5 not set not set
GraphData5 Affects primitives related to 5th grouped data items.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata5') GraphColors('gcdata5') 'square' 14 not set not set
GraphData6 Affects primitives related to 6th grouped data items.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata6') GraphColors('gcdata6') 'asterisk' 26 not set not set
GraphData7 Affects primitives related to 7th grouped data items.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata7') GraphColors('gcdata7') 'diamond' 15 not set not set

Style Element	Style Attributes	Attribute Values (Default Style)
GraphData8 Affects primitives related to 8th grouped data items.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata8') GraphColors('gcdata8') not set 20 not set not set
GraphData9 Affects primitives related to 9th grouped data items.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata9') GraphColors('gcdata9') not set 41 not set not set
GraphData10 Affects primitives related to 10th grouped data items.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata10') GraphColors('gcdata10') not set 42 not set not set
GraphData11 Affects primitives related to 11th grouped data items.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata11') GraphColors('gcdata11') not set 2 not set not set
GraphData12 Affects primitives related to 12th grouped data items.	Color ContrastColor MarkerSymbol LineStyle MarkerSize LineThickness	GraphColors('gdata12') GraphColors('gcdata12') not set not set not set not set

Graphical Style Attributes

Style Attribute	Type	Examples

CapStyle Affects shape of line at end of box whisker.	string Valid Values: "serif" | "line" | "bracket "	CapStyle="line"
Color Affects background color of the graph, walls, or floor. Affects the color of text.	color	Color= colors("docbg") Color=blue
Connect Affects box plot connect line.	string Valid Values: "median" | "mean" | "Q1" | "Q3" | "min" | "max"	Connect="median"
ContrastColor Affects color of line or marker.	color	ContrastColor= GraphColors("data") ContrastColor=#ffffff
DisplayOpts Affects displayed features of box plots, ellipses, histograms, bands, contours, and grid lines.	string of one or more options	DisplayOpts="fill | caps | mean | median | outliers | connect | notches" DisplayOpts="fill | outline" DisplayOpts="fill | outline" DisplayOpts="fill | outline" DisplayOpts="LabeledLineGradient" DisplayOpts="auto | on | off "
Font Affects all text font attributes	aggregate definition in parentheses	Font=(" 'Courier New', Courier, monospace", 4, bold italic)
FontFamily Affects font family.	string	FontFamily="Courier New"
FontSize Affects font size.	dimension or integer 1-7 indicating relative size	FontSize=10pt FontSize=3
FontStyle Affects font style.	enumeration Valid Values: italic | roman | slant	FontStyle=italic
FontWeight Affects font weight.	enumeration Valid Values: light | medium | bold	FontWeight=bold
FrameBorder Affects graph wall border.	boolean Valid Values: On | Off	FrameBorder=on

EndColor Affects contours, and also gradient legends. Final color that is used with a 2- or 3-color ramp.	color	EndColor=blue
LineStyle Affects border lines, axis lines, grid lines, and reference lines.	positive integer Valid Values: 1 (solid line) 2 - 46 (dashed/dotted line)	LineStyle=2
LineThickness Affects width thickness of lines.	dimension	LineThickness=2px LineThickness=2%
MarkerSize Affects marker size.	dimension	MarkerSize=5px MarkerSize=3%
MarkerSymbol Affects marker used.	string	MarkerSymbol="circle"
NeutralColor Affects contours, and also gradient legends. Middle color that is used with 3-color ramp.	color	NeutralColor=white
StartColor Affects contours, and also gradient legends. Initial color that is used with a 2- or 3-color ramp.	color	StartColor=red
TickDisplay Affects placement of all axis tick marks.	string Valid Values: "inside" | "outside" | "across"	TickDisplay = "across"
Transparency Affects backgrounds, fills, lines, and markers.	number Valid Values: 0.00 (opaque) 1.00 (transparent)	Transparency=0.25

