SAS® 9.4 Graph Template Language: User's Guide, Fifth Edition

SAS® Documentation
January 11, 2024

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS® 9.4 Graph Template Language: User's Guide, Fifth Edition*. Cary, NC: SAS Institute Inc.

SAS® 9.4 Graph Template Language: User's Guide, Fifth Edition

ISBN 978-1-62960-809-9 (PDF)

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

January 2024

9.4-P9:grstatug

Contents

PART 8 Graphical Output 489

PART 9 Graph Templates 593

About This Book

Using This Book

Prerequisites

This document is written for users who are experienced in using SAS. You should understand the concepts of programming in the SAS language. The following table summarizes the SAS tasks that you need to understand in order to use the Graph Template Language (GTL).

Task	Documentation
Invoke SAS at your site	Instructions provided by the on-site SAS support personnel
Use Base SAS software Use the DATA step to create and manipulate SAS data sets Use the SAS windowing environment or SAS Enterprise Guide to enter, edit, and submit program code	Base documentation library: ■ *SAS Programmer's Guide: Essentials* ■ *SAS Data Set Options: Reference* ■ *SAS Formats and Informats: Reference* ■ *SAS Functions and CALL Routines: Reference* ■ *SAS DATA Step Statements: Reference* ■ *SAS System Options: Reference* ■ *Base SAS Utilities: Reference*
Allocate SAS libraries and assign librefs Create external files and assign filerefs	Documentation for using SAS in your operating environment: ■ *SAS Companion for Windows* ■ *SAS Companion for UNIX Environments* ■ *SAS Companion for z/OS*

Task	Documentation
Manipulate SAS data sets by using SAS procedures	*Base SAS Procedures Guide*

About the Examples in This Book

The example programs that are shown in this document often provide all of the code that you need to generate the graphs that are shown in the figures. You can copy and paste the example code into your SAS session and generate the graphs yourself. You can run the example code in the SAS windowing environment and in SAS Studio. Unless otherwise noted, the examples use the default ODS destination for generating results. In the SAS windowing environment, the default ODS destination is ODS HTML. For information about results in SAS Studio, see *SAS Studio: User's Guide* for your version of SAS Studio. You can find the documentation for all versions of SAS Studio on the SAS Studio documentation page on support.sas.com.

Some of the examples in this book require a custom ODS output environment. In those examples, you close the default ODS destination or destinations, and then you use ODS statements to create the custom output environment. The code for these examples is written to work in both the SAS windowing environment and in SAS Studio.

If you generate the example graphs using an HTML destination, they are typically rendered as 640 pixel by 480 pixel images using the HTMLBlue style. Some of the examples show you how to change the graph size and style. The graphs shown for those examples are rendered in the specified size and style.

Because of size limitations, the graphs in this document are not shown in their default size of 640 pixels by 480 pixels. They are scaled down to meet the size requirements of our documentation production system. When graphs that are produced with ODS Graphics are reduced in size, several automatic processes take place to optimize the appearance of the output. Among the differences between default size graphs and smaller graphs are that the smaller graphs have scaled down font sizes. Also, their numeric axes might display a reduced number of ticks and tick values. Thus, the graphs that you generate from the example programs are not always identical to the graphs that are shown in the figures. However, both graphs accurately represent the data.

When you produce your own graphical output, you can change the graph size and attributes, if needed. Chapter 27, "Managing Your Graph's Appearance," on page 491 and Chapter 28, "Managing Your Graphics Output," on page 567 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.

Syntax Conventions for the SAS Language

Overview of Syntax Conventions for the SAS Language

SAS uses standard conventions in the documentation of syntax for SAS language elements. These conventions enable you to easily identify the components of SAS syntax. The conventions can be divided into these parts:

- syntax components
- style conventions
- special characters
- references to SAS libraries and external files

Syntax Components

The components of the syntax for most language elements include a keyword and arguments. For some language elements, only a keyword is necessary. For other language elements, the keyword is followed by an equal sign (=). The syntax for arguments has multiple forms in order to demonstrate the syntax of multiple arguments, with and without punctuation.

keyword
specifies the name of the SAS language element that you use when you write your program. Keyword is a literal that is usually the first word in the syntax. In a CALL routine, the first two words are keywords.

In these examples of SAS syntax, the keywords are bold:

CHAR (*string, position*)

CALL RANBIN (*seed, n, p, x*);

ALTER (*alter-password*)

BEST *w*.

REMOVE <*data-set-name*>

In this example, the first two words of the CALL routine are the keywords:

CALL RANBIN(*seed*, *n*, *p*, *x*)

The syntax of some SAS statements consists of a single keyword without arguments:

DO;
... *SAS code* ...
END;

Some system options require that one of two keyword values be specified:

DUPLEX | **NODUPLEX**

Some procedure statements have multiple keywords throughout the statement syntax:

CREATE <UNIQUE> **INDEX** *index-name* **ON** *table-name* (*column-1* <, *column-2*, …>)

argument
specifies a numeric or character constant, variable, or expression. Arguments follow the keyword or an equal sign after the keyword. The arguments are used by SAS to process the language element. Arguments can be required or optional. In the syntax, optional arguments are enclosed in angle brackets (< >).

In this example, *string* and *position* follow the keyword CHAR. These arguments are required arguments for the CHAR function:

CHAR (*string, position*)

Each argument has a value. In this example of SAS code, the argument *string* has a value of 'summer', and the argument *position* has a value of 4:

```
x=char('summer', 4);
```

In this example, *string* and *substring* are required arguments, whereas *modifiers* and *startpos* are optional.

FIND(*string, substring* <, *modifiers*> <, *startpos*>

argument(s)
specifies that one argument is required and that multiple arguments are allowed. Separate arguments with a space. Punctuation, such as a comma (,) is not required between arguments.

The MISSING statement is an example of this form of multiple arguments:

MISSING *character(s)*;

<LITERAL_ARGUMENT> *argument-1* <<LITERAL_ARGUMENT> *argument-2* ... >
specifies that one argument is required and that a literal argument can be associated with the argument. You can specify multiple literals and argument pairs. No punctuation is required between the literal and argument pairs. The ellipsis (...) indicates that additional literals and arguments are allowed.

The BY statement is an example of this argument:

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* …>;

argument-1 <options> <argument-2 <options> ...>
specifies that one argument is required and that one or more options can be associated with the argument. You can specify multiple arguments and associated options. No punctuation is required between the argument and the option. The ellipsis (...) indicates that additional arguments with an associated option are allowed.

The FORMAT procedure PICTURE statement is an example of this form of multiple arguments:

PICTURE name <(*format-options*)>
<*value-range-set-1* <(*picture-1-options*)>
<*value-range-set-2* <(*picture-2-options*)> …>>;

argument-1=value-1 <argument-2=value-2 ...>
specifies that the argument must be assigned a value and that you can specify multiple arguments. The ellipsis (...) indicates that additional arguments are allowed. No punctuation is required between arguments.

The LABEL statement is an example of this form of multiple arguments:

LABEL *variable-1=label-1 <variable-2=label-2 …>*;

argument-1 <, argument-2, ...>
specifies that one argument is required and that you can specify multiple arguments that are separated by a comma or other punctuation. The ellipsis (...) indicates a continuation of the arguments, separated by a comma. Both forms are used in the SAS documentation.

Here are examples of this form of multiple arguments:

AUTHPROVIDERDOMAIN (*provider-1:domain-1 <, provider-2:domain-2, …>*

INTO :*macro-variable-specification-1 <, :macro-variable-specification-2, …>*

Note: In most cases, example code in SAS documentation is written in lowercase with a monospace font. You can use uppercase, lowercase, or mixed case in the code that you write.

Style Conventions

The style conventions that are used in documenting SAS syntax include uppercase bold, uppercase, and italic:

UPPERCASE BOLD
identifies SAS keywords such as the names of functions or statements. In this example, the keyword ERROR is written in uppercase bold:

ERROR <*message*>;

UPPERCASE
identifies arguments that are literals.

In this example of the CMPMODEL= system option, the literals include BOTH, CATALOG, and XML:

CMPMODEL=BOTH | CATALOG | XML |

italic
identifies arguments or values that you supply. Items in italic represent user-supplied values that are either one of the following:

- nonliteral arguments. In this example of the LINK statement, the argument *label* is a user-supplied value and therefore appears in italic:

  LINK *label*;

- nonliteral values that are assigned to an argument.

  In this example of the FORMAT statement, the argument DEFAULT is assigned the variable *default-format*:

  FORMAT *variable(s) <format* > <DEFAULT = *default-format*>;

Special Characters

The syntax of SAS language elements can contain the following special characters:

=

an equal sign identifies a value for a literal in some language elements such as system options.

In this example of the MAPS system option, the equal sign sets the value of MAPS:

MAPS=*location-of-maps*

< >

angle brackets identify optional arguments. A required argument is not enclosed in angle brackets.

In this example of the CAT function, at least one item is required:

CAT (*item-1* <, *item-2*, …>)

|

a vertical bar indicates that you can choose one value from a group of values. Values that are separated by the vertical bar are mutually exclusive.

In this example of the CMPMODEL= system option, you can choose only one of the arguments:

CMPMODEL=BOTH | CATALOG | XML

...

an ellipsis indicates that the argument can be repeated. If an argument and the ellipsis are enclosed in angle brackets, then the argument is optional. The repeated argument must contain punctuation if it appears before or after the argument.

In this example of the CAT function, multiple *item* arguments are allowed, and they must be separated by a comma:

CAT (*item-1* <, *item-2*, …>)

'*value*' or "*value*"

indicates that an argument that is enclosed in single or double quotation marks must have a value that is also enclosed in single or double quotation marks.

In this example of the FOOTNOTE statement, the argument *text* is enclosed in quotation marks:

FOOTNOTE <*n*> <*ods-format-options* '*text*' | "*text*">;

;

a semicolon indicates the end of a statement or CALL routine.

In this example, each statement ends with a semicolon:

```
data namegame;
   length color name $8;
   color = 'black';
   name = 'jack';
   game = trim(color) || name;
run;
```

References to SAS Libraries and External Files

Many SAS statements and other language elements refer to SAS libraries and external files. You can choose whether to make the reference through a logical name (a libref or fileref) or use the physical filename enclosed in quotation marks.

If you use a logical name, you typically have a choice of using a SAS statement (LIBNAME or FILENAME) or the operating environment's control language to make the reference. Several methods of referring to SAS libraries and external files are available, and some of these methods depend on your operating environment.

In the examples that use external files, SAS documentation uses the italicized phrase *file-specification*. In the examples that use SAS libraries, SAS documentation uses the italicized phrase *SAS-library* enclosed in quotation marks:

```
infile file-specification obs = 100;
libname libref 'SAS-library';
```

Value Type Notation Used for GTL Statement Options

The value type notation that is used in the syntax descriptions and in some examples in this document are as follows:

boolean
specifies a literal value that resolves to true or false. The following table lists literal values that resolve to true or false.

Boolean Values

Values That Resolve To True	Values That Resolve To False
ON	OFF
_ON_	_OFF_
TRUE	FALSE
YES	NO
YES	_NO_
1	0

color
specifies a string that identifies a color. A color can be one of the following:

- any of the color names that are supported by SAS. See "Color-Naming Schemes" on page 673.
- one of the colors that exists in the SAS session when the style template is used, such as DMSBLACK or DMSCYAN. (Use these color specifications only if you are running SAS in the windowing environment.)
- an English description of an Hue/Light/Saturation (HLS) value. Such descriptions use a combination of words to describe the lightness, the saturation, and the hue (in that order). You can use the Color Naming System to form a color by doing one of the following:
  - combining a chromatic hue with a lightness, a saturation, or both
  - combining the achromatic hue gray with a lightness
  - combining the achromatic hue black or white without qualifiers.
  - combining words to form a wide variety of colors, such as light vivid green, dark vivid orange, or light yellow.
- specify hues that are intermediate between two neighboring colors. To do so, combine one of the following adjectives with one of its neighboring colors: brownish, greenish, purplish, or yellowish (for example, bluish purple or reddish orange).

column
specifies a column variable that contains either double-precision values or string values, or a dynamic variable that refers to such a column.

See also: *integer-column*, *numeric-column*, and *string-column*.

dimension
specifies a nonnegative number. The number can be followed by one of the following optional units of measure:

Units for Dimension

Unit	Description
CM	centimeters
IN	inches
MM	millimeters
PCT or %	percentage
PT	point size (72 points = 1 inch)
PX	pixels

expression
specifies a selective, relational, or logical program structure that calculates values when those values are not stored in the data. The expression must be

specified as an EVAL() argument. The following shows the structure of an EVAL() argument:

```
x = EVAL(expression)
```

The expression returns a number and can be formed with consonants, data columns, dynamic variables, functions, or other expressions. The following example uses the data column Time and the SGE functions MEAN and ACF:

```
EVAL(MEAN(Time) + ACF(Time, NLags=10))
```

For more information about expressions, see "Expressions" in *SAS Graph Template Language: Reference*.

fill-pattern
specifies a fill pattern as a two-character code that consists of a line-direction prefix (R for right, L for left, or X for cross hatch) and a line identification number, 1–5. For more information about fill patterns, see "Fill Pattern Options" on page 665.

format
specifies a SAS format or a user-defined format.

integer, integer-column
specifies a member of the set of positive whole numbers, negative whole numbers, and zero.

An integer column specifies a column that contains integer values, or a dynamic variable that refers to such a column.

line-pattern-name, line-pattern-number
specifies a string value of a line pattern, a numeric value of a line pattern, a dynamic variable that contains such a string or number, or a style reference to a line pattern. Line patterns are chosen for discriminability. Because of different densities, equal weighting is impossible for lines of the same thickness. Instead, line patterns are ordered to provide a continuum of weights, which is useful when displaying confidence bands.

For details about line attributes, see "Line Options" on page 666.

marker-name
specifies a string value of a maker symbol, a dynamic variable that contains a marker symbol, or a style reference to a marker symbol.

For details about marker attributes, see "Marker Options" on page 668.

number, numeric-column
specifies a value, a dynamic variable that contains a double-precision value, an expression that resolves to a double-precision value, or a style reference to a double-precision value.

A *numeric-column* specifies a column that contains double-precision values, or a dynamic variable that refers to such a column.

string, string-column
specifies a quoted character string.

A *string-column* specifies a column that contains string values, or a dynamic variable that refers to such a column.

Note: For quoted character string options in GTL, a space enclosed in quotation marks (" " or ' ') and empty quotation marks ("" or '') are not equivalent. A space enclosed in quotation marks specifies a blank space or a missing string value. Empty quotation marks have the same effect as not setting the option. To specify a blank space or missing value in a quoted string option, use a space enclosed in quotation marks (" " or ' ').

style-reference

specifies a reference to an attribute that is defined in a style element.

In the ODS Graphics templates that SAS provides, options for plot features are specified with a style reference in the form *style-element-name:attribute-name*, rather than a specific value. For example, the symbol, color, and size of markers for a basic scatter plot is specified in a SCATTERPLOT statement as follows:

```
scatterplot x=X y=Y /
  markersymbol=GraphDataDefault:markersymbol
  markercolor=GraphDataDefault:contrastcolor
  markersize=GraphDataDefault:markersize
```

The above style references guarantee a common appearance for markers used in all basic scatter plots. For non-grouped data, the marker appearance is controlled by the GraphDataDefault style element in the style template that you specify.

In order to create your own style template, or to modify a style template to use with ODS Graphics, you need to understand the relationship between style elements and graph features. For more information, see the usage guide.

Recommended Reading

Here is the recommended reading list for this title:

- *Statistical Graphics Procedures by Example*
- *Clinical Graphs Using SAS®*
- *SAS Graphics for Clinical Trials by Example*
- *The Little SAS® Book: A Primer*

PART 1

Getting Started

1

Get Started Using the Graph Template Language

Introduction to the Graph Template Language

What Is the Graph Template Language?

The Graph Template Language (GTL) is the heart of ODS Graphics. All of the graphs that are created by the SAS analytical procedures and by the SAS 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 guide, which helps you understand important concepts and offers many complete code examples that illustrate frequently used features. You also need *SAS Graph Template Language: Reference*, which is the language dictionary for GTL.

When You Need to Use GTL

You might need to use GTL in the following cases:

- You need to modify graphs that are created by analytical procedures. 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. For every graph created by these procedures, a corresponding template is stored in the Sashelp.Tmplmst item store. To customize these templates, you must have a basic understanding of GTL.
- You need graphs of data before you can start an analysis of the data. Or, you need graphs to visualize the results of a complex analysis involving multiple procedures or DATA steps. 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 have a basic understanding of GTL.

What You Can Create Using GTL

Single-Cell Graphs

The following table lists the single-cell plots that you can create using GTL.

Category	Plot		
2-D plots	Axis tables	Ellipse plots	Polygon plots
	Band plots	Fringe plots	Regression plots
	Bar charts	Heat maps	Scatter plots
	Bar-Line charts	High-Low plots	Series plots
	Block plots	Histograms	Spline plots
	Box plots	Line charts	Step plots

Category	Plot		
	Bubble plots	Loess plots	Point and Slope plots
	Contour plots	Model Band plots	Text plots
	Dendrograms	Mosaic plots	Vector plots
	Density plots	Needle plots	Waterfall charts
	Dot plots	Pie charts	
3-D plots	Bivariate histograms		
	Surface plots		

Multicell Graphs

Here are the multicell graphs that you can create using GTL.

- grid graphs
- lattice graphs
- data-driven lattice graphs
- data-driven panel graphs
- scatter plot matrices

Interactive Graphs

GTL enables you to create graphs that display additional information when certain user actions occur. You can add data tips to your graphs, which display information when the mouse pointer is positioned on an element in your graph. The data tip can display data or additional details. You can create drill-down graphs, which link elements in your graph to resources such as other graphs, detailed descriptions, and so on. When a linked element in your graph is clicked, the linked resource opens in a new browser window. You can create multiple levels of drill-down graphs to enable users to explore your data on their own. GTL also enables you to create animated graphs, which display graphs in a slide-show fashion to show trends in your data over time.

What You Need to Know to Get Started Using GTL

How Graphs Are Creating Using GTL

Here are the basic steps for creating a graph using GTL:

1. Define the structure of the graph by using the GTL syntax in a DEFINE STATGRAPH block in a TEMPLATE procedure step.
2. Run the TEMPLATE procedure to compile and save the template definition.
3. Run the SGRENDER procedure and specify the template and graph data set by name to generate the graph.

Graph Templates

What Is a Graph Template?

In ODS Graphics, a graph template is an ODS template of type STATGRAPH, which defines the content of a graph. A STATGRAPH template is created using the TEMPLATE procedure. Using the TEMPLATE procedure to create a STATGRAPH template is described in "TEMPLATE Procedure: Creating ODS Graphics" in *SAS Output Delivery System: Procedures Guide*. The graph template contains GTL statements that define the graph layout and content. GTL templates are rendered using the SGRENDER procedure, which specifies a data source that contains appropriate data values and the GTL template to use for rendering the graph. The SGRENDER procedure is described in "SGRENDER Procedure" in *SAS ODS Graphics: Procedures Guide*.

Structure of a Minimal Graph Template

GTL uses a structured building-block approach to defining a graph. The syntax provides a set of layout statements, plot statements, and other statements that define the graph. The following pseudo code shows a minimal GTL template. Callouts identify the statements.

Example Code 1.1 *A Minimal Template Definition*

```
proc template;     1
   define statgraph template-name; 2
      begingraph; 3
         layout layout-name; 4
            GTL-statements   5
         endlayout;
      endgraph;
   end;
run;
```

The statements are as follows:

1. The PROC TEMPLATE statement compiles and saves an ODS template.
2. The DEFINE statement starts the template definition block, and specifies the template type STATGRAPH (a graph template) and the template name. The END statement terminates the DEFINE block.
3. The BEGINGRAPH statement starts the graph statement block, and the ENDGRAPH statement terminates the graph statement block. GTL statements are placed inside the graph statement block.
4. The LAYOUT *layout-name* statement starts the outer graph layout block, and the ENDLAYOUT statement terminates the outer layout block. GTL statements are placed inside the layout block.
5. One or more GTL statements define the content of the graph.

All GTL templates must include these statements. You can add other GTL statements as needed. GTL statements are described in Chapter 6, “Overview of the GTL Statements,” on page 31. Templates are described in more detail later in this guide.