Appendix 3
Values for Marker Symbols and Line Patterns

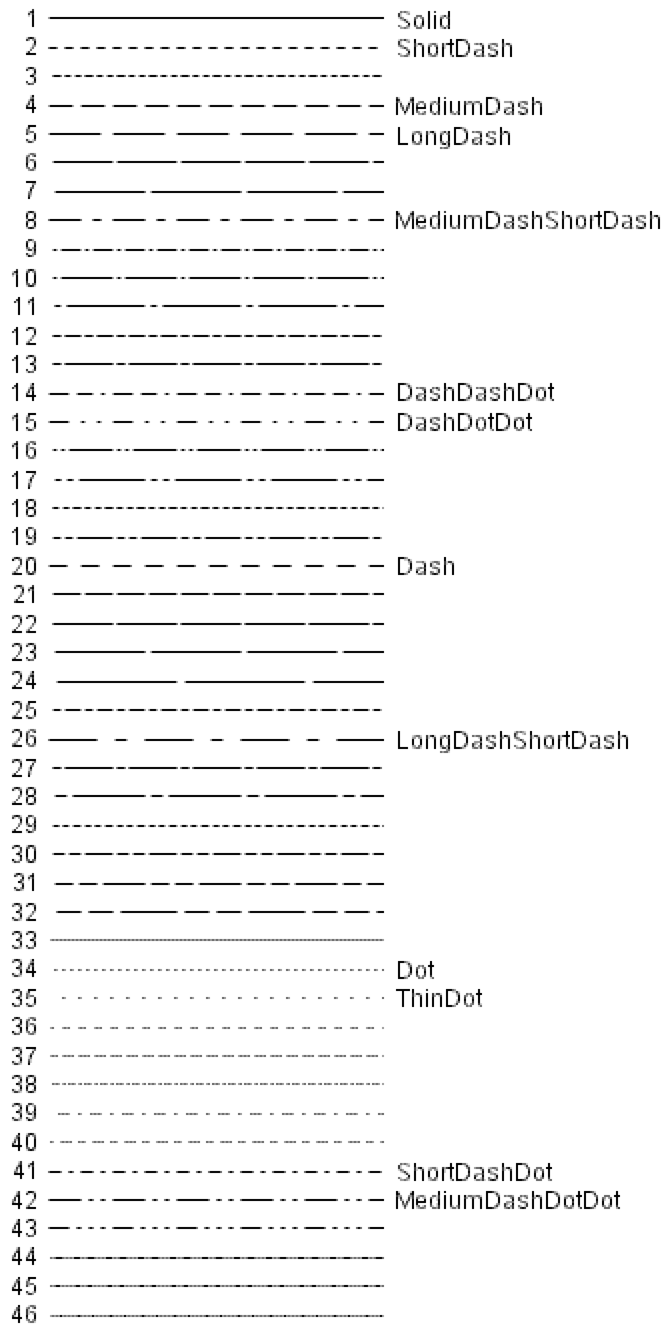
Values for Marker Symbols

The following symbols can be used with the Graphics Template Language:

ArrowDown	HomeDown	Tilde	CircleFilled
Asterisk	Ibeam	Triangle	DiamondFilled
Circle	Plus	Union	HomeDownFilled
Diamond	Square	X	SquareFilled
GreaterThan	Star	Y	StarFilled
Hash	Tack	Z	TriangleFilled

Values for Line Patterns

The following line patterns can be used with the Graphics Template Language. A line pattern can be specified by its number or name. Not all patterns have names. We recommend that you use the named patterns because they have been optimized to provide good discriminability when used in the same plot.


1 Solid
2 ShortDash
3
4 MediumDash
5 LongDash
6
7
8 MediumDashShortDash
9
10
11
12
13
14 DashDashDot
15 DashDotDot
16
17
18
19
20 Dash
21
22
23
24
25
26 LongDashShortDash
27
28
29
30
31
32
33
34 Dot
35 ThinDot
36
37
38
39
40
41 ShortDashDot
42 MediumDashDotDot
43
44
45
46

Appendix 4
SAS Formats Not Supported

Using SAS Formats

SAS formats can be assigned to input data columns with the FORMAT statement of the SGRENDER procedure. Additionally, several GTL statement options enable a SAS format as an option value. Examples include the TICKVALUEFORMAT= option for formatting axis tick values, and the TIPFORMAT= option for formatting data tips.

Not all SAS formats are supported in the GTL or with the SGPLOT, SGSCATTER, SGPANEL, and SGRENDER procedures. The tables in the following sections show the character and numeric SAS formats that are not supported.

When the GTL encounters an unsupported format, a note similar to the following is written to the SAS log:

```
NOTE: TICKVALUEFORMAT=bestx. is invalid. The format is invalid or unsupported.
      The default will be used.
```

Unsupported Numeric Formats

The following numeric formats are not supported in the GTL:

BESTD	BESTX	D	FLOAT	FRACT
FREE	IB	IBR	IEEE	IEEER
ODDSR	PCPIB	PD	PIB	PIBR

PK	RB	SSN	WORDF	WORDS
Z	ZD			

Unsupported Date and Time Formats Related to IS0 8601

The following date and time formats are not supported in the GTL:

$N8601B	$N8601BA	$N8601E	$N8601EA	$N8601EH
$N8601EX	$N8601H	$N8601X	B8601DA	B8601DN
B8601DT	B8601DZ	B8601LZ	B8601TM	B8601TZ
E8601DA	E8601DN	E8601DT	E8601DZ	E8601LZ
E8601TM	E8601TZ	IS8601DA	IS8601DN	IS8601DT
IS8601DZ	IS8601LZ	IS8601TM	IS8601TZ	

Other Unsupported Date and Time Formats

The following date and time formats are not supported in the GTL:

HDATE	HEBDATE	JDATEMDW	JDATEMNW	JDATEWK
JDATEYDW	JDATEYM	JDATEYMD	JDATEYMW	JDATEYT
JDATEYTW	JNENGO	JNENGOT	JNENGOTW	JNENGOW
JTIMEH	JTIMEHM	JTIMEHMS	JTIMEHW	JTIMEMW
JTIMESW	MDYAMPM	MINGUO	NENGO	NLDATEYQ
NLDATEYR	NLDATEYW	NLDATMYQ	NLDATMYR	NLDATMYW
NLSTRMON	NLSTRQTR	NLSTRWK	PDJULG	PDJULI
TWMDY	XYYMMDD	YYQZ		

Unsupported Currency Formats

The following currency formats are not supported in the GTL:

EURFRATS	EURFRBEF	EURFRCHF	EURFRCZK	EURFRDEM
EURFRDKK	EURFRESP	EURFRFIM	EURFRFRF	EURFRGBP
EURFRGRD	EURFRHUF	EURFRIEP	EURFRITL	EURFRLUF
EURFRNLG	EURFRNOK	EURFRPLZ	EURFRPTE	EURFRROL
EURFRRUR	EURFRSEK	EURFRSIT	EURFRTRL	EURFRYUD
EURTOATS	EURTOBEF	EURTOCHF	EURTOCZK	EURTODEM
EURTODKK	EURTOESP	EURTOFIM	EURTOFRF	EURTOGBP
EURTOGRD	EURTOHUF	EURTOIEP	EURTOITL	EURTOLUF
EURTONLG	EURTONOK	EURTOPLZ	EURTOPTE	EURTOROL
EURTORUR	EURTOSEK	EURTOSIT	EURTOTRL	EURTOYUD

Appendix 5
Memory Management for ODS Graphics

ODS Graphics uses Java technology to produce its graphs. Most of the time this fact is transparent to you because the required Java Runtime Environment (JRE) and Jar files are included with SAS software installation and the Java environment is automatically started and stopped for you. When Java is started, it allocates a fixed amount of memory that can grow up to the value set for the **`-Xmx`** suboption in the JREOPTIONS option (discussed in a moment). This memory is independent of the memory limit that SAS sets for the SAS session with its MEMSIZE= option.

Normally, the memory limit for Java is sufficient for most ODS Graphics applications. However, some tasks are very memory intensive and might exhaust all available Java memory, resulting in an OutOfMemoryError condition. You might encounter Java memory limitations when

- the product of the output size and the DPI setting results in very large output
- a classification panel has a very large number of classifier crossings
- a scatterplot matrix has a large number of variables
- creating 3D plots and 2D contours, which are memory intensive to generate
- a plot has a very large number of marker labels
- a plot uses many character variables or has a large number of GROUP values
- using the SG Editor to edit a graph with a large amount of data.

If you encounter a Java OutOfMemoryError, you can try executing your program again by restarting SAS and specifying a larger amount of memory for Java at SAS invocation.

To determine what the current Java memory settings are, you can submit a PROC OPTIONS statement that will show the value of the JREOPTIONS option:

```
proc options option=jreoptions;
run;
```

After you submit this procedure code, a list of JREOPTIONS settings is displayed in the SAS log. The JREOPTIONS option has many suboptions that configure the SAS Java environment. Many of the suboptions are installation and host specific and should not be modified, especially the ones that provide installed file locations. For managing memory, look for the **`-Xms`** and **`-Xmx`** suboptions:

`-Xms`

Use this option to set the minimum Java memory (heap) size, in bytes. Set this value to a multiple of 1024 greater than 1MB. Append the letter k or K to indicate kilobytes, or m or M to indicate megabytes. The default is 2MB. Examples:

```
-Xms6291456
-Xms6144k
-Xms6m
```

-Xmx

Use this option to set the maximum size, in bytes, of the memory allocation pool. Set this value to a multiple of 1024 greater than 2MB. Append the letter k or K to indicate kilobytes, or m or M to indicate megabytes. The default is 64MB. Examples:

```
-Xmx83886080
-Xmx81920k
-Xmx80m
```

As a general rule, you should set the minimum heap size (**-Xms**) equal to the maximum heap size (**-Xmx**) to minimize garbage collections.

Typically, SAS sets both **-Xms** and **-Xmx** to be about 1/4 of the total available memory or a maximum of 128M. However, you can set a more aggressive maximum memory (heap) size, but it should never be more than 1/2 of physical memory.

You should be aware of the maximum amount of physical memory your computer has available. Let us assume that doubling the Java memory allocation is feasible. So when you start SAS from a system prompt, you can add the following option:

```
-jreoptions (-Xmx256m -Xms256m)
```

Aternatively, you might need to specify the setting in quotation marks:

```
-jreoptions '(-Xmx256m -Xms256m)'
```

The exact syntax varies for specifying Java options, depending on your operating system, and the amount of memory that you can allocate varies from system to system. The set of JRE options must be enclosed in parentheses. If you specify multiple JREOPTIONS system options, SAS appends JRE options to JRE options that are currently defined. Incorrect JRE options are ignored.