Compiling and Saving the Template

Run the TEMPLATE procedure with your template code to compile and save your GTL template. When the TEMPLATE procedure is executed, it compiles and stores your template automatically. A note similar to the following is written to the SAS log, indicating where the template was stored.

```
NOTE: STATGRAPH 'template-name' has been saved to: SASUSER.TEMPLAT
```

To list the templates that are stored in the Sasuser.Template store, use the following TEMPLATE procedure step:

```
proc template;
  list / store=sasuser.templat;
run;
```

To write the code for a template that is stored in the Sasuser.Template store to the SAS log, use the following TEMPLATE procedure step:

```
proc template;
   path sasuser.templat;
   source template-name;
```

```
run;
```

Creating a Graph from a Graph Template

Run the SGRENDER procedure to generate a graph from a graph template. Here is an example.

```
proc sgrender data=data-source template=template-name;
run;
```

The TEMPLATE= option specifies the name of the template. The SGRENDER statement searches for the template in the SAS template stores in a predefined order. If the specified template is not in any of the template stores, an error results. To see the current template stores and the template store search order, use the following command:

```
ods path show;
```

Here is an example of the output.

```
Current ODS PATH list is:

1. SASUSER.TEMPLAT(UPDATE)
2. SASHELP.TMPLMST(READ)
```

For templates that are stored in the first path listed, which is Sasuser.Templat in this example, you need to specify only the template name. The DATA= option specifies the source data for the graph, such as Sashelp.Cars.

Template stores are described in *SAS Output Delivery System: User's Guide*. The SGRENDER procedure is described in *SAS ODS Graphics: Procedures Guide*.

Graph Layouts

A layout in GTL is a single-cell or multicell container for your graph content. A layout is defined by a layout statement block, which contains GTL statements that create the graph content. Your template code must define one top-level layout to contain the content for your graph. The top-level template block can contain GTL statements that generate plots, text, legends, and so on, as well as other layouts. GTL provides several statements for single-cell and multicell layouts. The GTL layout statements are described in "Layout Statements" on page 34.

Axis Management

The GTL layout statements provide options that enable you to control various attributes of your graph axes such as the following:

- axis labels
- axis ranges

- tick values and tick-value formats
- tick marks
- grid lines

The axis-management options in GTL layout statements are described in “Axis Management in GTL Layouts” on page 37.

Plots

GTL supports a wide variety of 2-D and 3-D plots. A plot statement generates each plot. Use one or more plot statements in your template code to create the primary content for your graph. The GTL plot statements are described in “Plot Statements” on page 31.

Legends

GTL provides two basic types of legends: discrete legends and continuous legends. Merged legends are also supported, which enable you to consolidate legends for two plots into one legend. You can also create custom items for your legends.The statements for basic legends are described in “Legend Statements” on page 32. The more advanced legend features are described in “Specialized Legends” on page 18.

Titles, Footnotes, and Other Text

GTL provides text statements that enable you to add titles, footnotes, and other text to your graphs. You can add multiple titles and footnotes outside the graph area, and you can add text inside the graph area. The text statements are described in Chapter 19, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 317.

Learning By Example: Create Your First Bar Chart Using GTL

About the Scenario in This Topic

This topic presents an example that demonstrates how to generate a simple bar chart using GTL. When you have completed this example, you will know the basic

steps that are needed to create a plot using GTL. In this example, you will perform the following tasks:

- Use the ODS GRAPHICS statement to set graphics options for your graph.
- Use ODS statements to specify an output document format and style for your output.
- Use the TEMPLATE procedure to create your graph template.
- Use the SGRENDER procedure to render your graph from your graph template.

Creating a Bar Chart Using GTL

Here is a summary of the steps for this example.

1. Identify the plot data. In this example, data set Sashelp.Cars is used.
2. Sort the data in data set Sashelp.Cars by variable Type and store the result in data set Work.Cars.
3. Specify the height and width of the graph.
4. Specify the output document format as HTML, the output style as Daisy, and the output filename as mychart.html.
5. Use the TEMPLATE procedure to create statgraph template MYGRAPHTEMPLATE. In the template code, do the following:
   - Specify a title and footnote for the graph.
   - Create an OVERLAY layout for the graph, and specify labels for the category and response axes.
   - Create a vertical bar chart.
   - Create a legend for the bar chart.
6. Use the SGRENDER procedure to render the graph. Specify the input data set and the name of the graph template.

Here is the SAS code.

```
ods _all_ close;                                                /*
1 */
ods graphics / reset width=460px height=440px;                  /*
2 */
ods html style=Daisy path="." file="mychart.html";     .         /*
3 */

proc sort data=sashelp.cars out=cars;                           /*
4 */
   by type;
run;

proc template;                                                  /*
5 */
   define statgraph mygraphtemplate;
      begingraph;
```

```
        entrytitle "Models Produced By Vehicle Type and Origin"; /*
6 */
        entryfootnote "Data: SASHELP.CARS";
        layout overlay /                                         /*
7 */
           xaxisopts=(label="Vehicle Type")                      /*
8 */
           yaxisopts=(label="Models Produced"
              griddisplay=on gridattrs=(color=gray pattern=dot));
           barchart category=type / name="bar"                   /*
9 */
              stat=freq barlabel=true
              group=origin;
           discretelegend "bar" / title="Origin";                /*
10 */
        endlayout;                                               /*
11 */
      endgraph;
   end;
run;

proc sgrender data=cars template=mygraphtemplate;                /*
12 */
run;

ods html close;                                                  /*
13 */
ods html;                                                        /*
14 */
```

1 The ODS _ALL_ CLOSE statement closes all currently open ODS destinations in order to conserve resources.

2 The ODS GRAPHICS statement resets all of the ODS Graphics options to their default values and then specifies the width and height of the graph.

3 The ODS HTML statement opens the HTML destination. Option STYLE=DAISY specifies the style. Option PATH= specifies an output directory for the ODS output, and option FILE= specifies filename Mychart.html as the name of the output file.

Note: You must have Write access to the path that you specify in the PATH= option.

4 The SORT procedure step sorts data set Sashelp.Cars by variable Type in ascending order and stores the result in data set Work.Cars.

5 The TEMPLATE procedure statement begins the TEMPLATE procedure step. The DEFINE STATGRAPH MYGRAPHTEMPLATE specifies a statgraph template named MYGRAPHTEMPLATE. The BEGINGRAPH statement opens the graph definition block.

6 The ENTRYTITLE and ENTRYFOOTNOTE statements specify a title and footnote for the graph. By default, the title is centered at the top of the graph, and the footnote is centered at the bottom.

7 The LAYOUT OVERLAY statement opens a LAYOUT OVERLAY statement block.

8 The XAXISOPTS option specifies a label for the category axis. The YAXISOPTS option specifies a label for the response axis and grid lines at the major tick marks.

9 The BARCHART statement generates a vertical bar chart. Option CATEGORY= specifies variable Type as the category variable. Option NAME= specifies a name that will be used to associate a legend with this bar chart. The STAT= option requests frequency as the bar statistic, and GROUP= specifies variable Origin as the group variable. Option BARLABEL= specifies that the frequency statistic for each be displayed above each bar.

10 The DISCRETELEGEND statement creates a discrete legend for the bar chart named BAR.

11 The ENDLAYOUT statement closes the LAYOUT OVERLAY statement block.

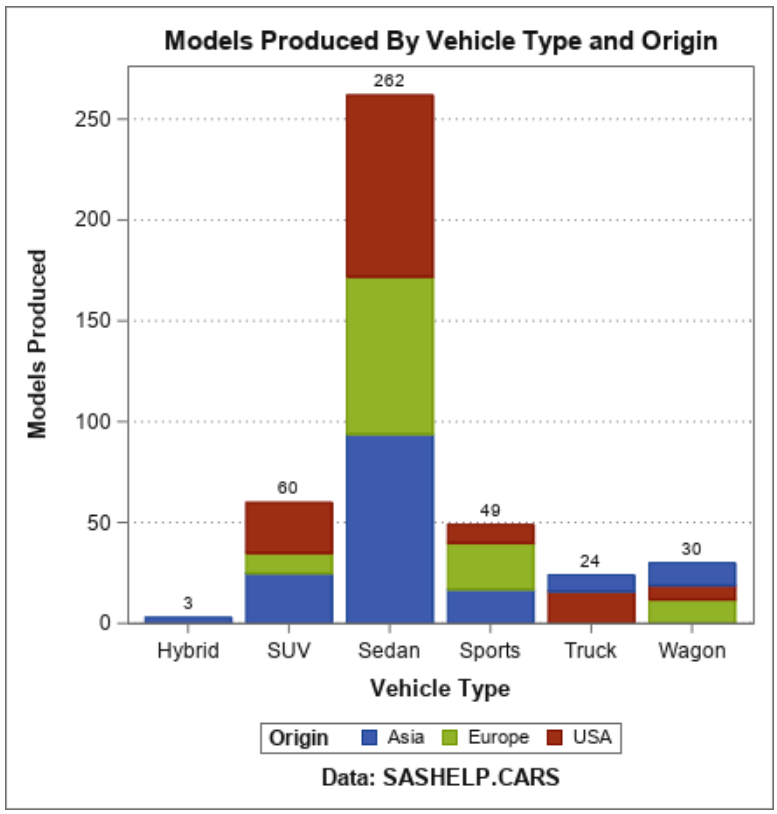
12 The SGRENDER procedure step renders the graph. Option DATA= specifies the input data, and option TEMPLATE= specifies the name of the graph template. When the procedure is executed, the graph is displayed in the Results window.

13 The ODS HTML CLOSE statement closes the HTML destination.

14 The ODS HTML statement opens the HTML destination for subsequent procedure executions. This step is not required if you are using SAS Studio.

Here is the output for this example.

Figure 1.1 *Example Program Output*



Using what you have learned from this example, you can begin exploring how to create other plots using GTL.

Where to Go from Here

Now that you have a basic understanding of how to use GTL, review the remaining topics in this book to expand on your knowledge of the language. The following table provides suggestions for your next steps.

Table 1.1 *Next Steps*

Task	Reference
See examples of the types of graphs that you can create using GTL.	Chapter 10, “Gallery of the GTL Plots,” (p. 61)
Learn how to perform common tasks that are associated with developing GTL templates.	Chapter 3, “Common Tasks Associated with Creating GTL Templates,” (p. 21)
Explore the other features of GTL.	Chapter 2, “A Quick Look at Additional Features of GTL,” (p. 15)
Identify resources that can help you develop your GTL templates.	Chapter 4, “Additional Resources to Help You Develop Your GTL Templates,” (p. 25)

2

A Quick Look at Additional Features of GTL

Appearance Options

GTL provides a variety of ways in which you can customize the appearance of your graphs. For the entire graph, you can specify a different ODS style when you execute your template. For specific elements of your graph, you can use the attribute override options, when available, in your GTL statements. The ways in which you can customize graph appearance are described in “Methods for Changing the Appearance of Your Plots” on page 494.

Attribute Maps

For better control over the visual attributes of your graph, you can use an attribute map to apply one or more visual attributes to specific group values or ranges of color-response values. An attribute map enables you to create graphs with consistent visual properties over multiple data sets. Attribute maps map a discrete value or range of color-response values to a set of one or more visual attribute specifications. You can specify markers, colors, line styles, and so on. GTL supports two types of attribute maps: discrete attribute maps and range attribute maps.

Attribute maps are described in "Key Concepts for Using Attribute Maps" in *SAS Graph Template Language: Reference*.

Custom Plot Markers

GTL provides wide range of markers that you can use in your plots. You can also create your own custom markers from a Unicode character or from an image. You can use your custom markers in any plot that supports markers. Creating and using custom plot markers is described in "Custom Marker Definition Statements" in *SAS Graph Template Language: Reference*.

Annotations

Types of Annotations That Are Supported in GTL

You can add annotations to your graphs that call attention to specific aspects of your data or provide additional information. Annotations can be text, lines, arrows, various geometrical shapes, and images. There are two methods of creating annotations in GTL: using drawing statements in your GTL code or using the ODS Graphics annotate facility.

Drawing Statements

GTL provides several drawing statements that you can include in your graph template code to draw annotations anywhere on your graph. The annotations can

be used to describe aspects of your graphs, create custom graph features such as a broken axis, or add a corporate logo to you graphs. The draw statements can be independent of the graph data or data-dependent. The annotations are drawn by statements in your template code. Therefore, to modify your annotations, you must modify your template code, and then recompile and re-execute your template.

Using the draw statements to annotate your graph is described in Chapter 22, “Adding Code-Driven Graphics Elements to Your Graph,” on page 425.

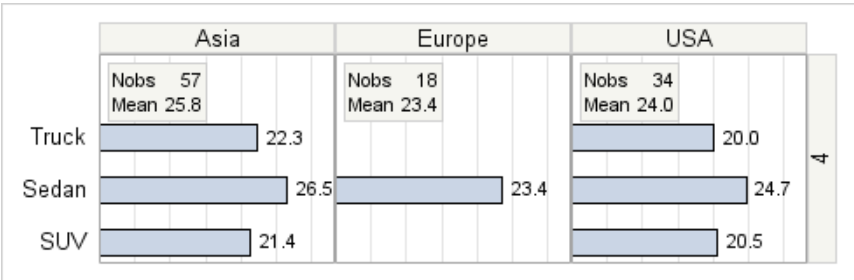
ODS Graphics Annotate Facility

The ODS Graphics annotate facility enables you to store annotate instructions in a SAS data set, and then apply the annotate instructions to a graph at execution time. Unlike the drawing statements, the annotate drawing instructions are independent of the graph template code. To modify your annotations, you modify your annotate data set, and then execute your template, or you execute your template using a different annotate data set. To apply an annotate data set to your graph, you must first add the ANNOTATE statement to your graph template code, and then specify the SGANNO= option in your SGRENDER procedure statement.

Using the ODS Graphics annotate facility to annotate your graphs is described in Chapter 23, “Adding Data-Driven Annotations to Your Graph,” on page 439.

Insets

You can add insets to your graphs to display supplemental information in the plot area of your graph. The inset text can be a string or a table of text. Insets are typically created using ENTRY statements in a LAYOUT GRIDDED block. Options in the SCATTERPLOTMATRIX statement, and in the LAYOUT DATAPANEL and LAYOUT DATALATTICE statements also enable you to create insets. Here is an example of insets in a classification panel.



Insets are described in Chapter 21, “Adding Insets to Your Graph,” on page 383.

Specialized Legends

Axis Legends

An axis legend is a legend of axis tick values. It is used for discrete axes when the tick values do not fit within the axis area. When used, the axis tick values are replaced with consecutive integers. An axis legend is then created that associates the tick integer values with the actual tick values. Axis legends are described in “Extracting Discrete Axis Tick Values into a Legend” on page 144.

Merged Legends

You can merge two discrete legends from two grouped plots with line and marker overlays into one legend. Each item in the merged legend combines the visual attributes of the lines and markers from each plot. Merged legends are described in “Merging Legend Items from Two Plots into One Legend” on page 368.

Custom Legend Items

You can add items to your discrete legend that do not appear in your plots. The items can be a line, marker, marker and line, fill color swatch, or text. You can use this feature to add information to your legend or to create a common legend that you can use in multiple graphs. Creating and using custom legend items is described in “Adding Items to a Discrete Legend” on page 364.

Axis Tables

You can add an axis-aligned table of values along an axis in your graph to display additional information. You can place the table above or below your plot in the plot area, or you can place it in an upper or lower margin. Axis tables are described in “Creating an Axis-Aligned Inset with an Axis Table” on page 417.

Run-Time Programming Features

GTL supports run-time features that enable you to create flexible graph templates. The features include:

- Dynamic variables and macro variables, which enable you to pass information into your templates at execution time. Dynamic variables are described in "Dynamic Variables and Macro Variables" in *SAS Graph Template Language: Reference*.
- Functions and expressions, which enable you to compute option values at execution time. The functions and expressions that you can use in your graph templates are described in "Functions" in *SAS Graph Template Language: Reference* and "Expressions" in *SAS Graph Template Language: Reference*.
- Conditional logic, which enables you to execute statements conditionally at execution time. Using conditional logic in your graph templates is described in "Conditional Logic" in *SAS Graph Template Language: Reference*.

3

Common Tasks Associated with Creating GTL Templates

About the GTL Tasks

This chapter lists common tasks that are associated with GTL. For each task, one or more links are provided to information about how to complete the task. Because there are many tasks associated with GTL, the tasks are categorized as follows:

- output environment
- graph creation
- graph enhancement

The task lists in this chapter can help you quickly locate information in this reference that you need to create your graphs.

Output Environment Tasks

The following table lists tasks that are related to establishing the output environment for GTL.

Task	Reference
Select an ODS destination for your output.	"Getting Started with the Output Delivery System" in *SAS Output Delivery System: User's Guide*
Select an ODS style for your output.	"Using ODS Styles to Control Graph Appearance" (p. 495)
Specify an output directory or folder for your image output.	"Controlling the Location of the Image Output" (p. 579)
Change the name of your output image file.	"Specifying and Resetting the Image Name" (p. 574)
Change the format of your image output.	"Controlling the Image Name and Image Format" (p. 574)
Change the size of your graph.	"Controlling Graph Size" (p. 581)
Scale your graph.	"Scaling Graphs" (p. 582)
Change the resolution of your graph.	"Controlling Image Resolution" (p. 586)

Graph Creation Tasks

The following table lists tasks that are related to generating basic plots using GTL.

Task	Reference
Identify the plot or plots that you want to use in your graph.	Chapter 10, "Gallery of the GTL Plots," (p. 61)
Select a layout for your graph.	"Layout Statements" (p. 34)
Create your initial graph template.	"Graph Templates" (p. 6)
Modify the axes in your graph, if necessary.	"Axis Management in GTL Layouts" (p. 37)
Add titles, footnotes, and other text elements to your graph.	Chapter 19, "Adding Titles, Footnotes, and Text Entries to Your Graph," (p. 317)
Add a legend to your graph.	Chapter 20, "Adding Legends to Your Graph," (p. 337)

Task	Reference
Use dynamic variables and macro variables to pass information into your template at execution time.	Chapter 30, "Using Dynamic Variables and Macro Variables in Your Templates," (p. 605)
Use conditional logic in your template code for greater flexibility.	"Conditional Logic" (p. 621)
Use expressions in your template code to compute values at execution time.	"Expressions" (p. 619)
Use functions in your template code to compute values at execution time.	Chapter 32, "Using Functions in Your Templates," (p. 627)
Compile your graph template.	"Compiling and Saving the Template" (p. 7)
Execute your graph template.	"Creating a Graph from a Graph Template" (p. 8)

Graph Enhancement Tasks

The following table lists tasks that are related to adding enhancements to your graph.

Task	Reference
Add annotations to your graph.	Chapter 22, "Adding Code-Driven Graphics Elements to Your Graph," on page 425 Chapter 23, "Adding Data-Driven Annotations to Your Graph," on page 439
Add data skins to your graph.	"Using Data Skins" (p. 554)
Add one or more insets to your graph.	Chapter 21, "Adding Insets to Your Graph," (p. 383)
Add data tips to your graph.	Chapter 24, "Adding Data Tips to Your Graph," (p. 473)
Add drill-down links to your graph.	Chapter 25, "Adding Drill-Down Links to Your Graph," (p. 477)
Create an attribute map for more control over your graph's visual attributes.	"Using Attribute Maps" (p. 536)

Task	Reference
Animate your graph.	Chapter 26, “Creating Animated Graphs,” (p. 483)
Enhance a predefined template.	Chapter 34, “Modifying Predefined Templates,” (p. 641)

4

Additional Resources to Help You Develop Your GTL Templates

Resources on the Web

Examples and Resources on the Web

The SAS website contains a large number of examples that can help you visualize and code your graphs. The examples cover a range of SAS technologies including the ODS Graphics procedures.

- Graphically Speaking is a blog focused on using ODS Graphics for data visualization in SAS. The blog covers topics related to the ODS Graphics procedures, the SAS Graph Template Language, and the SAS ODS Graphics Designer.

  https://blogs.sas.com/content/graphicallyspeaking/

- The SAS Training Post is a blog that provides tutorials, tips, and practical information about SAS. Dr. Robert Allison contributed a great deal of SAS/GRAPH and ODS Graphics data visualization examples to this blog during his tenure at SAS. His blog is no longer active, but it provides an extensive set of very useful examples.

  https://blogs.sas.com/content/sastraining/author/robertallison/

- The SAS Knowledge Base contains an abundance of searchable samples and SAS Notes. You can browse by topic, search for a particular note or a particular technology such as the name of a procedure, and conduct other searches.

https://support.sas.com/en/knowledge-base.html

Note: The SAS Knowledge Base content is currently available only in English.

- The Graphics Samples Output Gallery in the SAS Knowledge Base is a collection of graphs organized by SAS procedure. The graphs link to the source code. The gallery is maintained by SAS Technical Support.

  https://support.sas.com/en/knowledge-base/graph-samples-gallery.html

- The Focus Area Graphics site provides a simple interface to business and analytical graphs. The site is maintained by the SAS Data Visualization team.

  https://support.sas.com/rnd/datavisualization/index.htm

- You can share your questions, suggestions, and experiences related to graphics on the Graphics Programming community site.

  https://communities.sas.com/t5/Graphics-Programming/bd-p/sas_graph.

In addition, SAS offers instructor-led training and self-paced e-learning courses to help you get started with platform graphics software. The SAS Training site provides information about the courses that are available.

sas.com/training.

Books

The following books provide information about creating graphs using GTL.

- *Getting Started with the Graph Template Language in SAS®: Examples, Tips, and Techniques for Creating Custom Graphs*
- *Clinical Graphs Using SAS®*

For a list of other books that might help you, see “Recommended Reading” on page xviii.

PART 2

GTL Statements and Features

5

A Quick Look at GTL Statement Syntax

Statement Syntax

All GTL statements have the following syntax:

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

Here are some examples of GTL statements.

```
/* 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;
```

Statement-Block Syntax

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.