If you choose to create a custom configuration file, you would simply replace the existing **-Xms** and **-Xmx** suboption values in the JREOPTIONS=(all Java options) portion of the configuration file.

For more information, see the SAS Companion for your operating system.

Glossary

anti-aliasing
a rendering technique for improving the appearance of text and curved lines in a graph by blurring the jagged edges normally present. The degree of improvement is relative to the nature of the graphical content (for example, vertical and horizontal lines do not benefit from anti-aliasing). Extra processing is required to perform anti-aliasing.

attribute bundle
a common collection of visual properties associated with a graphical primitive such as a line, marker, or text. For example, all lines have visual properties of pattern, thickness, and color. All markers have visual properties of symbol, size, weight, and color. Attribute bundles can be associated with style elements in order to indirectly assign visual properties. See also marker properties, line properties, and text properties.

axis
a graphical element used to locate or identify the values of other graphical elements, such as points or bars. An axis consists of an axis line with tick marks, tick values, and a descriptive label. Not all parts of the axis need to be displayed. An axis is typically an interval axis (linear, log, or time) or a discrete axis. A two-dimensional graph can have up to four independent axes: X, Y, X2, and Y2. A three-dimensional graph has three independent axes: X, Y, and Z. See also Cartesian coordinate system.

axis offset
the gaps that normally appear at the ends of an axis line. The gaps enable markers, bars, and other graphic primitives that are drawn at extreme data values to be rendered without clipping. An offset can also be used to add extra space between an axis line and visual elements in the graph. An offset distance is expressed as a value from 0 to 1, which represents a percentage of axis length to the axis data minimum or axis data maximum. An offset can be specified for either end of any axis.

axis threshold
a numerical bias from 0 to 1 that determines whether an extra tick is added at either end of a non-discrete, interval axis. If the minimum and maximum thresholds are set to 0, then no ticks are added beyond the actual data range. If both minimum and maximum thresholds are set to 1, then the data range is completely bounded by the first and last ticks.

axis tick
a short line segment perpendicular to the axis line. A tick can cross the axis line, or be drawn from the axis inside or outside the wall.

axis tick value
a formatted data value represented by a tick.

axis type
a keyword denoting axis functionality. For example, the axis type of interval axes can be LINEAR, TIME, or LOG. The axis type of a discrete axis is DISCRETE.

axis viewport
the range of values displayed on an interval axis. This range can be larger or smaller than the actual data range of the axis. An axis viewport that is larger than the data range effectively zooms out from the plot or plots. An axis viewport that is smaller than the data range zooms in on the plot or plots.

background
a property of a layout container, a legend, or text. A background can be fully transparent or fully opaque. If opaque, the background is drawn with a fill color.

band plot
a plot that draws a horizontal band using two Y values for each X value, or that draws a vertical band using two X values for each Y value. A band plot is typically used to show confidence, error, prediction, or control limits. The points on the upper and lower band boundaries can be joined to create two outlines, or the area between the boundaries can be filled.

binned data
data that has been summarized or transformed in some way to facilitate its rendering by a parameterized plot. Continuous numeric data is typically binned by setting a bin width (interval size) and then computing the number of bins, or by setting the number of bins and computing the bin width. A histogram is often used to represent binned data.

bins
numeric intervals into which continuous numeric data can be categorized.

block
another term for statement block.

block plot
a plot that displays one or more rectangles (blocks) along an axis, where each rectangle identifies a block of consecutive observations having the same value for a specified block variable. The first block begins at the start of the axis (mapped to the values of a specified variable), and represents the first observation's block value, and continues through consecutive observations having the same block value. A change in the block variable's value ends the first block and starts the second, which continues through consecutive observations until the block value changes again. The last block extends to the end of the axis.

border
the outermost outline of a graph, a layout container, or a graphical element (for example, a legend or text).

category variable
a classification variable with a finite number of distinct (discrete) values. These variables are typically used to split data into subsets. For example, in a bar chart, each unique value is displayed as a bar on a DISCRETE axis.

cell
a distinct rectangular subregion of a graph that can contain plots, text, or legends. A display unit of a layout container. Some layouts, such as OVERLAY, have only one cell, which has one plot area. Other layouts, such as GRIDDED, have a rectangular grid of cells. Each cell can also be populated with another layout container or be left empty.

cell block
a block beginning with a CELL statement and ending with an ENDCELL statement that defines the graphical content of a cell. The cell block is available only within a LATTICE layout.

cell header
a graphical element (typically text or a legend) that is aligned at the top of a cell and provides information about the cell contents. A cell header is defined within a cell block, which is available only within a LATTICE layout.