Here are some examples of GTL blocks.

```
/* 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.

6

Overview of the GTL Statements

GTL Statement Categories

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

GTL provides numerous plot statements that generate plots of various types. Use one or more plot statements in your template code to create the primary content for your graph. There are several categories of plot statements in GTL. For information

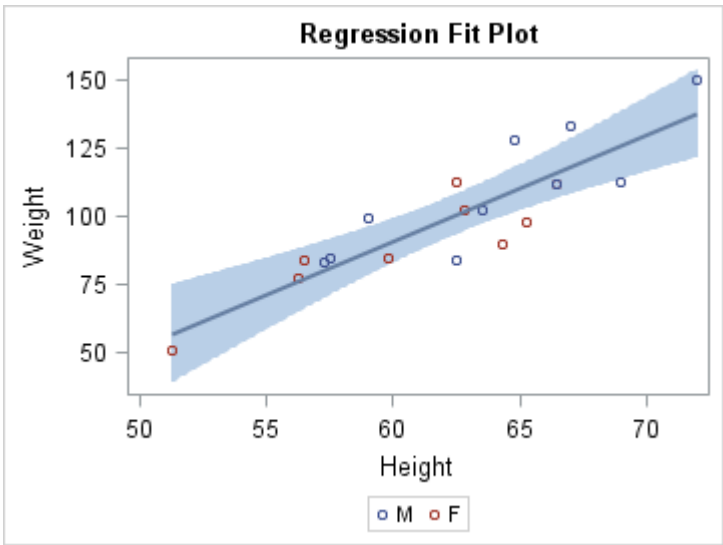
about the categories of the GTL plot statements and a list of the plot statements by category, see Chapter 8, “How the GTL Plot Statements Are Categorized,” on page 51 and Chapter 9, “Plot Statements Listed by Category,” on page 55. For a quick look at the plot types that are available in GTL and information about the plot statement that generates each, see Chapter 10, “Gallery of the GTL Plots,” on page 61.

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 or lines, or overlaid plots.
CONTINUOUSLEGEND	Name of an associated plot	Shows a numeric scale with a color ramp. Used in conjunction with contours, surfaces, and scatter plots.

The following figure shows a graph with a discrete legend.



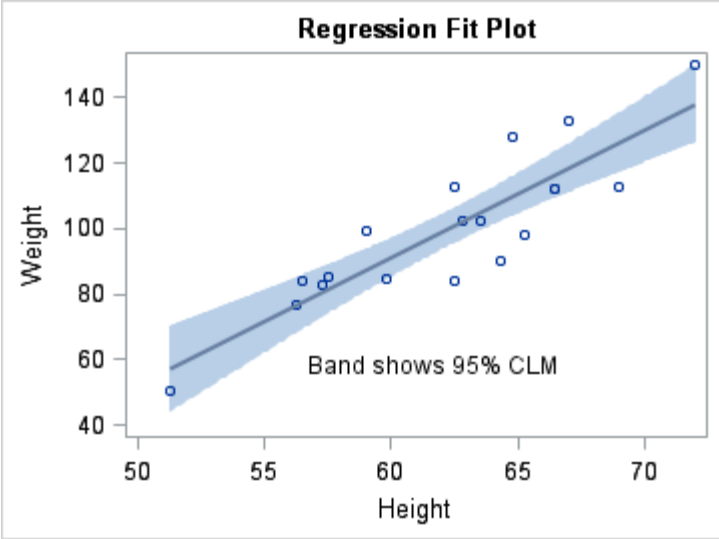
For more information, see Chapter 20, “Adding Legends to Your Graph,” on page 337.

Text Statements

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

Statement	Required Arguments	Comments
ENTRYTITLE	String	Text to appear above graph. Specify the ENTRYTITLE statement inside the BEGINGRAPH block as a child of the BEGINGRAPH block. It must not be embedded in any other GTL statement block.
ENTRYFOOTNOTE	String	Text to appear below graph. Specify the ENTRYFOOTNOTE statement inside the BEGINGRAPH block as a child of the BEGINGRAPH block. It must not be embedded in any other GTL statement block.
ENTRY	String	Text to appear within graph. Specify the ENTRY statement inside a layout block.

The following figure shows a graph with a text entry.

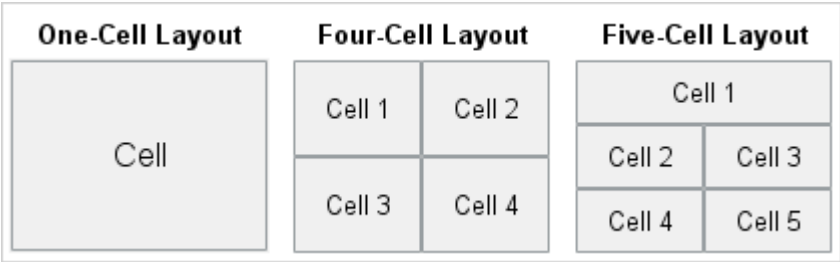


For more information, see Chapter 19, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 317.

Layout Statements

Layouts in GTL

GTL enables you to create any arrangement of graphs by defining layouts in your graph template. A layout is a container whose contents are arranged into cells. A cell can contain output from GTL statements, such as plots, text, and so on, or it can contain another layout. By using the layouts, you can arrange your graph as a single-cell or a multi-cell graph. Here are some examples.



Layout Types

GTL provides three primary types of layouts:

single-cell layouts
: consist of a one-cell container. The cell can contain a single plot, multiple overlaid plots, text, a legend, or another layout container. See "Single-Cell Layout Statements" on page 35.

multi-cell, non-data-driven layouts
: consist of a simple grid or of a panel that contains multiple cells. You define the contents of each cell individually. A cell can contain a single plot, text, a legend, or another layout container. See "Multi-Cell, Non-Data-Driven Layout Statements" on page 36.

multi-cell, data-driven layouts
: consist of a panel that contains a grid of cells. Each cell contains one graph. You define a graph prototype and identify one or more class variables. One graph (one cell) is created for each combination of class variables. See "Multi-Cell, Data-Driven Layout Statements" on page 36.

Single-Cell Layout Statements

GTL provides several statements that you can use to create single-cell layouts. The following table lists these statements.

Statement	Description
LAYOUT OVERLAY	Enables you to combine 2-D plots, titles, footnotes, legends, and text into a single cell. See Chapter 11, "Creating Overlay Graphs Using the OVERLAY Layout ," on page 97.
LAYOUT OVERLAY3D	Enables you to combine 3-D plots, titles, footnotes, legends, and text into a single cell. The BIHISTOGRAM3DPARM statement and the SURFACEPLOTPARM statement generate 3-D plots. See Chapter 13, "Creating Overlay 3-D Graphs Using the OVERLAY3D Layout ," on page 187.
LAYOUT OVERLAYEQUATED	Enables you to combine 2-D plots into a single graph with equated axes. That is, the units for the X and Y axes are always of equal size. Equated axes are necessary whenever distances between data points, or angles between vectors from the origin are meaningful. See Chapter 12, "Creating Overlay Graphs with Equated Axes Using the OVERLAYEQUATED Layout," on page 177.
LAYOUT REGION	enables you to combine 2-D plots with no axes, titles, footnotes, legends, and text into a single cell. The PIECHART statement and the MOSAICPLOTPARM statement generate plots that have no axes. See Chapter 17, "Creating

Statement	Description
	Graphs with No Axis Using the REGION Layout," on page 303.

Multi-Cell, Non-Data-Driven Layout Statements

GTL provides two statements that you can use to create multi-cell, non-data-driven layouts. The following table lists these statements.

Statement	Description
LAYOUT GRIDDED	Enables you to combine multiple 2-D and 3-D graphs into a grid when you do not need to align the graphs across cells or to scale the ranges of the data across the axes. The cells in LAYOUT GRIDDED are completely independent. You specify the content of each cell independently. See Chapter 14, "Creating Gridded Graphs Using the GRIDDED Layout ," on page 207.
LAYOUT LATTICE	Enables you to create panel of 2-D and 3-D graphs in which the data areas are automatically aligned, the axis data ranges are automatically scaled, and labels and headings are placed across the columns and rows. LAYOUT LATTICE is useful when you need to scale the data ranges across the columns and rows or to extract axis information from the cells to the outside of the grid. See Chapter 15, "Creating Lattice Graphs Using the LATTICE Layout," on page 221.

Multi-Cell, Data-Driven Layout Statements

GTL provides two layout statements that create a grid of graphs based on a graph prototype and one or more classification variables. A separate graph (cell) is created for each combination (crossing) of the specified classification variables. The primary differences between the two layout statements are the number of classification variables that you can specify and the order in which the individual crossings are added to the grid. The following table lists these layout statements.

Statement	Description

LAYOUT DATAPANEL	Enables you to specify one or more classification variables from which a panel of 2-D graphs is generated. You specify these variables in a list:
	`classvars=(var1, var2, var3, var4 ...)`
	`classvars=(country, product)`
	The order in which the individual crossings are added to the grid is based on the following criteria:
	■ the order in which you specify the classification variables. The last variable that you specify is the first variable to vary.
	■ whether you specify that cells should be populated by row or by column.
	■ the order that variable values occur in your data. You might want to sort the data by the classification variable.

LAYOUT DATALATTICE	Enables you to specify up to two classification variables from which a lattice of 2-D graphs is generated. You specify whether each variable is a row classifier or a column classifier.
	`rowvar=country`
	`columnvar=product`
	Variable values are always returned in the order of occurrence in the data. If you want to control the order in which individual graphs appear in the rows and columns, you can sort the input data by the classification variables.

See Chapter 16, “Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts,” on page 255.

Axis Management in GTL Layouts

For the GTL layouts that support axes, you can use options in the layout statement to manage the axes. The axes that you can manage depend on the layout type. The following table lists the layouts that support axes and how the axes are managed for each.

Layout Statement	How the Axes Are Managed
LAYOUT OVERLAY LAYOUT OVERLAY3D LAYOUT OVERLAYEQUATED	In OVERLAY layouts, by default, the outermost layout container automatically creates axes that are appropriate for the plots that it contains. Each layout statement provides options that you can use to modify various attributes of one or both axes.

Layout Statement	How the Axes Are Managed
	For information, see: ■ “Managing Axes in OVERLAY Layouts ” on page 103 ■ “Managing Axes in OVERLAYEQUATED Layouts” on page 180 ■ “Managing Axes in OVERLAY3D Layouts” on page 201
LAYOUT LATTICE	In LATTICE layouts, by default, the axes for each cell are determined by the outermost layout container that defines the content for each cell. To manage each cell’s internal axes, you specify axis options in the layout statement for that cell. The LATTICE layout provides options that enable you to replace each cell’s internal axes with external axes that are shared by each cell. One axis appears for each row (Y) and one axis appears for each column (Y). The LATTICE layout container automatically creates axes that are appropriate for the plots in each cell for that row or column. In many cases, the result is a cleaner graph. The LAYOUT LATTICE statement provides options that you can use to modify various attributes of the row and column axes. For more information, see “Managing Axes in LATTICE Layouts” on page 231.
LAYOUT DATAPANEL LAYOUT DATALATTICE	In DATAPANEL and DATALATTICE layouts, by default, the layout container creates axes for each row and column that are appropriate for the plots that are specified in the prototype. Each layout statement provides options that you can use to modify various attributes of one or both axes. For more information, see “Managing Axes in DATALATTICE and DATAPANEL Layouts” on page 268.

The REGION and GRIDDED layouts do not manage axes. The REGION layout supports only plot statements that do not require axes, such as PIECHART and MOSAICPLOTPARM. The GRIDDED layout does not support external axes. The axes must be managed on a per-cell basis.

7

Common Features Supported by GTL Statements

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.

Option	Description
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.

Option	Description
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.
DATASKIN=(*data-skin-name*)	Specifies a skin to be applied to filled areas of a plot. See "Using Data Skins" on page 554.

Plot Transparency

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

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

For more information, see "Using Transparency" on page 552.

Plot Identification

In GTL, legends require a reference (association) with a plot. The association is established by naming the plot, and then referring to the plot name in the legend statement.

Option	Description
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.

For more information about legends, see Chapter 20, “Adding Legends to Your Graph,” on page 337.

Another association exists between a model band plot and its dependent plot. In that case, a confidence name is specified in the MODELBAND statement, and then that confidence name is specified in a confidence option in the dependent plot statement. For more information about the MODELBAND statement, see “MODELBAND” in *SAS Graph Template Language: Reference*.

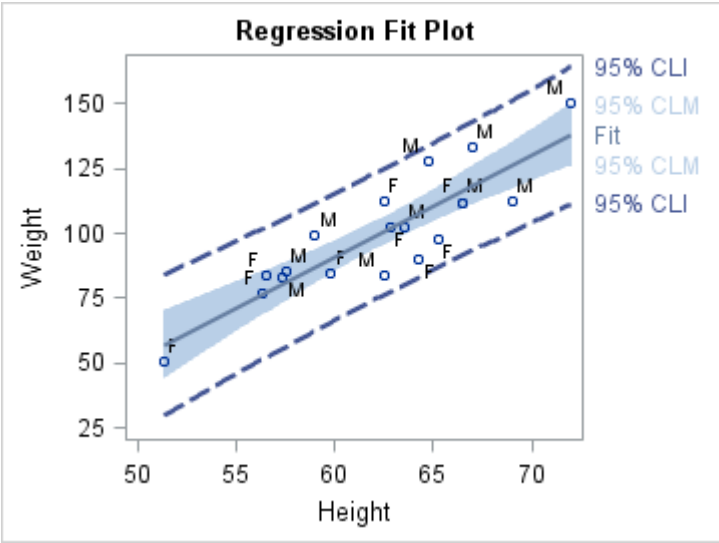
Labels for Plot Features

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

Option	Description
DATALABEL= *column*	Specifies a column to label data points in a scatter plot, series plot, needle plot, step plot, or vector plot.
DATALABELATTRS= *text-properties*	Specifies text properties for data labels.
DATALABELPOSITION=TOPRIGHT | TOP | TOPLEFT | LEFT | CENTER | RIGHT | BOTTOMLEFT | BOTTOM | BOTTOMRIGHT	Specifies the position of the data labels relative to the data points.
CURVELABEL= "*string*" | *column* | *expression*	Specifies a string, column, or expression to label one or more lines in a REFERENCELINE, DENSITYPLOT, LINEPARM, REGRESSIONPLOT,

Option	Description
	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. **Note:** The CURVELABELLOCATION=OUTSIDE option is not supported in the LATTICE, DATALATTICE, and DATAPANEL layouts.
CURVELABELPOSITION=	Specifies positioning options for the curve label(s).

The following figure shows a graph with data labels and curve labels.



For more information, see Chapter 19, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 317.

For bar, curve, and data labels, many plots support options that enable you to split long labels into multiple lines to avoid collisions. The following table lists these options.

Option	Description
BARLABELFITPOLICY=AUTO | NONE	Specifies a policy for avoiding bar label collisions in a vertical bar chart. When set to AUTO, the bar labels are rotated when a collision occurs.

Option	Description
DATALABELSPLIT=TRUE | FALSE	Specifies whether the data labels should be split into multiple lines, if necessary, to fit the label in the available space.
DATALABELSPLITCHAR="*character-list*"	Specifies one or more characters on which the data labels can be split.
DATALABELSPLITCHARDROP=TRUE | FALSE	Specifies whether the split characters on which a split occurs are removed from the displayed data label.
DATALABELSPLITJUSTIFY="*justification*"	Specifies how each line of the split data labels is to be justified.
CURVELABELSPLIT=TRUE | FALSE	Specifies whether the curve labels should be split into multiple lines, if necessary, to fit the label in the available space.
CURVELABELSPLITCHAR="*character-list*"	Specifies one or more characters on which the curve labels can be split.
CURVELABELSPLITCHARDROP=TRUE | FALSE	Specifies whether the split characters on which a split occurs are removed from the displayed curve label.
CURVELABELSPLITJUSTIFY="*justification*"	Specifies how each line of the split curve labels is to be justified.
VALUEFITPOLICY="*fit-policy*"	Specifies how block plot text values are adjusted to fit within the containing block.
VALUESPLITCHAR="*character-list*"	Specifies one or more characters on which the block labels can be split when value splitting is used in a block plot.
VALUESPLITCHARDROP=TRUE | FALSE	Specifies whether the split characters on which a split occurs are removed from the displayed block label when value splitting is used in a block plot.

For axis labels, options are available that enable you to split long labels into multiple lines to avoid truncation. The following table lists these options.

Option	Description
LABELFITPOLICY=AUTO | SPLIT | SPLITALWAYS	Specifies a fit policy for the axis label. If the original label is too long, AUTO specifies that the short label is used instead (if specified), and SPLIT specifies that the original label is split into multiple lines as needed to make it fit the available. SPLITALWAYS specifies that the original

Option	Description
	label is always split into multiple lines regardless of the available space.
LABELSPLITCHAR="*character-list*"	Specifies one or more characters on which the axis label can be split when the SPLIT or SPLITALWAYS fit policy is used.
LABELSPLITCHARDROP=TRUE | FALSE	Specifies whether the split characters on which a split occurs are removed from the displayed axis label when the SPLIT or SPLITALWAYS fit policy is used.
LABELSPLITJUSTIFY="*justification*"	Specifies how each line of the split axis label is to be justified when the SPLIT or SPLITALWAYS fit policy is used.

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. You can also use the GROUPDISPLAY= option to change how groups are displayed.

Option	Description
GROUP= *column*	Specifies a group variable, always treated as having discrete values. For an example use, see the example for "Legend Statements" on page 32.
INDEX= *positive-integer-column*	Specifies an integer column that associates each distinct data value with a predefined graphical style element GraphData1, GraphData2, …
GROUPDISPLAY=STACK | CLUSTER | OVERLAY	Specifies whether group values for each category are displayed as clusters of bars or as stacked bars for bar charts, or as a cluster of markers or an overlay of markers for other plot types.

For more information about using groups, see Chapter 27, "Managing Your Graph's Appearance," on page 491.

Axis Assignment

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

Option	Description
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 "Axis Management in GTL Layouts" on page 37.

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 an ODS HTML destination. You can customize the data tip information.

Option	Description
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 Chapter 24, "Adding Data Tips to Your Graph," on page 473.

Features Supported by Layout, Legend, and Text Statements

About Layout, Legend, and Text Statement Features

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

Backgrounds

Option	Description
OPAQUE= FALSE | TRUE	Specifies whether the background is transparent. By default, OPAQUE=FALSE
BACKGROUNDCOLOR= *color*	Specifies a color for the background when the background is opaque.

Borders

Option	Description
BORDER= FALSE | TRUE	Specifies whether a border is displayed. By default, BORDER=FALSE.
BORDERATTRS= (*line-options*)	Specifies the properties of the border when a border is displayed.

Padding

Option	Description
PAD= *number* PAD=(<TOP=*number*> <BOTTOM=*number*> <LEFT=*number*> <RIGHT=*number*>	Specifies 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.
OUTERPAD=AUTO | *number* OUTERPAD=(<TOP=*number*> <BOTTOM=*number*> <LEFT=*number*> <RIGHT=*number*>	Specifies whether extra space is added outside the border of layouts, legends, titles, and footnotes. The default is AUTO, which specifies that the default outside padding for the component is used.

Test Position

Option	Description
HALIGN= LEFT | CENTER | RIGHT VALIGN= TOP | CENTER | BOTTOM	Specifies the alignment for a text entry. For ENTRY statements, a position can be specified relative to the container. The vertical position of the title and footnote text is fixed, but the horizontal position can be adjusted.

Legend and Layout Positions

Option	Description
HALIGN= LEFT | CENTER | RIGHT | *value* VALIGN= TOP | CENTER | BOTTOM | *value*	Specifies the alignment for a legend or layout. For legends, and for layouts that are nested within an overlay type layout, a position can be specified relative to the parent container. The values CENTER, LEFT, RIGHT, TOP, and BOTTOM position the legend or layout at fixed locations horizontally or vertically. You can use *value* to express the position as a percentage of the available vertical or horizontal space using any

Option	Description
	numeric value between 0 and 1 inclusive. In order to use these options on a layout block, the layout block must be embedded in an overlay-type layout. Otherwise, the HALIGN= and VALIGN= options are ignored. In that case, a note is written to the SAS log. This restriction does not apply to the legend statements. Note the following equivalencies: ■ VALIGN=TOP is equivalent to VALIGN=1 ■ VALIGN=BOTTOM is equivalent to VALIGN=0 ■ HALIGN=RIGHT is equivalent to HALIGN=1 ■ HALIGN=LEFT is equivalent to HALIGN=0

PART 3

Selecting GTL Plot Statements

8

How the GTL Plot Statements Are Categorized

Plot Types

Overview of the Plot Types

In GTL, the following types are used to describe the plots:

- computed plot
- parameterized plot
- standalone plot
- dependent plot
- 2-D plot
- 3-D plot

The following sections describe these types.

Computed Plots

Computed plots internally perform computational transformations on the input data and add new columns to a data object as needed in order to render the requested plot. For example, a LOESSPLOT statement 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. Several options are available for most computed plots that enable you to control the computation that is 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. It is important to know whether a plot is computed because some layouts such as PROTOTYPE currently do not allow computed plots to be included.

Parameterized Plots

Parameterized plots simply render the input data that 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 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 that you want, and you have done the summarization yourself. A PARM suffix is added to the names of many parameterized plots. A parameterized plot is also useful 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. It is important to know whether a plot is parameterized because these plots ensure that no additional computation takes 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.

Standalone Plots

A standalone plot is one that can be drawn without any other accompanying plot. In general, a plot is standalone 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. It is important to know which plots are standalone because most layouts need to know the extents of the X and Y axis in order 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 only one point is 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 standalone plot, which provides the necessary information to determine the data extents of the two axes.

2-D Plots and 3-D Plots

GTL supports both 2-D graphics and 3-D graphics. Currently there are only two 3-D plot statements (SURFACEPLOTPARM and BIHISTOGRAM3DPARM). You must use 3-D plot statements in an OVERLAY3D layout. You cannot use 3-D plot statements in a 2-D layout. You cannot use 2-D plot statements in an OVERLAY3D layout. For more information about layouts, see “Layout Statements” on page 34.

Plot Categories

The GTL plot statement categories are based on the plot types. More than one type can apply to each category. The categories are as follows:

- 2-D standalone, computed plots
- 2-D standalone, parameterized plots
- 3-D standalone, parameterized plots
- dependent plots

Chapter 9, “Plot Statements Listed by Category,” on page 55 provides a list of the plot statements that are included each of these categories.

Additional Plot Category: the Primary Plot

For standalone plots, the plot can also be categorized as the primary plot. When you overlay two or more plots, the layout container determines the axes properties. The axes properties include the axis type for each axis, the data range of each axis, and the default format and label to use for each axis. By default, the first standalone plot statement that is encountered in the template definition is used to determine the axes properties. The axes are then shared with any remaining plots in the template. Because this plot determines the axes properties for the entire graph, it is categorized as the primary plot. Only a standalone plot can serve as the primary plot.

In some cases, you might need a particular overlay stacking order, which requires you to order your statements accordingly. In that case, the first standalone plot in the template definition might not produce the axis properties that you want. You can include the PRIMARY=TRUE option in any standalone plot statement in your template definition to designate that plot as the primary plot. That plot is then considered the primary plot regardless of statement order. Only one plot statement should have the PRIMARY=TRUE option. If more than one standalone plot statement has this option, the first statement encountered that has the PRIMARY=TRUE option is considered the primary plot.

9

Plot Statements Listed by Category

Standalone, 2-D, Computed Plots

Table 9.1 *Standalone, 2-D, Computed Plots*

Statement	Required Arguments	Comments
BARCHART	One *column*	Horizontal or vertical. For details, see "BARCHART" in *SAS Graph Template Language: Reference*.
BOXPLOT	One *numeric-column*	Horizontal or vertical. For details, see "BOXPLOT" in *SAS Graph Template Language: Reference*.
CONTOURPLOTPARM	Three *numeric-columns*	Draws contour plot from pre-gridded data. Basic "gridding" feature is provided using an option. For details, see "CONTOURPLOTPARM" in *SAS Graph Template Language: Reference*.
DENSITYPLOT	One *numeric-column*	Theoretical distribution curve (for example, NORMAL or KDE). For details, see

Statement	Required Arguments	Comments
		"DENSITYPLOT" in *SAS Graph Template Language: Reference*.
ELLIPSE	Two *numeric-columns*	Confidence or prediction ellipse for a set of points. For details, see "ELLIPSE" in *SAS Graph Template Language: Reference*.
HEATMAP	Two *columns*	Draws a map of tiles that are placed at each X= and Y= crossing and colored based on a third variable. **Note:** This statement is valid in SAS 9.4M3 and later releases. For details, see "HEATMAP" in *SAS Graph Template Language: Reference*.
HISTOGRAM	One *numeric-column*	Horizontal or vertical. For details, see "HISTOGRAM" in *SAS Graph Template Language: Reference*.
LINECHART	One *column*	Draws a chart that shows the relationship of one variable to another as trends. For details, see "LINECHART" in *SAS Graph Template Language: Reference*.
LOESSPLOT	Two *numeric-columns*	Fit plot using loess. For details, see "LOESSPLOT" in *SAS Graph Template Language: Reference*.
PBSPLINEPLOT	Two *numeric-columns*	Fit plot using Penalized B-spline. For details, see "PBSPLINEPLOT" in *SAS Graph Template Language: Reference*.
PIECHART	One *column*	Must be used within a LAYOUT REGION block. For details, see "PIECHART" in *SAS Graph Template Language: Reference*.
REGRESSIONPLOT	Two *numeric-columns*	Fit plot using linear, quadratic, or cubic regression. For details, see "REGRESSIONPLOT" in *SAS Graph Template Language: Reference*.
SCATTERPLOTMATRIX	Two or more *numeric-columns*	Grid of scatter plots. Might include computed ellipses, histograms, density curves. For details, see "SCATTERPLOTMATRIX" in *SAS Graph Template Language: Reference*.