child block
a block that is contained within another block when two or more blocks are nested. For example, a CELLHEADER block is always a child of a CELL block.

classification level
for a single classification variable, each unique value is regarded as a classification level. For two or more variables, a classification level is one of the unique combinations (crossings) of the unique values of each variable. For example, if three variables have four, two, and three distinct values, there are 24 classification levels.

classification panel
a multi-cell graph created by a layout, such as DATAPANEL, in which the number of cells is determined by the number of classification levels of one or more classification variables. Each cell displays a common plot based on subsets of the input data.

classification variable
a variable whose values classify the observations in a data set into different groups that are meaningful for analysis.

clip
to truncate a plot or graphical element (such as a line, marker, or band) when it reaches a boundary such as a plot wall.

column
a vertical component of a table. Each column has a unique name, contains data of a specific type, and has certain attributes. A column is analogous to a variable in SAS terminology. In the Graph Template Language a column can also be a set of layout cells, stacked vertically and sharing the same alignment.

column axis
an external axis appearing above or below a column of cells and serving as a common reference for the column of a multi-cell layout, such as a LATTICE, DATAPANEL, or DATALATTICE layout.

column gutter
the space between columns of cells in a multi-cell layout.

column header
text that labels the column contents in a multi-cell layout. This text can be aligned above or below the cells in a column. In a LATTICE layout, the column header is not restricted to text (it can contain a plot or a legend, for example).

column major order
an order for populating cells of a layout or entries in a legend when the number of rows is specified. By default, cells or entries are filled starting from the top left and moving down. When the bottom row of the first column is filled, a new column begins filling to the right of the previous column, and so on until all content items have been placed in cells or entries. There might be empty cells or entries in the last column.

column weight
in a LATTICE layout, the proportion of width allotted to a specific column of the layout. The sum of all column weights is 1.

computed plot
a plot in which input data is internally summarized or otherwise transformed to create new data that is actually rendered by the plot. Examples of computed plot statements are BARCHART, BOXPLOT, HISTOGRAM, ELLIPSE, and REGRESSIONPLOT.

conditional logic
syntax that enables one set of statements or an optional alternate set of statements to execute at run time. In the Graph Template Language, an IF/ENDIF block defines conditional logic: IF (condition) statements; ELSE statements; ENDIF; The ELSE statement is not required.

continuous legend
a legend that shows a mapping between a color ramp or color segments and corresponding numeric values. Plots that support a COLORMODEL= option can use this type of legend.

crossing
a combination of the unique values of one or more classification variables. See also classification level.

cube
in three-dimensional graphics, the outlines formed by the intersection of three pairs of parallel planes; each pair is orthogonal to the primary X, Y, and Z axes. The display of the cube is optional.

data object
a transient version of a SAS data set created by ODS. When an input SAS data set is bound to a compiled graph template, an ODS data object is created, based on all the columns requested in the template definition and any new columns that have been directly or indirectly computed. A data object can persist when used with the ODS OUTPUT statement.

data tips
data or other detailed information that is displayed when a user positions a mouse pointer over an element in a graph. For example, a data tip typically displays the data value that is represented by a bar, a plot point, or some other element.

define block
in the TEMPLATE procedure, a define block (beginning with a DEFINE statement and ending with an END statement) creates various types of templates, including STATGRAPH, STYLE, and TABLE.

dependent plot
a plot that cannot be rendered by itself. Dependent plots must be overlaid with a stand-alone plot. Dependent plots do not provide data ranges to establish axes. REFERENCELINE, DROPLINE, and LINEPARM statements produce dependent plots. See also stand-alone plot.

dependent variable
a variable that is observed to change in response to the independent variables. In a function y=f(x), the value of the dependent variable y is a function of the independent variable x. For example, in a Graph Template Language REGRESSIONPLOT statement, the Y variable is the dependent variable.

design size
the intended size of a graph that is specified in the graph template definition. The DESIGNHEIGHT and DESIGNWIDTH options of the BEGINGRAPH statement set the intended height and width, which are used to determine the scale factors when the graph is resized. The intended height and width are used unless overridden by the ODS Graphics statement HEIGHT or WIDTH options when the template is executed.

device-based graphic
a graph created with traditional SAS/GRAPH software that requires DEVICE specification. ODS graphics (template-based graphics) do not use device technology.

discrete axis
an axis for categorical data values. The distance between ticks has no significance. A bar chart always has a discrete axis.

discrete legend
a legend that provides values or descriptive information about graphical elements in a grouped or overlaid plot.

dots per inch
a measure of the graph resolution by its dot density. Short form: DPI.