WATERFALLCHART	Two columns. Y must be numeric	Waterfall chart consisting of bars that represent an initial value of Y and a series of intermediate bars that are identified by X and that lead to a final value of Y. For details, see

Statement	Required Arguments	Comments
		"WATERFALLCHART" in *SAS Graph Template Language: Reference*.

Standalone, 2-D, Parameterized Plots

Table 9.2 *Standalone, 2-D, Parameterized Plots*

Statement	Required Arguments	Comments
BANDPLOT	Three *columns*, at least two numeric limits	Area bounded by two straight or curved lines. The input data must be sorted by the X or Y variable. For details, see "BANDPLOT" in *SAS Graph Template Language: Reference*.
BARCHARTPARM	Two *columns*, Y must be numeric	Horizontal or vertical. Summarized data provided by user. For details, see "BARCHARTPARM" in *SAS Graph Template Language: Reference*.
BLOCKPLOT	Two *columns*	Strip of X- axis aligned rectangular blocks containing text. The X data must be sorted. For details, see "BLOCKPLOT" in *SAS Graph Template Language: Reference*.
BOXPLOTPARM	One *numeric-column* and one *string-column*	Horizontal or vertical. Needs special data format. For details, see "BOXPLOTPARM" in *SAS Graph Template Language: Reference*.
BUBBLEPLOT	Three *numeric-columns*	Plot of bubbles where a bubble is placed at each X= and Y= crossing and sized according to a response variable. By default, the bubbles appear as outlined circles. For details, see "BUBBLEPLOT" in *SAS Graph Template Language: Reference*.
DENDROGRAM	Three *numeric-columns*	Tree diagram that represents the results of a hierarchical clustering analysis. For details, see "DENDROGRAM" in *SAS Graph Template Language: Reference*.
ELLIPSEPARM	Five *numbers* or *numeric-columns*	Draws ellipse given center, slope, semi-major, and semi-minor axis lengths. For details, see "ELLIPSEPARM" in *SAS Graph Template Language: Reference*.

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. For details, see "FRINGEPLOT" in *SAS Graph Template Language: Reference*.
HEATMAPPARM	Two *columns* | *expressions* and *one numeric-column* | *expression*	Draws a map of tiles that are placed at each X= and Y= crossing and colored based on a response variable. For details, see "HEATMAPPARM" in *SAS Graph Template Language: Reference*.
HIGHLOWPLOT	Three columns. HIGH, and LOW must be numeric	Draws a high-low bar or line plot. For details, see "HIGHLOWPLOT" in *SAS Graph Template Language: Reference*.
HISTOGRAMPARM	Two *numeric-columns*	Horizontal or vertical. The Y data must be nonnegative. For details, see "HISTOGRAMPARM" in *SAS Graph Template Language: Reference*.
MOSAICPLOTPARM	List of categorical *columns* enclosed in parenthesis and a *numeric-column*.	Creates a mosaic plot from pre-summarized categorical data. The numeric column must contain only positive values. For details, see "MOSAICPLOTPARM" in *SAS Graph Template Language: Reference*.
NEEDLEPLOT	Two *columns*, Y must be numeric	Draws parallel, vertical line segments connecting data points to a baseline. For details, see "NEEDLEPLOT" in *SAS Graph Template Language: Reference*.
POLYGONPLOT	Three *columns*	Draws one or more polygons from data that is stored in a data set. **Note:** This statement is valid in SAS 9.4M1 and later releases. For details, see "POLYGONPLOT" in *SAS Graph Template Language: Reference*.
SCATTERPLOT	Two *columns*	Draws markers at data point locations. The markers can be sized according to the response variable by using one or more options. For details, see "SCATTERPLOT" in *SAS Graph Template Language: Reference*.
SERIESPLOT	Two *columns*	Draws line segments to connect a set of data points. For details, see "SERIESPLOT" in *SAS Graph Template Language: Reference*.

Statement	Required Arguments	Comments
STEPPLOT	Two *columns*, Y must be numeric	Draws stepped line segments to connect a set of data points. For details, see "STEPPLOT" in *SAS Graph Template Language: Reference*.
TEXTPLOT	Three *columns*	Draws text markers at data point locations. **Note:** This statement is valid in SAS 9.4M2 and later releases. The marker text is provided by a column in the plot data. For details, see "TEXTPLOT" in *SAS Graph Template Language: Reference*.
VECTORPLOT	Four *numeric-columns*, X and Y origins can be numeric constants.	Creates directed line segment(s) based on pairs of data points. For details, see "VECTORPLOT" in *SAS Graph Template Language: Reference*.

Standalone, 3-D, Parameterized Plots

Table 9.3 *Standalone, 3-D, Parameterized Plots*

Statement	Required Arguments	Comments
BIHISTOGRAM3DPARM	Three *numeric-columns*	Bivariate histogram. The Z data must be nonnegative. For details, see "BIHISTOGRAM3DPARM" in *SAS Graph Template Language: Reference*.
SURFACEPLOTPARM	Three *numeric-columns*	Smooth surface. For details, see "SURFACEPLOTPARM" in *SAS Graph Template Language: Reference*.

Dependent Plots

Table 9.4 Dependent Plots

Statement	Required Arguments	Comments
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. For details, see "DROPLINE" in *SAS Graph Template Language: Reference*.
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. For details, see "LINEPARM" in *SAS Graph Template Language: Reference*.
MODELBAND	CLM or CLI name of associated fit plot	Confidence bands. Used only in conjunction with a fit plot. For details, see "MODELBAND" in *SAS Graph Template Language: Reference*.
REFERENCELINE	X or Y location, *column*	Draws line(s) perpendicular to an axis. For details, see "REFERENCELINE" in *SAS Graph Template Language: Reference*.

10

Gallery of the GTL Plots

A Quick Look at the Gallery

GTL provides a wide variety of plots. This section lists the basic plots that are available with GTL and provides a sample of each. The statement that generates each plot or chart is also listed. For additional information, see "Plot Statements" on page 31.

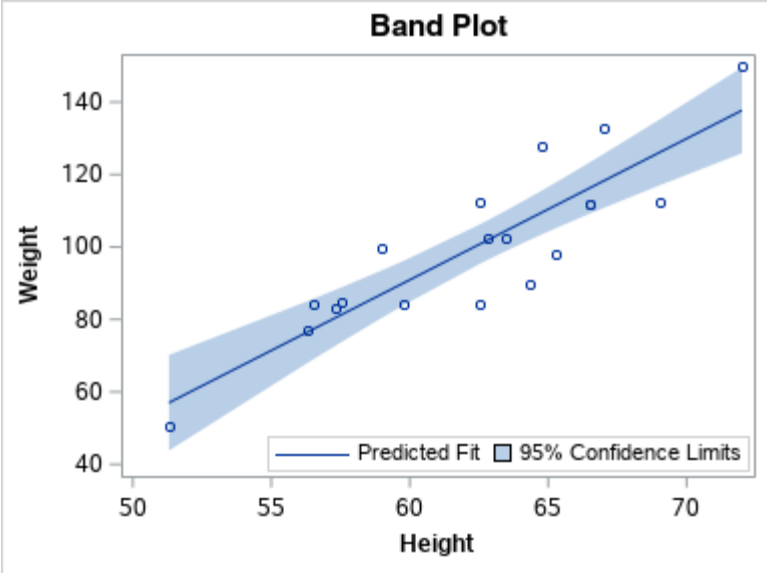
This gallery contains a sample of each of the following chart and plot types:

Band Plots	Heat Maps	Regression Plots
Bar Charts	High-Low Plots	Scatter Plots
Bar-Line Charts	Histograms (Univariate)	Series Plots
Block Plots	Histograms (Bivariate)	Spline Plots
Box Plots	Line Charts	Step Plots
Bubble Plots	Loess Plots	Straight-Line Plot
Contour Plots	Model Band Plots	Surface Plots
Dendrograms	Mosaic Plots	Text Plots
Density Plots	Needle Plots	Vector Plots
Dot Plots	Penalized B-Spline Plots	Waterfall Charts
Ellipse Plots	Pie Charts	
Fringe Plots	Polygon Plots	

Gallery of Basic Single-Cell Plots

Band Plots

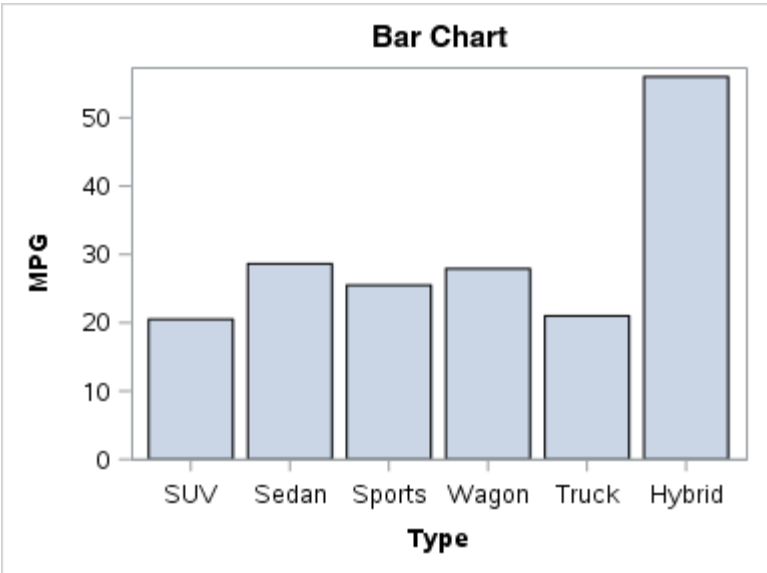
A band plot is typically used to display confidence or prediction limits as a band along the X or Y axis. The plot data provides the X or Y values and the high and low values that are required to draw the band. Here is a sample.



Use the BANDPLOT statement in GTL to generate a band plot. The BANDPLOT statement syntax is described in "BANDPLOT" in *SAS Graph Template Language: Reference*. For an example, see "BANDPLOT Statement" in *SAS Graph Template Language: Reference*.

Bar Charts

Bar charts use vertical or horizontal bars to represent statistics based on the values of a category variable. You can also provide a response variable. Here is a sample.



Use the BARCHART statement or the BARCHARTPARM statement in GTL to generate a bar chart. The BARCHART statement draws a bar chart from raw data. It computes all of the values that are required to draw the chart. The BARCHART statement syntax is described in "BARCHART" in *SAS Graph Template Language: Reference*.

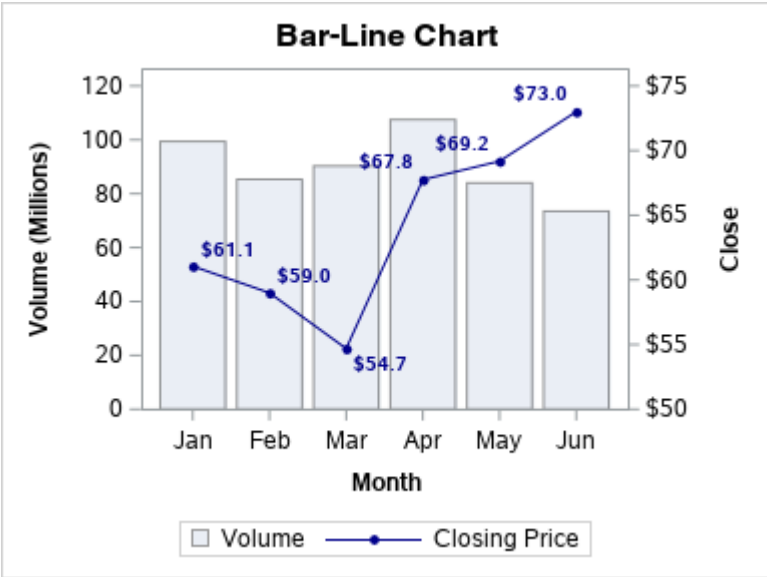
For examples, see the following:

- "Horizontal Bar Chart" in *SAS Graph Template Language: Reference*
- "Grouped Vertical Bar Chart" in *SAS Graph Template Language: Reference*
- "Interval Bar Chart" in *SAS Graph Template Language: Reference*
- "Bar Chart with Bar Colors Controlled by a Statistic" in *SAS Graph Template Language: Reference*

The BARCHARTPARM statement draws a bar chart from pre-summarized data. It provides no data transformation. The BARCHARTPARM statement syntax is described in "BARCHARTPARM" in *SAS Graph Template Language: Reference*. For an example, see "BARCHARTPARM Statement" in *SAS Graph Template Language: Reference*.

Bar-Line Charts

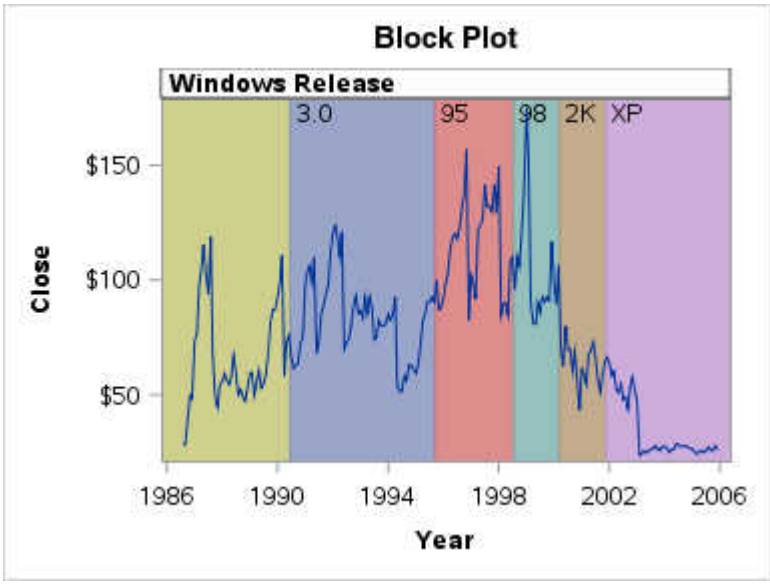
A bar-line chart is a vertical bar chart with one or more plot overlays. This chart graphically represents the value of a statistic calculated for one or more variables in an input SAS data set. Here is a sample.



Use the BARCHART and LINECHART statements in GTL to generate a vertical bar-line chart or a horizontal bar-line chart. Place the LINECHART statement after the BARCHART statement in your template code. The BARCHART statement syntax is described in "BARCHART" in *SAS Graph Template Language: Reference*, and the LINECHART statement syntax is described in "LINECHART" in *SAS Graph Template Language: Reference*. For an example, see "Bar-Line Chart" in *SAS Graph Template Language: Reference*.

Block Plots

A block plot consists of rectangular strips at specific intervals along the X axis. The intervals are provided in the plot data. Each block can display the X-axis boundary value for that block. The block can also display an alternate text value for that X-axis value that is also provided in the plot data. Here is a sample.

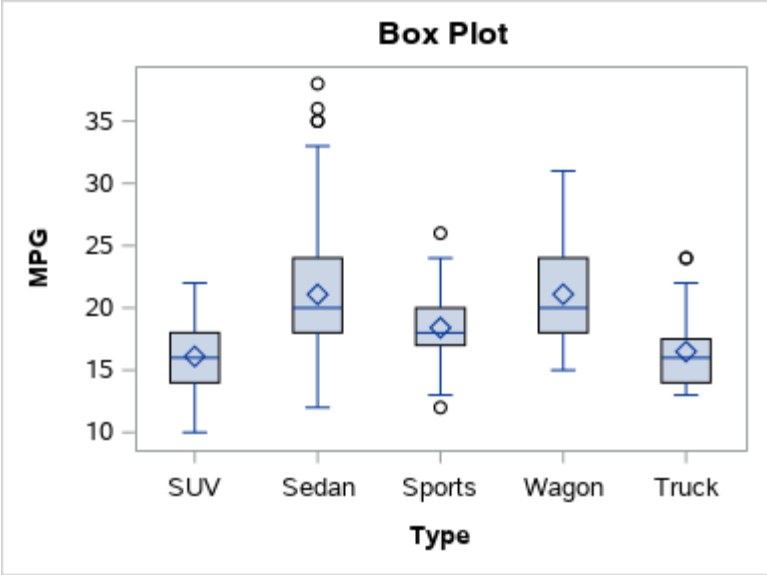


Use the BLOCKPLOT statement in GTL to generate a block plot.The BLOCKPLOT statement syntax is described in "BLOCKPLOT" in *SAS Graph Template Language: Reference*. For examples, see the following:

- "BlockPlot Overlaid with SeriesPlot" in *SAS Graph Template Language: Reference*
- "Standalone BlockPlot in Lattice Layout" in *SAS Graph Template Language: Reference*

Box Plots

A box plot summarizes the data and indicates the median, upper and lower quartiles, and minimum and maximum values. The plot provides a quick visual summary that easily shows center, spread, range, and any outliers. Here is a sample.



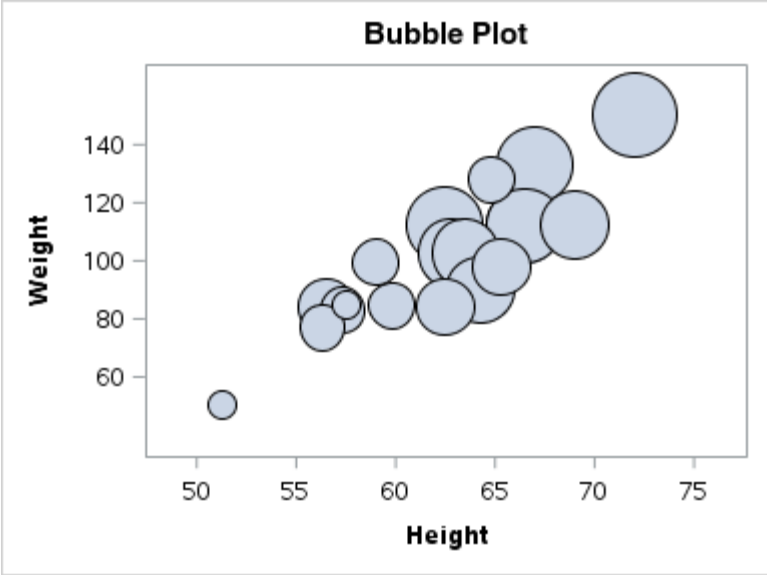
Use the BOXPLOT or BOXPLOTPARM statement to generate a box plot in GTL. These statements can draw a vertical or horizontal schematic (Tukey) or skeletal box plot representation. Labeled outliers can also be drawn. The BOXPLOT statement draws a box plot from raw input data. It computes all of the values needed to draw the plot. The BOXPLOT statement syntax is described in "BOXPLOT" in *SAS Graph Template Language: Reference*. For examples, see the following:

- "BOXPLOT" in *SAS Graph Template Language: Reference*
- "Box Plot of City MPG and Vehicle Type Grouped by Origin" in *SAS Graph Template Language: Reference*

The BOXPLOTPARM statement draws a box plot from pre-computed data. It provides no data transformation. The BOXPLOTPARM statement syntax is described in "BOXPLOTPARM" in *SAS Graph Template Language: Reference*. For an example, see "BOXPLOTPARM Statement" in *SAS Graph Template Language: Reference*.

Bubble Plots

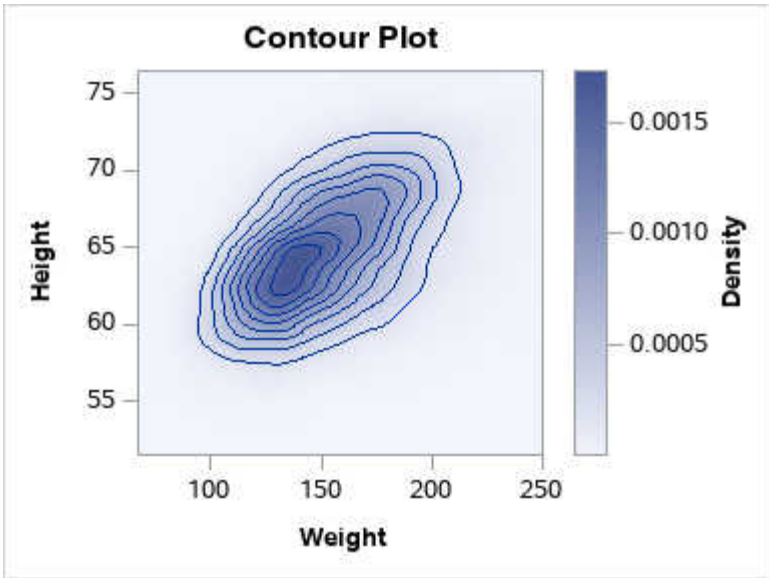
A bubble plot consists of X and Y values, which specify the location of the center of each bubble, and the value of a third variable that determines the size of the bubble. Here is a sample.



Use the BUBBLEPLOT statement to generate a bubble plot in GTL.The BUBBLEPLOT statement syntax is described in "BUBBLEPLOT" in *SAS Graph Template Language: Reference*. For an example, see "BUBBLEPLOT Statement" in *SAS Graph Template Language: Reference*.

Contour Plots

A contour plot represents a three-dimensional surface by plotting Z values on a two-dimensional format. The data provides the X, Y, and Z values. Here is a sample.

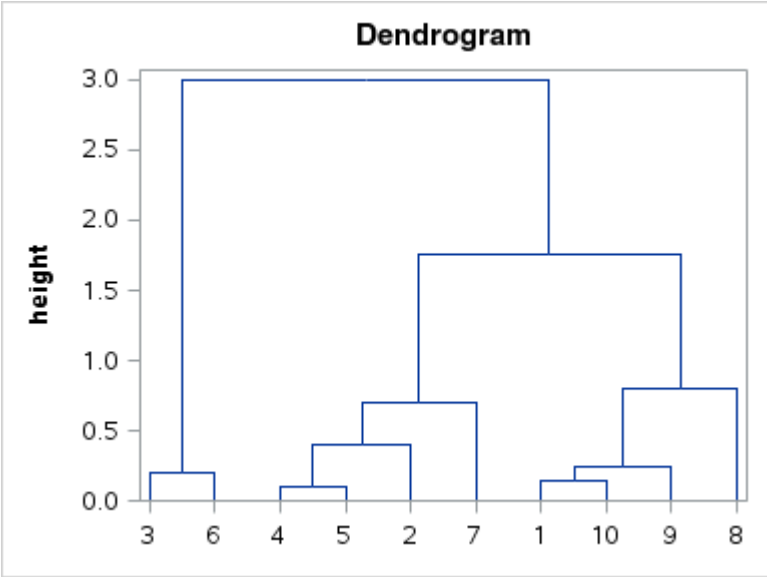


Use the CONTOURPLOTPARM statement to generate a contour plot in GTL. The plot data provides the X, Y, and Z values. The CONTOURPLOTPARM statement computes all of the values needed to draw the plot. You can choose from a variety of contour types for your plot. This example shows the FILL contour type. The CONTOURPLOTPARM statement syntax is described in "CONTOURPLOTPARM"

in *SAS Graph Template Language: Reference*. For an example, see "CONTOURPLOTPARM Statement" in *SAS Graph Template Language: Reference*.

Dendrograms

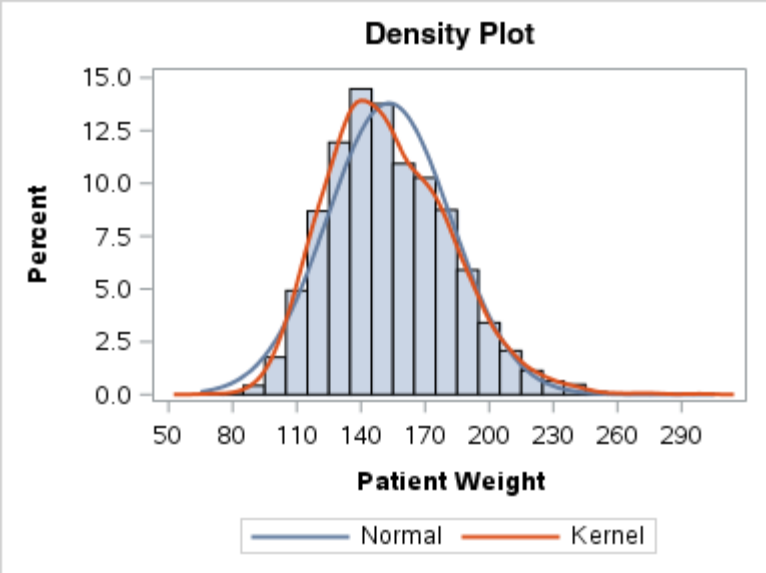
A dendrogram is a tree diagram that represents the results of a hierarchical clustering analysis. Here is a sample.



Use the DENDROGRAM statement to generate a dendrogram in GTL. Typically, a DENDROGRAM statement is used by itself in an OVERLAY layout. Reference lines and band plots can be included in the layout with the DENDROGRAM statement. However, including other plot statements in the layout can produce unexpected results. The DENDROGRAM statement syntax is described in "DENDROGRAM" in *SAS Graph Template Language: Reference*. For an example, see "DENDROGRAM Statement" in *SAS Graph Template Language: Reference*.

Density Plots