DPI
See dots per inch.

drop line
a line drawn from a point in the plot area perpendicular to an axis.

dynamic variable
a variable defined in a template with the DYNAMIC statement that can be initialized at template run time.

equated axes
in two-dimensional plots, axes that use the same drawing scale (ratio of display distance to data interval) on both axes. For example, an interval of 2 on the X axis maps to the same display distance as an interval of 2 on the Y axis. The aspect ratio of the plot display equals the aspect ratio of the plot data. In other words, a 45-degree slope in data will be represented by a 45-degree slope in the display. Equated axes are always of

TYPE=LINEAR. The number of intervals displayed on each axis does not have to be the same.

external axis
an axis that is outside all cells of a layout. An external axis represents a common scale for all plots in a row or column of a multi-cell layout.

fill
to apply a color within a bounded area. Many plots, such bar charts and band plots, have bounded areas that can be filled or unfilled. When filled, a color is applied. When unfilled, the areas are transparent.

fit policy
one of several algorithms for avoiding tick-value collision when space allotted to a predefined area does not permit all the text to fit. For example, an axis might have a THIN policy that eliminates the display of tick values for alternate ticks. A ROTATE policy would turn the tick values at a 45-degree angle. A TRUNCATE policy would truncate all long tick values to a fixed length and add an ellipsis (. . .) at the end to imply truncation. A STAGGER policy would create two rows of tick values with consecutive tick values alternating between rows. A compound policy such as STAGGERROTATE could be used to automatically choose the best fit policy for the situation.

footnote area
the region below the graph area where text produced by ENTRYFOOTNOTE statements appears.

frequency variable
in an input data set, a non-negative and non-zero integer variable that represents the frequency of occurrence of the current observation, essentially treating the data set as if each observation appeared <userSuppliedValue>n</userSuppliedValue> times, where <userSuppliedValue>n</userSuppliedValue> is the value of the FREQ variable for the observation.

fringe plot
a plot consisting of short, equal-length line segments drawn from and perpendicular to an axis. Each observation of a numeric variable corresponds to the location for a line segment.

function
a computational routine that returns a value. In the Graph Template Language, all SAS functions that can be used in SAS WHERE expressions are supported. Many functions for statistical computations are available in the Graph Template Language.

glyph
a letter, character, punctuation mark, pictogram, or symbol that is rendered in the context of some written language. A typeface (font) consists of a coordinated set of glyphs. See also Unicode.

graph
a visualization created by SAS/GRAPH software. A graph that is created by the ODS Graphics system can contain titles, footnotes, legends, and one or more cells, and is typically saved as an image or an SGE file. A generic term for final graphical output without regard to content or format.

graph(ical) area
the region where the visualization displays between the title area and footnote area. The graphical area consists of one or more cells. See also title area and footnote area.

graph(ics) template
See ODS Graphics template.

grid
rows and columns of a multi-cell layout.

gridded data
input that contains at least three numeric variables. Two of the variables are treated as X and Y variables and the third variable Z is treated as if it were a function of X and Y. The X and Y variable values occur at uniformly spaced intervals (although the size and number of intervals might be different for X and Y). All X,Y pairs are unique, and Z values are interpolated so that every X,Y pair has a Z value. Raw data that has at least three numeric variables can be converted to gridded data with the G3GRID procedure (in SAS/GRAPH). The procedure offers both bivariate and spline interpolation methods for computing Z values.

group index
a numeric variable with positive integer values that correspond to values of a group variable. The index values are used to associate GraphData1 – GraphData<userSuppliedValue>N</userSuppliedValue> style elements with group values.

group variable
an optional classification variable supported by many plot types that enables the data for each distinct group value to be rendered in a visually different manner. For example, a grouped scatter plot displays a distinct marker and color for each group value. A series plot displays a distinct line pattern and color for each group value.

gutter
the space between columns or rows of cells in a multi-cell layout.

image format
a file format that displays a graphical representation. PNG, GIF, TIFF, and JPEG are examples of image formats, each with different characteristics.

image map
in an HTML file, the information contained in the map tag. This information can be used by a browser to display data tips on the image. See also data tip.

independent variable
a variable that persists and affects dependent variables. For a function y=f(x), the value of the dependent variable y is a function of the independent variable x. For example in a REGRESSIONPLOT statement, the X= variable is the independent variable.

inset
a graphical element such as a legend, line of text, or a table of text that is embedded inside of a graph's plot area.

interval axis
an axis where the distance between tick marks represents monotonically increasing or decreasing numeric units of some scale (like a ruler). The standard interval axis is called a LINEAR axis. Specialized interval axes include a TIME axis and a LOG axis.

layout
a generic term for a rectangular container that lays out the positions and sizes of its child components.

layout block
a block beginning with a LAYOUT statement and ending with an ENDLAYOUT statement.

layout grid
a multi-cell layout arranged as a grid of cells in rows and columns.

layout type
a keyword indicating the functionality of the layout. For example OVERLAY, LATTICE, and DATAPANEL are layout types.

legend
a compound graphics element that provides information about other graphical elements in plots. See also discrete legend and continuous legend.

legend entry
a combination of a graphical element such as a marker or line along with text describing the value or use of the graphical element. A discrete legend can have several legend entries.

legend title
text that explains how to interpret the legend.

line property
a value that defines the pattern, thickness, or color of a line. By default, the value for a line property is derived from a style element in the current style. See also attribute bundle.

linear axis
an interval axis with ticks placed on a linear scale.

log axis
an axis displaying a logarithmic scale. A log axis is useful when data values span orders of magnitude.

macro variable reference
in a template definition, a reference to a macro variable that has been declared with MVAR or NMVAR statements. These references are meant to be resolved at template run time and should not be preceded with an ampersand. If a standard macro variable reference (a name preceded with an ampersand) appears in a template definition, it is resolved at template compile time.

marker
(1) a symbol such as a circle, triangle, or diamond that is used to indicate the location of a data point in a plot. (2) a type of annotation that is used in SAS/GRAPH ODS Graphics Editor to highlight particular data in a plot or graph.

marker property
a value that defines the symbol used as a marker, or its size, weight, or color. By default, the value for a marker property is derived from a style element in the current style. See also attribute bundle.

multi-cell layout
a layout that supports a rectangular grid of cells, each of which can contain a graphical element, such as a plot, a legend, a nested layout, and so on.

nested layout
a layout block that appears within the scope of another layout block.

ODS
See Output Delivery System.

ODS Graphics
an extension to ODS that is used to create analytical graphs using the Graph Template Language.

ODS Graphics Editor
an interactive application that can be used to edit and annotate ODS Graphics output.

ODS Graphics template
a template of the type STATGRAPH that is defined with the TEMPLATE procedure. A graphics template contains the definition of a graph (as Graph Template Language statements) and references to data columns.

opaque
a property of a background. Opaque backgrounds are filled with a color. Non-opaque backgrounds are transparent.

outlier
a data point that differs from the general trend of the data by more than is expected by chance alone. An outlier might be an erroneous data point or one that is not from the same sampling model as the rest of the data.

Output Delivery System
a component of SAS software that can produce output in a variety of formats such as markup languages (HTML, XML), PDF, listing, RTF, PostScript, and SAS data sets. Short form: ODS.

overlay
a plot that can be superimposed on another plot when specified within an overlay-type layout. A common overlay combination is a fit line on a scatter plot.

overlay layout
a type of layout that supports the superimposition of graphical components, such as plots, legends, and nested layouts.

padding
space added inside the border of a graphical component, such as a layout or a legend.

panel
a graph with multiple cells.

parameterized plot
a non-computed plot that requires parameterized data. The Graph Template Language offers several plots in both computed and parameterized versions, for example, BARCHART and BARCHARTPARM. Some computed plots such as REGRESSIONPLOT can be emulated with a SERIESPLOT if the input data represented points on a fit line.

parent block
when two or more blocks are nested, any layout block that contains one or more layout blocks is a parent of the contained blocks.

plot
a visual representation of data such as a scatter plot, a series line, or a histogram. In the ODS Graphics context, plot is a generic term for the graphical element or elements drawn by a plot statement. Multiple plots can be overlaid in a cell to create a graph.

plot area
the space, bounded by the axes, where a visual representation of data, such as a scatter plot, a series line, or a histogram, is drawn.

plot type
a plot family such as bar chart (which would include horizontal, vertical, and grouped bar charts), or a classification scheme for plots based on some useful criteria, such as whether the plots are computed or parameterized.

primary axis
the X or Y axis contrasted to the X2 or Y2 secondary axis.

primary plot
the plot in an overlay that determines axis features, such as axis type and axis label.

prototype layout
an overlay plot composite that appears in each cell of a classification panel. Each instance of the prototype represents a different subset (classification level) of the data.

regression plot
a straight or curved line showing a linear or higher order regression fit for a set of points.

required argument
a variable or constant that must be specified in order to evaluate an expression or render a plot, legend, text, or a layout. For example, a scatter plot has two required arguments: X=column and Y=column.

response variable
See dependent variable.

rich text
a generic term for text that can have different font characteristics (color, family, size, weight, style) on a character-by-character basis and can also be used as a superscript or subscript. All text statements in GTL support rich text.

role
a generic term for the purpose a variable serves in a plot or the keyword used to designate the assigned variable. All plots have predefined roles and most plots support user-defined roles that can be used for data tips. For example, a series plot has predefined roles named for X , Y, GROUP, DATALABEL, CURVELABEL, and INDEX. Additional roles can be added to specify the content of data tips.

row
a set of layout cells that are side-by-side and share the same alignment.

row axis
an external axis appearing on the left or right of a row of cells in a multi-cell layout.

row gutter
space between rows of cells of a multi-cell layout.

row header
typically, the text that identifies the row contents in a multi-cell layout. This text can be aligned to the right or left of the cells in a row. The row header is not restricted to text (it can contain a plot or a legend, for example).

row major order
an order for populating cells of a layout or entries of a legend when the number of columns is specified. For example, in the default case: Start at the top left and fill cells or entries left-to-right. When the right-most column is filled, begin a new row below the previous row. Continue this until all content items have been placed in cells or entries. There might be empty cells/entries in the last row.

row weight
in a LATTICE layout, the proportion of height allotted to a specific row of the layout. The sum of all row weights is 1.

secondary axis
the X2 or Y2 axis as contrasted to the X or Y primary axis.