A density plot creates a density curve that shows the distribution of values for a numeric variable. The most common density plot uses the normal distribution, which is defined by the mean and the standard deviation. Here is a sample.

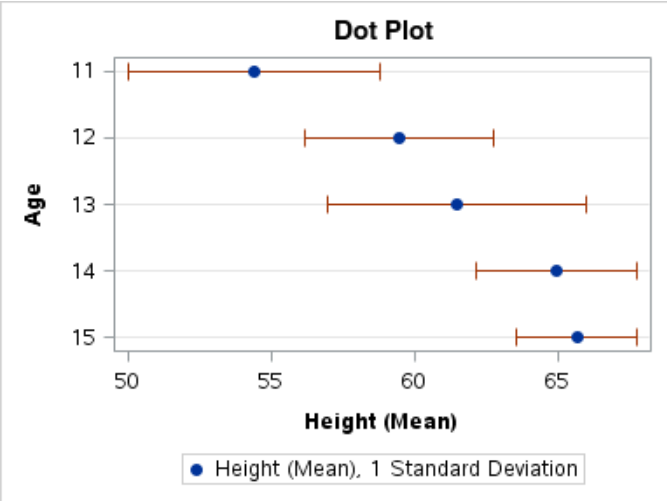


Use the DENSITYPLOT statement to generate a univariate probability density curve in GTL. You can specify a NORMAL or KERNEL distribution. The DENSITYPLOT statement syntax is described in "DENSITYPLOT" in *SAS Graph Template Language: Reference*. For examples, see the following:

- "Overlaid Density Plots" in *SAS Graph Template Language: Reference*
- "Density Plot and Histogram" in *SAS Graph Template Language: Reference*

Dot Plots

A dot plot displays horizontally the values of a category variable, which is typically a statistic such as frequency or mean. Each category value is represented by a dot. The plot can also display statistical limits for each value. Here is a sample.

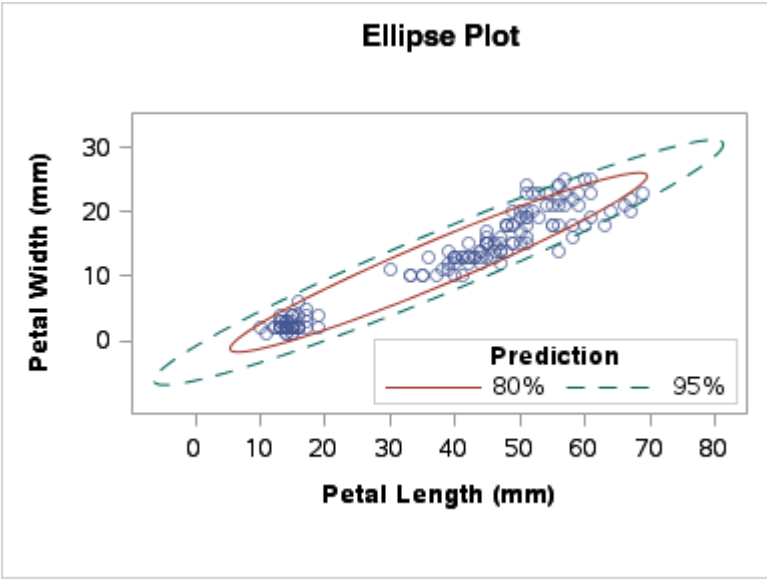


Use the SUMMARY procedure to compute the desired statistic for your data. Use the SCATTERPLOT statement with the MARKERATTRS= option to create the dot plot. If you want to display statistical limits, add the XUPPERLIMIT= and

XLOWERLIMIT= options to your SCATTERPLOT statement. The SUMMARY procedure syntax is described in "SUMMARY Procedure" in *Base SAS Procedures Guide*. The SCATTERPLOT statement syntax is described in "SCATTERPLOT" in *SAS Graph Template Language: Reference*. For an example, see "Dot Plot" in *SAS Graph Template Language: Reference*.

Ellipse Plots

An ellipse plot creates a confidence or prediction elliptical curve computed from input data. In order to produce useful output, the ellipse should be used with another plot statement that uses numeric axes. Here is a sample.



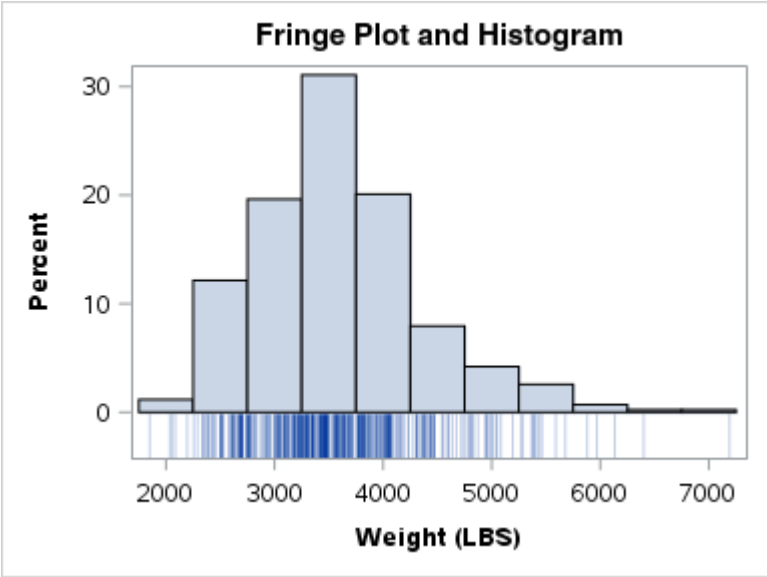
Use the ELLIPSE or ELLIPSEPARM statement to generate an ellipse plot in GTL. The ELLIPSE statement draws the ellipses from the plot data. It computes all of the values needed to draw the ellipses. The ELLIPSE statement syntax is described in "ELLIPSE" in *SAS Graph Template Language: Reference*. For examples, see the following:

- "Prediction Ellipses" in *SAS Graph Template Language: Reference*
- "95% Confidence Ellipses by Plant Species" in *SAS Graph Template Language: Reference*

The ELLIPSEPARM statement draws confidence ellipses from data that is computed by another plot. A confidence ellipse must be overlaid with a plot that is derived from the same data. Typically, it is overlaid with a scatter plot. The ELLIPSEPARM statement syntax is described in "ELLIPSEPARM" in *SAS Graph Template Language: Reference*. For an example, see "ELLIPSEPARM Statement" in *SAS Graph Template Language: Reference*.

Fringe Plots

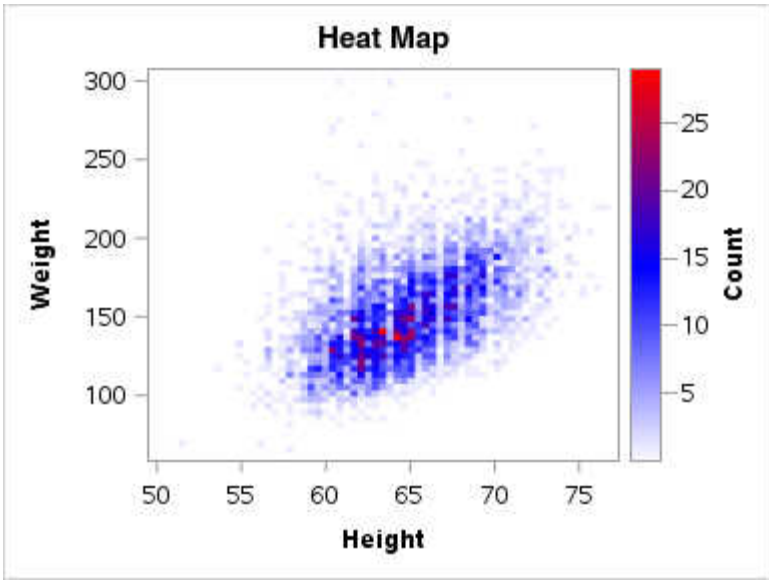
A fringe plot displays data values as a fringe on the X or X2 axis of the plot and often is used to display each observation in the data. All of the fringe lines are of equal length. Here is a sample.



Use the FRINGEPLOT statement to generate a fringe plot in GTL.The FRINGEPLOT statement syntax is described in "FRINGEPLOT" in *SAS Graph Template Language: Reference*. For an example, see "FRINGEPLOT Statement" in *SAS Graph Template Language: Reference*.

Heat Maps

A heat map is a grid of rectangles in which the location of each rectangle is determined by two independent variables. You can specify a third variable to determine the color of each rectangle. Here is a sample.



Use the HEATMAPPARM statement or, starting with SAS 9.4M3, the HEATMAP statement to generate a heat map in GTL. The HEATMAP statement draws a heat map from raw input data. It computes all of the values needed to draw the heat map. The HEATMAP statement syntax is described in "HEATMAPPARM" in *SAS Graph Template Language: Reference*. For an example, see "HEATMAP Statement" in *SAS Graph Template Language: Reference*.

The HEATMAPPARM statement draws a heat map from pre-computed data. The HEATMAPPARM statement syntax is described in "HEATMAP" in *SAS Graph Template Language: Reference*. For an example, see "HEATMAPPARM Statement" in *SAS Graph Template Language: Reference*.

High-Low Plots

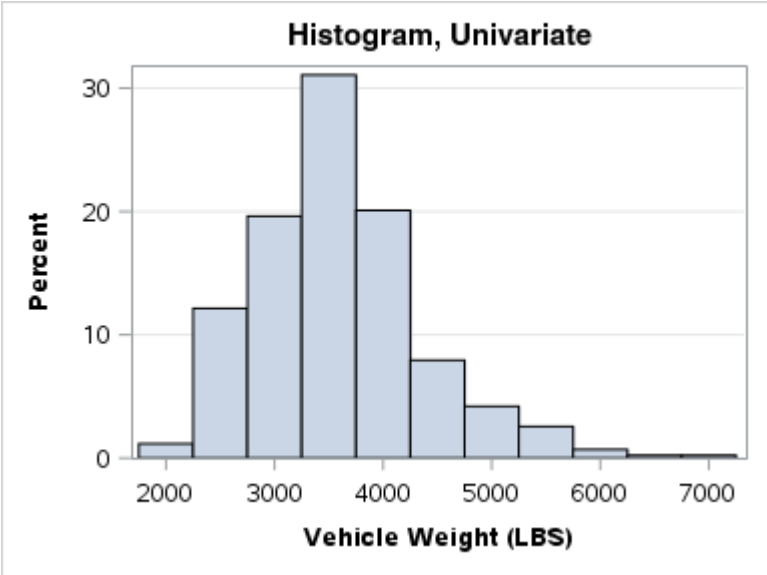
A high-low plot creates a display of floating vertical or horizontal lines or bars that represent high and low values for each value of a variable. Here is a sample.

Use the HIGHLOWPLOT statement to generate a high-low plot in GTL. You can choose lines or bars, and you can select from a variety of cap shapes for the ends of the high and low lines or bars. You can also display labels and arrowheads for the high and low values and tick marks for a specific variable value. The HIGHLOWPLOT statement syntax is described in "HIGHLOWPLOT" in *SAS Graph Template Language: Reference*. For examples, see the following:

- "Vertical High-low Chart" in *SAS Graph Template Language: Reference*
- "Horizontal High-low Chart" in *SAS Graph Template Language: Reference*

Histograms (Univariate)

A univariate histogram consists of a series of bins representing the frequency of a variable over a discrete interval or class. You can also specify a group variable. Here is a sample.



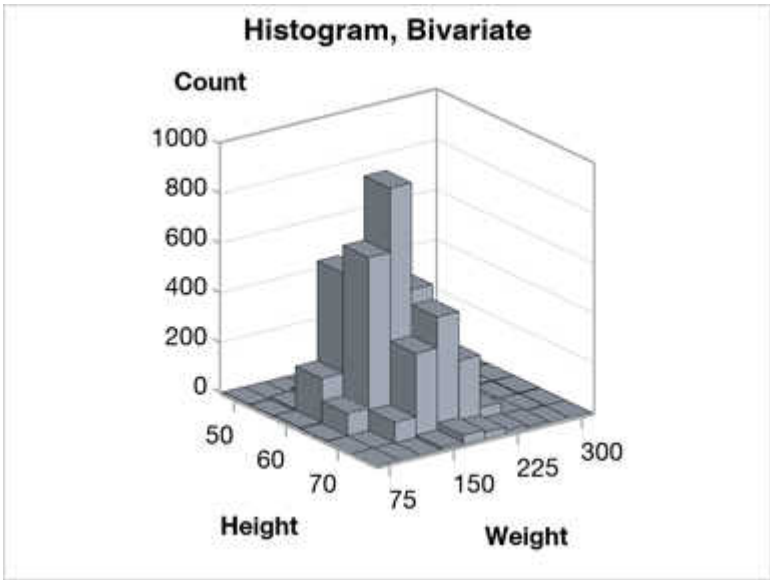
Use the HISTOGRAM or HISTOGRAMPARM statement to generate a histogram in GTL. The HISTOGRAM statement draws a histogram from raw input data. It computes all of the values needed to draw the histogram. The HISTOGRAM statement syntax is described in "HISTOGRAM" in *SAS Graph Template Language: Reference*. For examples, see the following:

- "Histogram of Vehicle Weight" in *SAS Graph Template Language: Reference*
- "Histogram of Highway Mileage Grouped by Origin" in *SAS Graph Template Language: Reference*

The HISTOGRAMPARM statement draws a histogram from pre-computed data. It provides no data transformation. The HISTOGRAMPARM statement syntax is described in "HISTOGRAMPARM" in *SAS Graph Template Language: Reference*. For an example, see "HISTOGRAMPARM Statement" in *SAS Graph Template Language: Reference*.

Histograms (Bivariate)

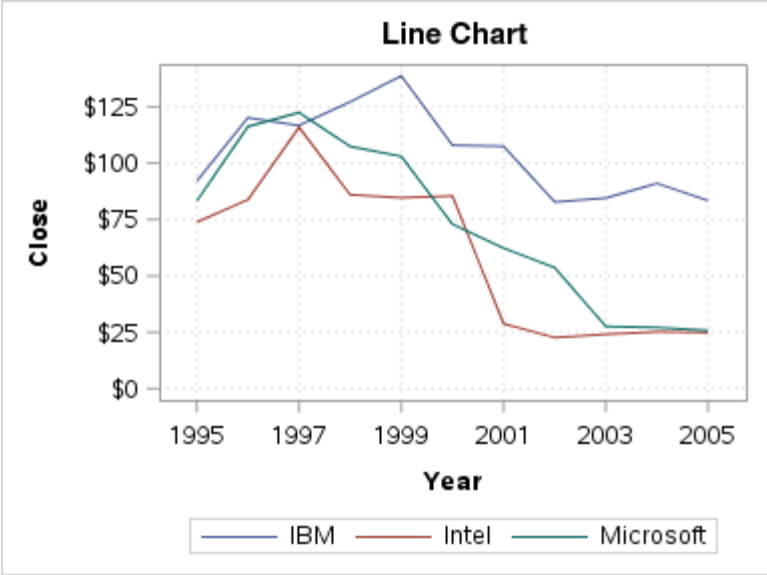
A 3-D histogram is a three-dimensional bivariate histogram of three variables X, Y, and Z, where the values of X and Y have already been gridded. The Z variable represents a response value for the frequency, percentage counts, or densities of each bin combination. Here is a sample.



Use the BIHISTOGRAM3DPARM statement to generate a bivariate histogram in GTL. The plot data must be binned by both X and Y, and the Z variable values must not be negative. The BIHISTOGRAMPARM statement syntax is described in "BIHISTOGRAM3DPARM" in *SAS Graph Template Language: Reference*. For an example, see "BIHISTOGRAM3DPARM Statement" in *SAS Graph Template Language: Reference*.

Line Charts

A line chart consists of straight-line segments that connect consecutive data points along an axis. You can use the line chart with a bar chart to create a horizontal bar-line chart. Line charts can be horizontal or vertical. The area below each line can be filled. Here is a sample.

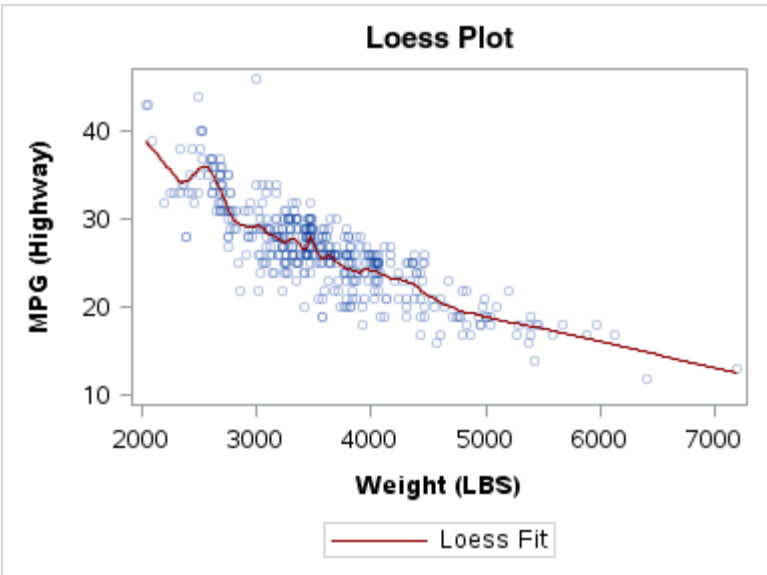


Use the LINECHART statement to generate a line chart in GTL. The LINECHART statement syntax is described in "LINECHART" in *SAS Graph Template Language: Reference*. For examples, see

- "Grouped Line Chart with Custom Line and Fill Attributes" in *SAS Graph Template Language: Reference*
- "Grouped Line Chart with Discrete Attribute Map" in *SAS Graph Template Language: Reference*

Loess Plots

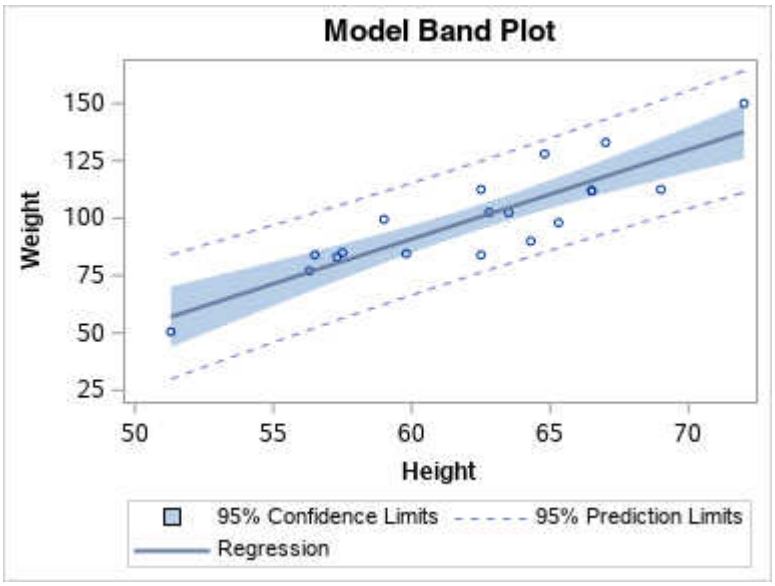
A loess plot is a nonlinear fit line that enables you to perform locally weighted polynomial regression. You can include a scatter plot of two numeric variables, and you can include confidence limits. Here is a sample.



Use the LOESSPLOT statement to generate a loess plot in GTL. The LOESSPLOT statement supports only models of one independent and one dependent variable. The LOESSPLOT statement syntax is described "LOESSPLOT" in *SAS Graph Template Language: Reference*. For an example, see "LOESSPLOT Statement" in *SAS Graph Template Language: Reference*.

Model Band Plots

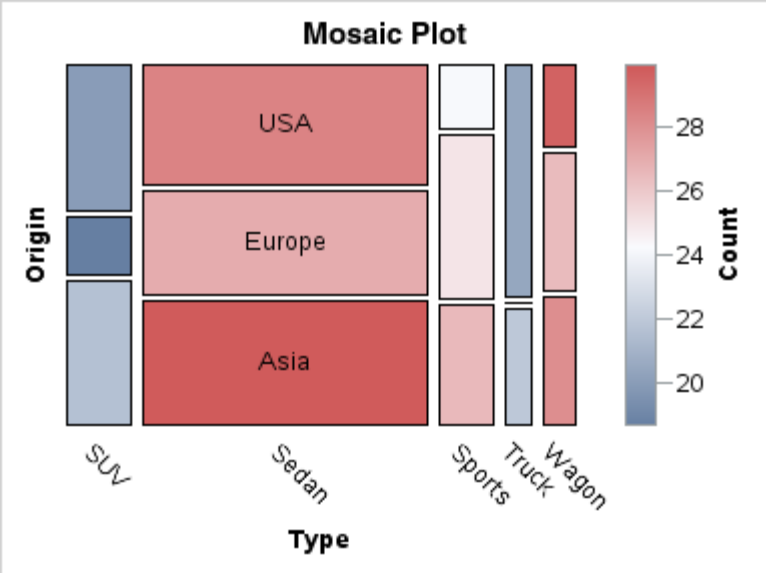
A model band shows the confidence limits for an associated smoother plot such as a regression plot. The model band must be associated with a smoother statement that specifies a fitted model and a type of confidence level to compute. Here is a sample.



Use the MODELBAND statement to generate a model band plot in GTL. The MODELBAND statement must be associated with a smoother statement that specifies a fitted model and a type of confidence level to compute. The MODELBAND statement syntax is described in "MODELBAND" in *SAS Graph Template Language: Reference*. For an example, see "MODELBAND Statement" in *SAS Graph Template Language: Reference*.

Mosaic Plots

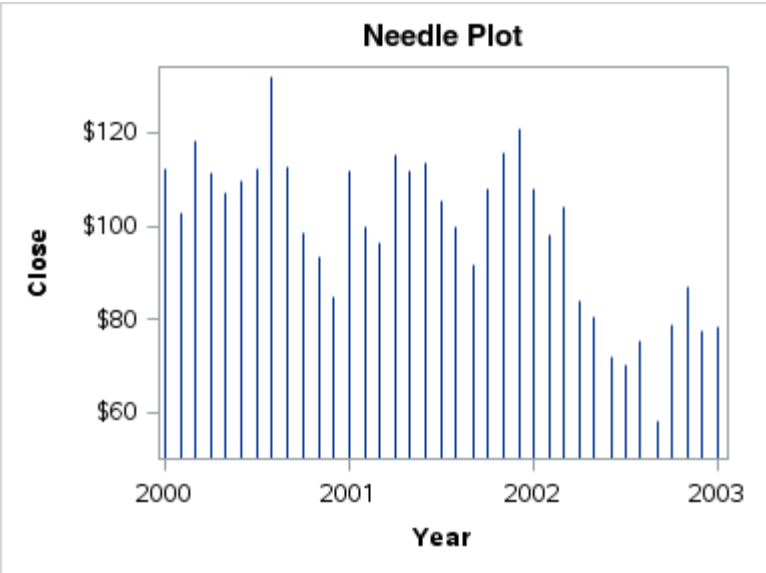
A mosaic plot displays relative frequencies for the categorical variables. Each crossing of the categorical values is represented by a tile. The area of each tile is proportional to the frequency of that crossing. The tile colors can be mapped to a numeric variable. Here is a sample.



Use the MOSAICPLOTPARM statement from pre-computed categorical data to generate a mosaic plot in GTL. The MOSAICPLOTPARM statement must be placed in a REGION, GRIDDED, or LATTICE layout. The MOSAICPLOTPARM statement syntax is described in "MOSAICPLOTPARM" in *SAS Graph Template Language: Reference*. For an example, see "MOSAICPLOTPARM Statement" in *SAS Graph Template Language: Reference*.

Needle Plots

A needle plot uses vertical line segments, or needles, to connect each data point to a baseline. Here is a sample.

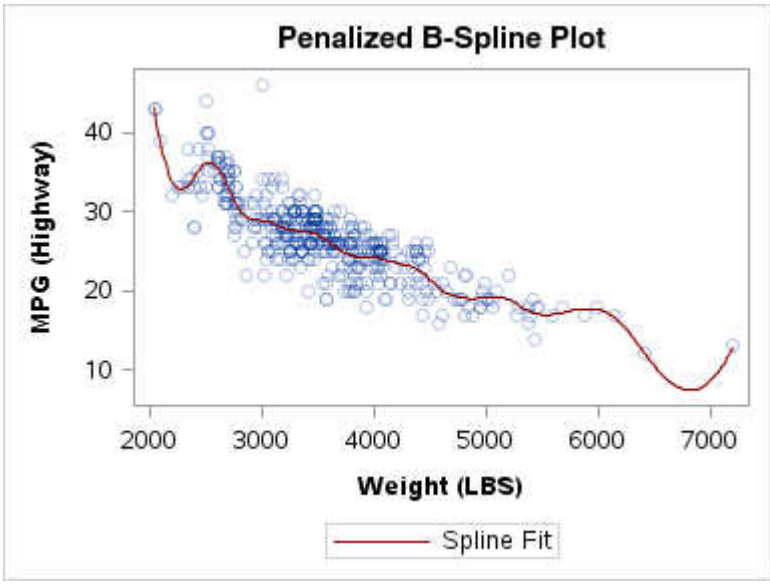


Use the NEEDLEPLOT statement to generate a needle plot in GTL. The NEEDLEPLOT statement syntax is described in "NEEDLEPLOT" in *SAS Graph*

Template Language: Reference. For an example, see "NEEDLEPLOT Statement" in *SAS Graph Template Language: Reference*.

Penalized B-Spline Plots

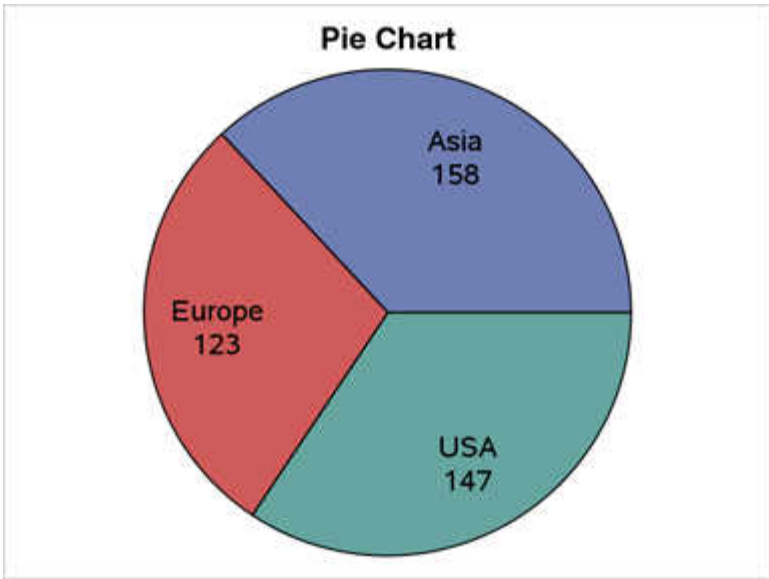
A penalized B-spline plot supports models of one independent and one dependent variable. The plot can compute confidence levels for the fit line. Here is a sample.



Use the PBSPLINEPLOT statement to generate a penalized B-spline plot in GTL. The PBSPLINEPLOT statement supports only models of one independent and one dependent variable. In addition to the penalized B-spline plot, the PBSPLINEPLOT statement can compute confidence levels for the fit line. The PBSPLINEPLOT statement syntax is described in "PBSPLINEPLOT" in *SAS Graph Template Language: Reference*. For an example, see "PBSPLINEPLOT Statement" in *SAS Graph Template Language: Reference*.

Pie Charts

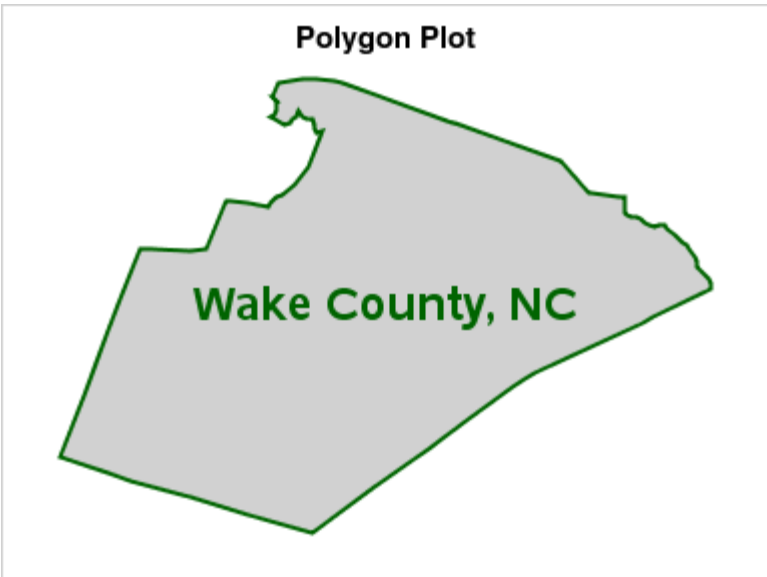
A pie chart represents input data as slices on the pie. Here is a sample.



Use the PIECHART statement to generate a pie chart in GTL. The PIECHART statement must be placed in a REGION, GRIDDED, or LATTICE layout. You can select the information that you want to display in each pie slice. The PIECHART statement syntax is described in "PIECHART" in *SAS Graph Template Language: Reference*. For an example, see "PIECHART Statement" in *SAS Graph Template Language: Reference*.

Polygon Plots

A polygon plot contains one or more polygons. The polygons can be filled, outlined, or both. They can also be labeled. The polygon plot is useful for drawing shapes on your graphs that highlight data, outline boundaries, and so on. Here is a sample that draws an outline of Wake County in North Carolina.



Use the POLYGONPLOT statement to generate s polygon plot in GTL.

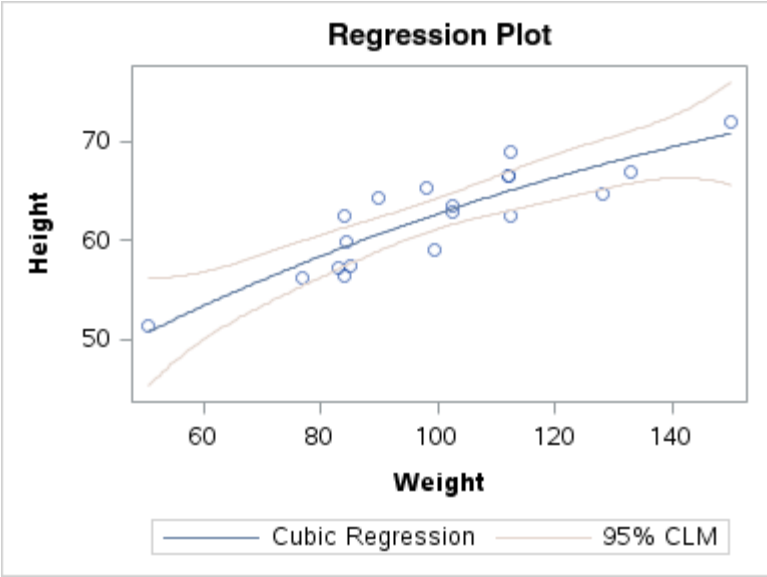
Note: This feature applies to SAS 9.4M1 and to later releases.

The POLYGONPLOT statement syntax is described in "POLYGONPLOT" in *SAS Graph Template Language: Reference*. For examples, see the following:

- "Drawing a Simple Polygon That Highlights Data" in *SAS Graph Template Language: Reference*
- "Drawing a Geographical Boundary" in *SAS Graph Template Language: Reference*

Regression Plots

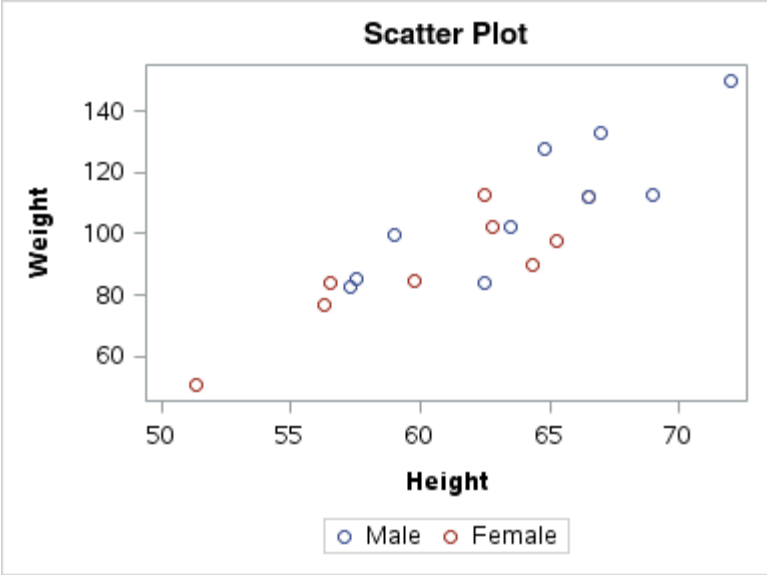
A regression plot includes a scatter plot of two numeric variables along with an overlaid linear or nonlinear fit line that enables you to perform a regression analysis. Here is a sample.



Use the REGRESSIONPLOT statement to generate a regression plot in GTL. The REGRESSIONPLOT statement supports only models of one independent and one dependent variable. You can specify one of three types of regression equation: linear, quadratic, or cubic. In addition to the regression line, the REGRESSIONPLOT statement can compute confidence levels for the fitted line. The REGRESSIONPLOT statement syntax is described in "REGRESSIONPLOT" in *SAS Graph Template Language: Reference*. For an example, see "REGRESSIONPLOT Statement" in *SAS Graph Template Language: Reference*.

Scatter Plots