SGE file
a file created in the ODS Graphics environment that contains an editable graph. Such files have a .SGE file extension and can be edited only with the ODS Graphics Editor. You can edit SGE files from the SAS Results window or by opening the SGE file from within the ODS Graphics editor.

sidebar
an area of certain multi-cell layouts external to the grid of cells where text or other graphical elements can appear. The LATTICE, DATAPANEL, and DATALATTICE layout support four sidebar areas (TOP, BOTTOM, LEFT, and RIGHT).

single-cell layout
a layout type that supports only one cell. The OVERLAY, OVERLAY3D, and OVERLAYEQUATED layouts are examples of single-cell layouts.

sparse data
in classification panels with two or more classifiers, some crossings of the classification values might not be present in the input data. Such input data is called sparse data. By default, a DATAPANEL layout does not generate cells for sparse data, but if requested, it can produce empty cells as place holders for the non-existent crossings.

stand-alone plot
a plot that has its own data range and can therefore appear by itself in a layout.

statement block
a group of statements that has both a logical beginning and ending statement. For example, a LAYOUT statement along with its ENDLAYOUT statement and all contained statements are a block. Some blocks can be nested within other blocks.

style
an ODS template that can be used to control the visual aspects (colors, fonts, lines, markers, and so on) of a graph or table. A style consists of many style elements and each style element is made up style attributes. Style templates are created with the DEFINE STYLE statement of the TEMPLATE procedure.

style attribute
a visual property such as a color, line pattern, or font property that has a reserved name. For example, COLOR, FONTFAMILY, FONTSIZE, FONTWEIGHT, and FONTSTYLE are all styles attributes of the style elements such as GraphTitleText, GraphLabelText, and so. Style attributes are collectively referenced by a style element within a style definition.

style element
a named collection of style attributes that affects a specific part of ODS output. For example, the GraphTitleText style element specifies the color and font properties of title text and possibly other text in the graph. See also style attribute.

style reference
a part of the Graph Template Language syntax that indicates the current value of a specific attribute of a specific style element. For example, SIZE=GraphTitleText:FontSize means to assign to SIZE the value of the FontSize attribute of the GraphTitleText style element from the current style.

template
a compiled entry in a template store (item store). Common templates types include STATGRAPH, STYLE, and TABLE.

template compile time
the phase when the source program of a template definition is submitted. The syntax of the definition is evaluated for correctness. If no errors are detected, the definition is converted to a binary format and stored for later access.

template definition
the TEMPLATE procedure source program that creates a template. A template definition can be generated from a compiled template. Also called the template source.

template run time
the actions performed when a compiled template is bound to a data object and then rendered to produce a graph. Run-time errors can occur that prevent a graph from being produced.

template store
an item store that contains definitions that were created by the TEMPLATE procedure. Definitions that SAS provides are in the item store Sashelp.Tmplmst. You can store definitions that you create in any template store to which you have write access. See also item store.

template-based graphic
SAS/GRAPH output where a compiled ODS template of the type STATGRAPH is used to produce graphical output, that is, a graph produced within the ODS graphics environment as opposed to a graph produced in the traditional device-based environment.

text properties
a common set of characteristics that can be specified for any text string: COLOR, FAMILY, SIZE, WEIGHT, and STYLE. By default, values for these properties are derived from a style element in the current style. See also attribute bundle and style attribute.

time axis

an axis type that displays only SAS date, time, or datetime values. Axis tick value increments can be specified as time or date intervals, such as MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR.

title area

the region above the graph area where text produced by ENTRYTITLE statements appears.

transparency

the degree to which a graphic element (such as a marker or filled area) is opaque or transparent. Transparency is indicated with a number from 0 (completely opaque) to 1 (completely transparent).

Unicode

an encoding system that provides a single comprehensive mapping of all characters (glyphs) in all languages to unique numeric values called code points.

viewport

See axis viewport.

wall

the area bounded by orthogonal axis pairs. In two-dimensional graphs, there is one wall bounded by the XY axes. In three-dimensional graphs, there are three walls, bounded by the XY, YZ, and XZ axes. A wall has an optional outline and can be opaque or transparent.

weight variable

a positive numeric variable in the input data set that represents a weight to be applied to the current observation.

Index

G

H

I

M

Your Turn

We welcome your feedback.

- If you have comments about this book, please send them to **yourturn@sas.com**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **suggest@sas.com**.