A scatter plot generates a marker for each observation that has nonmissing X and Y values. Markers can be symbols or character strings. Symbol markers can be labeled. Here is a sample.

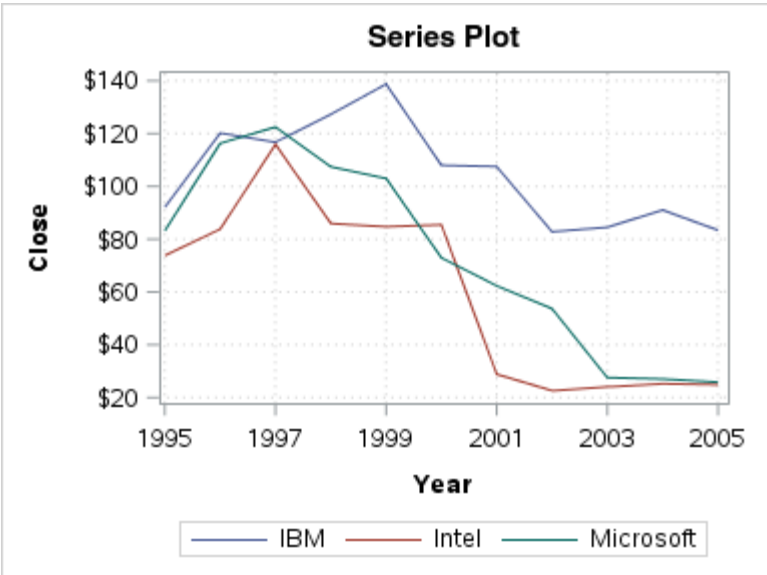


Use the SCATTERPLOT statement to generate a scatter plot in GTL. The SCATTERPLOT statement syntax is described in "SCATTERPLOT" in *SAS Graph Template Language: Reference*. For examples, see

- "Grouped Scatter Plot" in *SAS Graph Template Language: Reference*
- "Discrete Scatter Plot" in *SAS Graph Template Language: Reference*

Series Plots

A series plot displays a series of line segments that connect observations of input data. It is typically used to show time-dependent data. Here is a sample.

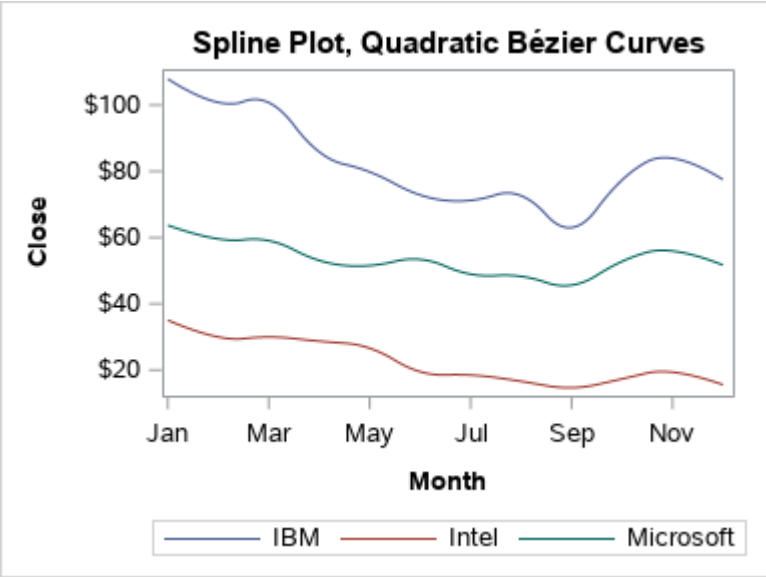


Use the SERIESPLOT statement to generate a series plot in GTL. The SERIESPLOT statement description is described in "SERIESPLOT" in *SAS Graph Template Language: Reference*. For examples, see the following:

- "Overlay Series Plot" in *SAS Graph Template Language: Reference*
- "Grouped Series Plot" in *SAS Graph Template Language: Reference*
- "Series Plot with Line-Thickness Response and Arrowheads" in *SAS Graph Template Language: Reference*

Spline Plots (Quadratic Bézier Curves)

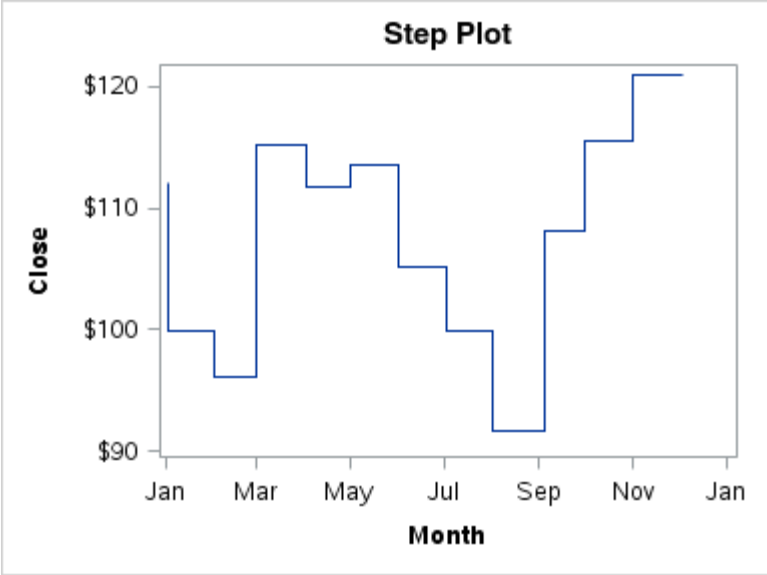
A quadratic Bézier spline plot is a series plot with a quadratic Bézier spline interpolation that produces smooth curves. Here is a sample.



Use the SERIESPLOT statement with the SPLINETYPE=QUADRATICBEZIER option to generate a spline plot in GTL.The SERIESPLOT statement syntax is described in "SERIESPLOT" in *SAS Graph Template Language: Reference*. For an example, see "Series Plot with Quadratic Bézier Spline Curves" in *SAS Graph Template Language: Reference*.

Step Plots

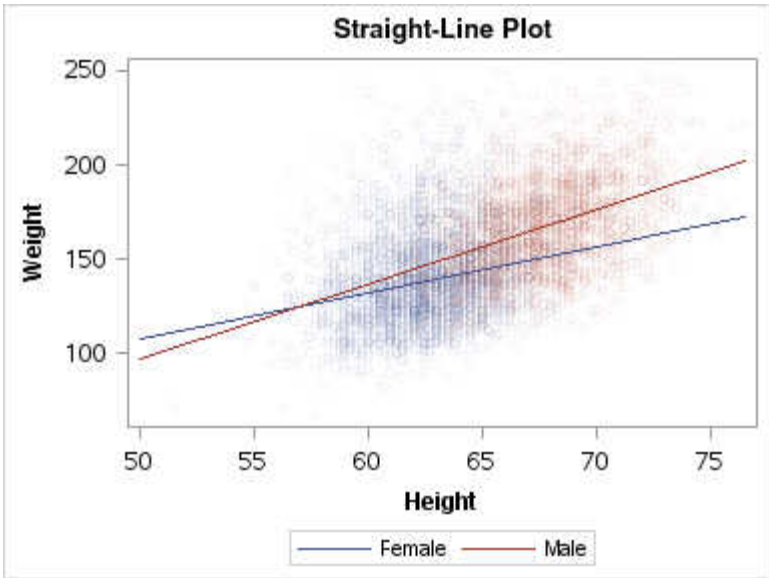
A step plot displays a series of horizontal and vertical line segments that connect observations of input data. The plots use a step function to connect the data points. The vertical line can change at each step. Here is a sample.



Use the STEPPLOT statement to generate a step plot in GTL. The STEPPLOT statement syntax is described in "STEPPLOT" in *SAS Graph Template Language: Reference*. For an example, see "STEPPLOT Statement" in *SAS Graph Template Language: Reference*.

Straight-Line Plot (Point and Slope)

A straight-line plot is a straight line of a specified slope and intercept point. Here is a sample.

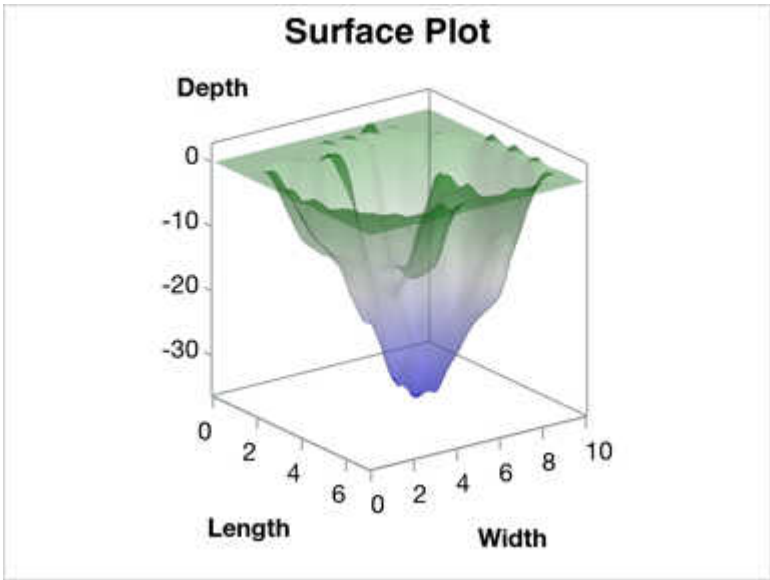


Use the LINEPARM statement to generate a straight-line chart in GTL. The X= and Y= arguments specify the coordinates for the intercept point, and the SLOPE= argument specifies the line slope. You can specify a numeric value or expression as the argument value, or you can specify a numeric column as the argument value. You must use the LINEPARM statement with another plot statement that establishes

the axes. The LINEPARM statement syntax is described in "LINEPARM" in *SAS Graph Template Language: Reference*. For an example, see "LINEPARM Statement" in *SAS Graph Template Language: Reference*.

Surface Plots

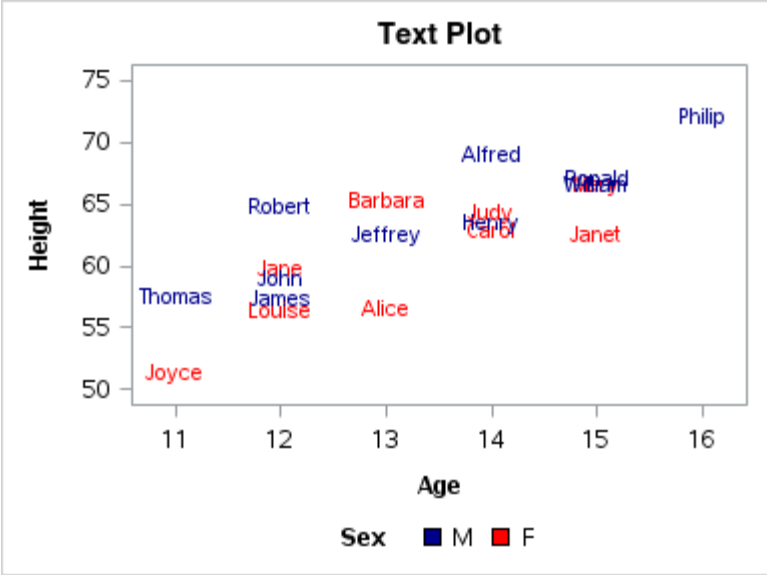
A 3-D surface plot forms an evenly spaced grid of horizontal values (X and Y) and one or more vertical values (Z) for each combination. Here is a sample.



Use the SURFACEPLOTPARM statement to generate a 3-D surface plot in GTL. The input data should form an evenly spaced grid of horizontal values (X and Y) and one or more vertical values (Z) for each combination. You can use the G3GRID procedure (which requires a SAS/GRAPH license) to interpolate the necessary values. The input data must be sorted by Y and X in order to obtain the correct lighting. The SURFACEPLOTPARM statement syntax is described in "SURFACEPLOTPARM" in *SAS Graph Template Language: Reference*. For an example, see "Surface Plot" in *SAS Graph Template Language: Reference*.

Text Plots

A text plot displays the associated text values at X and Y locations in the graph. It is similar to a scatter plot. The text can be numbers or characters. Here is a sample.



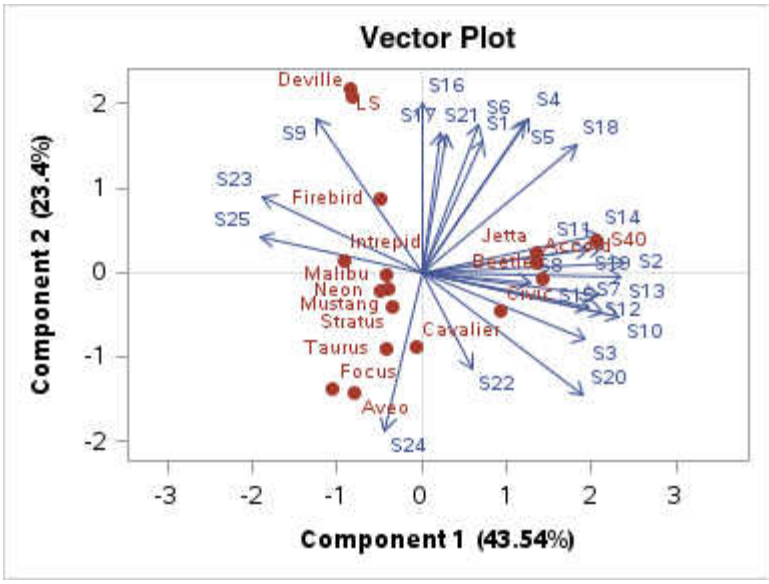
Use the TEXTPLOT statement to generate a text plot in GTL. The input data provides the text strings and their X and Y locations. By default, the text plot shows the text strings only. The bounding box for each text string can be outlined, filled, or both.

Note: This feature applies to SAS 9.4M2 and to later releases.

The TEXTPLOT statement syntax is described in "TEXTPLOT" in *SAS Graph Template Language: Reference*. For an example, see "TEXTPLOT Statement" in *SAS Graph Template Language: Reference*.

Vector Plots

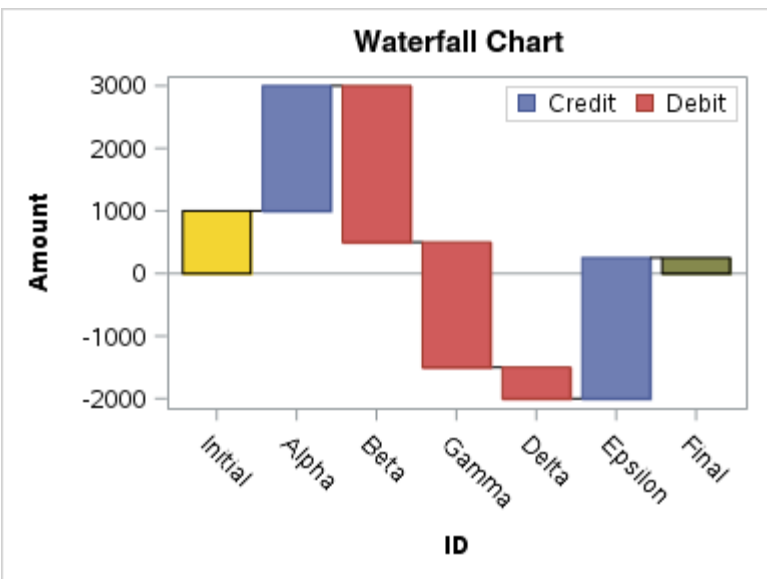
A vector plot draws arrows from a point of origin to each data point. Vectors are directed line segments. A vector plot uses vectors to represent both direction and magnitude at each point. Here is a sample.



In a vector plot, each vector starts at 0, 0 in the data space and is terminated with an open arrowhead. Zero-length vectors are represented by a dot at the starting point. Use the VECTORPLOT statement to generate a vector plot in GTL. The VECTORPLOT statement syntax is described in "VECTORPLOT" in *SAS Graph Template Language: Reference*. For an example, see "VECTORPLOT Statement" in *SAS Graph Template Language: Reference*.

Waterfall Charts

A waterfall chart shows how the value of a variable increases or decreases until it reaches a final value. A waterfall chart is often used to show credit and debit transactions or successive changes to a given state. Here is a sample.

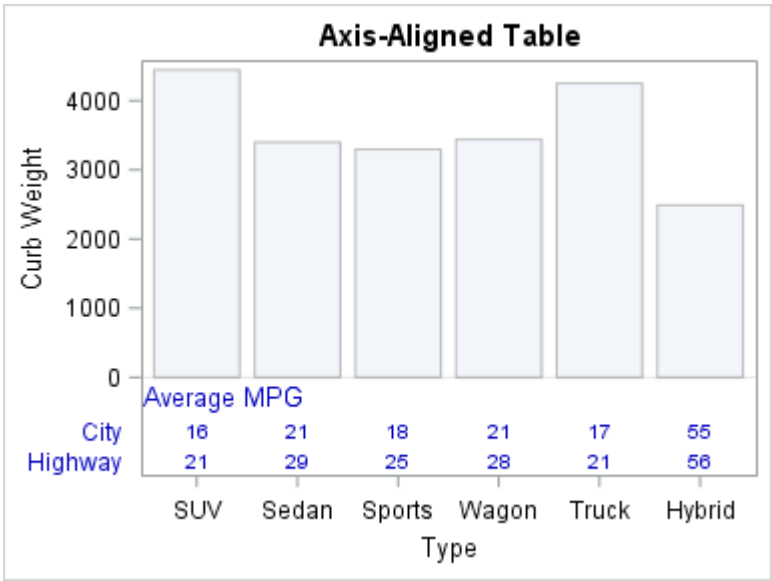


In the chart, bars represent an initial value of Y and a series of intermediate values identified by X leading to a final value of Y. The bars are calculated from data that

are called transaction bars. The transaction bars represent the values of the RESPONSE variable across a series of intermediate values for the specified CATEGORY variable. Use the WATERFALLCHART statement to generate a waterfall chart in GTL.The WATERFALLCHART statement syntax is described in "WATERFALLCHART" in *SAS Graph Template Language: Reference*. For an example, see "WATERFALLCHART Statement" in *SAS Graph Template Language: Reference*.

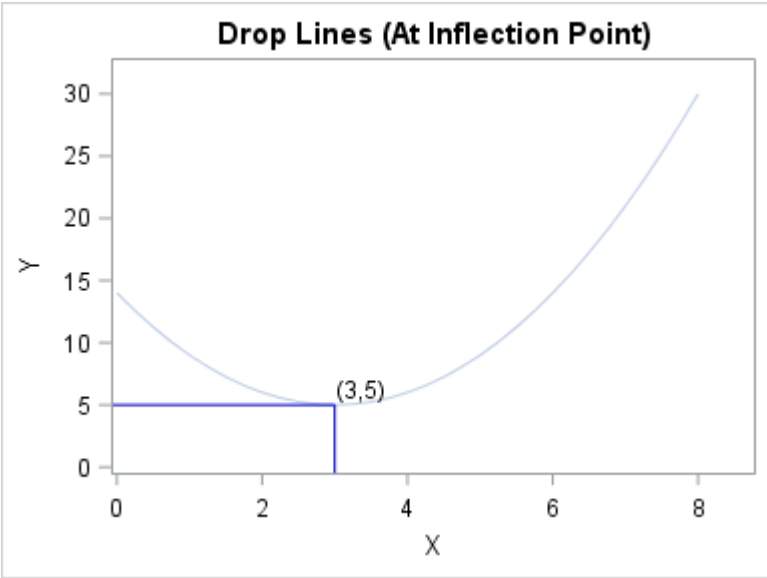
Gallery of Additional Plot Features

Axis-Aligned Tables



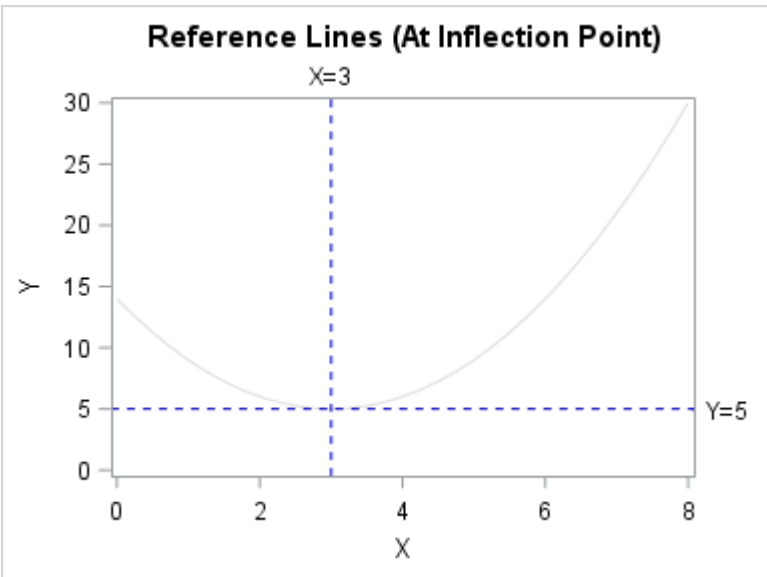
An axis-aligned table is generated by the AXISTABLE statement. The AXISTABLE statement enables you to place values along the X or Y axis that are aligned with the axis tick values. It offers more flexibility than the BLOCKPLOT statement, which is used to denote changes in block values along the axis. The AXISTABLE statement syntax is described in "AXISTABLE" in *SAS Graph Template Language: Reference*. For an example, see "AXISTABLE Statement" in *SAS Graph Template Language: Reference*.

Drop Lines



Drop lines are drawn by the DROPLINE statement. A drop line is drawn perpendicular from a specified point to the X (bottom), X2 (top), Y (left), or Y2 (right) axis. You must use the DROPLINE statement with another plot statement that establishes the axes. The DROPLINE statement syntax is described in "DROPLINE" in *SAS Graph Template Language: Reference*. For an example, see "DROPLINE Statement" in *SAS Graph Template Language: Reference*.

Reference Lines



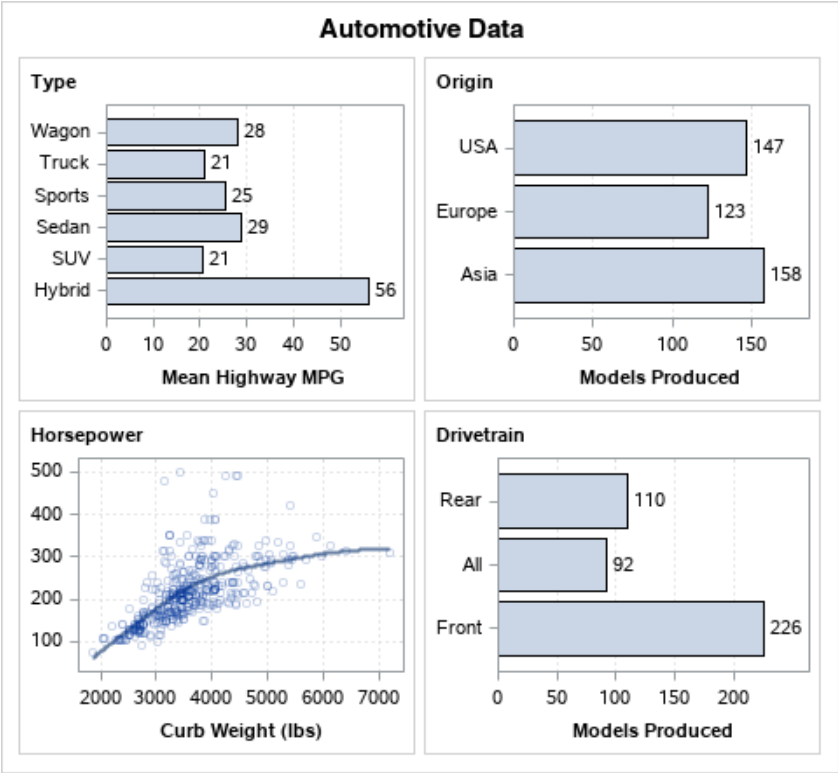
A reference line is drawn by the REFERENCELINE statement. Specifying the X= option creates a line perpendicular to the X axis at an X-intercept. Specifying the Y= option creates a line perpendicular to the Y axis at a Y-intercept. You must use the REFERENCELINE statement with another plot statement that establishes the axes. The REFERENCELINE statement syntax is described in "REFERENCELINE" in *SAS Graph Template Language: Reference*. For examples, see the following:

- "Specifying a Single Reference Line" in *SAS Graph Template Language: Reference*
- "Specifying Reference Lines Using Data Columns" in *SAS Graph Template Language: Reference*
- "Specifying Reference Lines Using the COLN and COLC Functions" in *SAS Graph Template Language: Reference*

Gallery of Multicell Graphs

Grid Graphs

A grid graph is a simple grid of similar or different plots. The plots are placed in cells arranged in rows and columns. Columns have the same width and rows have the same height, which means that each cell is of the same size. The plots in each cell are independent of the other graphs in the grid. Here is an example.

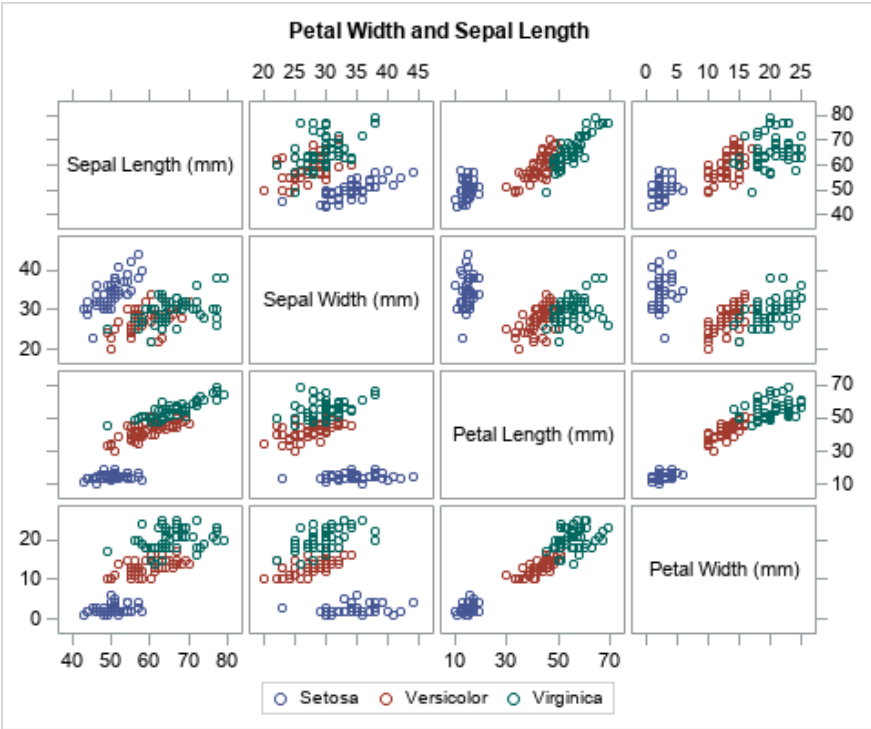


The GRIDDED layout can also be used with ENTRY statements to add one or more inset tables of values to a graph. Use the LAYOUT GRIDDED statement to create a gridded graph in GTL. The LAYOUT GRIDDED statement syntax is described in "LAYOUT GRIDDED" in *SAS Graph Template Language: Reference*. For examples, see the following:

- "Creating a 2-By-2 Grid of Plots" in *SAS Graph Template Language: Reference*
- "Creating an Inset" in *SAS Graph Template Language: Reference*

Scatter Plot Matrices

A scatter plot matrix is a panel graph of scatter plots for multiple combinations of variables. You can overlay fit plots and ellipses on your scatter plots. Here is a sample.

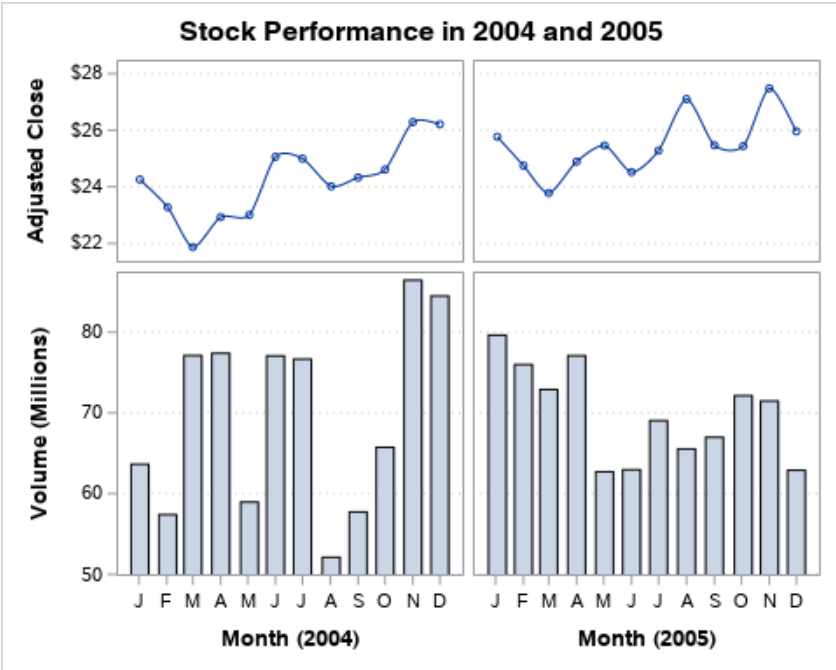


A pairwise matrix, shown in the sample, produces a symmetric scatter plot matrix from a list of at least two variables. Use the SCATTERPLOTMATRIX statement to generate a pairwise matrix of scatter plots in GTL. By default, the SCATTERPLOTMATRIX statement produces a symmetric scatter plot matrix from a list of at least two variables. The columns of the matrix are in the same left-to-right order as the column list. The rows of the matrix are in the same bottom-to-top order as the column list. You can reverse the order. You must place the SCATTERPLOTMATRIX statement in a GRIDDED layout block.

The SCATTERPLOTMATRIX statement syntax is described in "SCATTERPLOTMATRIX" in *SAS Graph Template Language: Reference*. For an example, see "SCATTERPLOTMATRIX Statement" in *SAS Graph Template Language: Reference*.

Lattice Graphs

A lattice graph is a lattice of similar or different graphs. The graphs are placed in cells arranged in rows and columns. By default, columns are of the same width and rows are of the same height. However, you can alter the width of individual columns and the height of individual rows to vary cell sizes. The graphs in each cell can be independent or they can share common axes with the other graphs in the lattice. Here is an example.



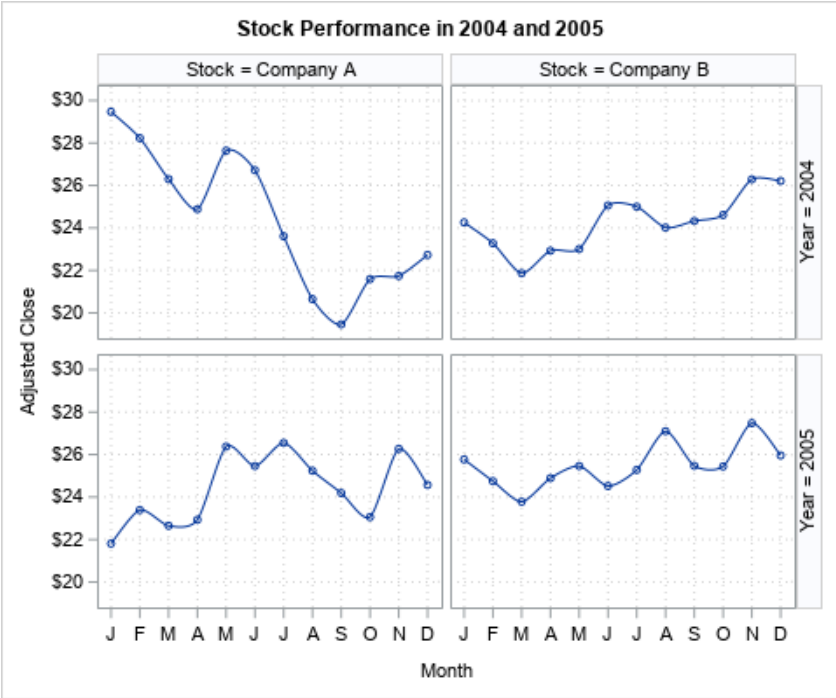
In this example, the graphs in the lattice share common row and column axes, and the rows have different heights. Use the LAYOUT LATTICE statement to generate a latticed graph in GTL. The LAYOUT LATTICE statement syntax is described in "LAYOUT LATTICE" in *SAS Graph Template Language: Reference*. For examples, see the following:

- "Lattice with Internal Axes" in *SAS Graph Template Language: Reference*
- "Lattice with Row and Column Axes" in *SAS Graph Template Language: Reference*
- "Lattice with Sidebar" in *SAS Graph Template Language: Reference*

Data-Driven Lattice Graphs

A data-driven lattice graph is a lattice of similar graphs that is generated from a row variable and a column variable. The graph in each cell is defined by a graph prototype. The row and column variable values that intersect a cell are used to

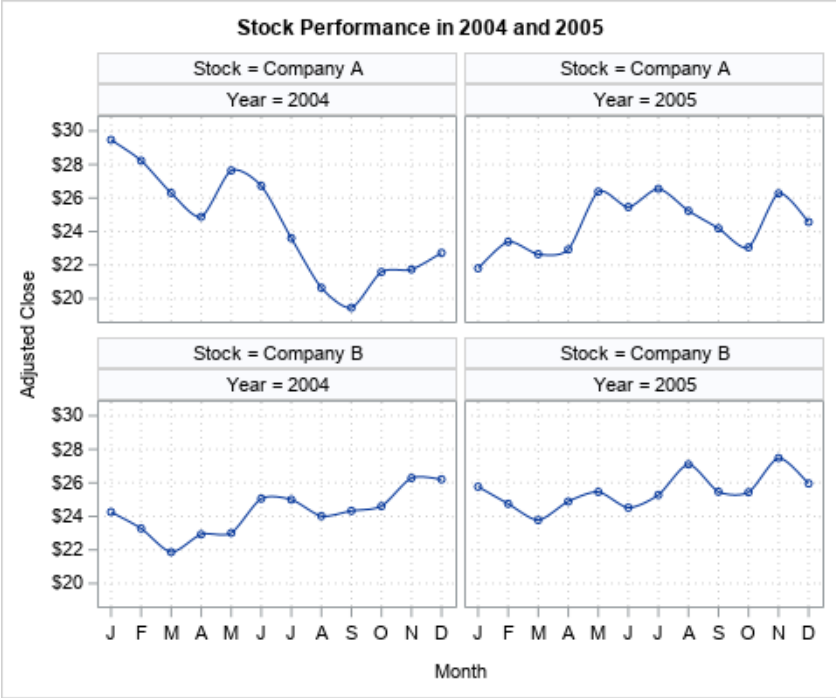
generate the graph in that cell from the prototype. The graphs in the lattice share common row and column axes. Here is an example.



Column and row headers indicate the values of the column and row variables for each cell. Use the LAYOUT DATALATTICE statement to generate a data-driven lattice graph in GTL. The LAYOUT DATALATTICE statement syntax is described in "LAYOUT DATALATTICE" in *SAS Graph Template Language: Reference*. For an example, see "LAYOUT DATALATTICE" in *SAS Graph Template Language: Reference*.

Data-Driven Panel Graphs

A data-driven panel graph is a panel of similar graphs that is generated from one or more classification variables. The graph in each cell is defined by a graph prototype. The classification variable values for a cell are used to generate the graph in that cell from the prototype. The graphs in the panel share common row and column axes. Here is an example.



A header for each cell indicates the values of the classification variables that were used to generate the graph in that cell. Use the LAYOUT DATAPANEL statement to generate a data-driven panel graph in GTL. The LAYOUT DATAPANEL statement syntax is described in "LAYOUT DATAPANEL" in *SAS Graph Template Language: Reference*. For an example, see "LAYOUTDATAPANEL Statement" in *SAS Graph Template Language: Reference*.

PART 4

Creating Graphs Using GTL

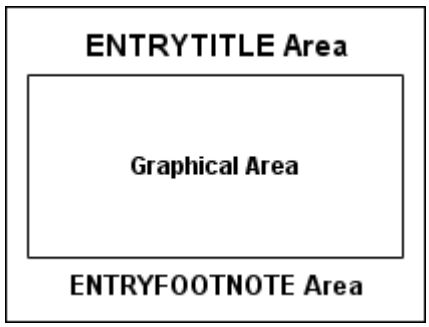
11

Creating Overlay Graphs Using the OVERLAY Layout

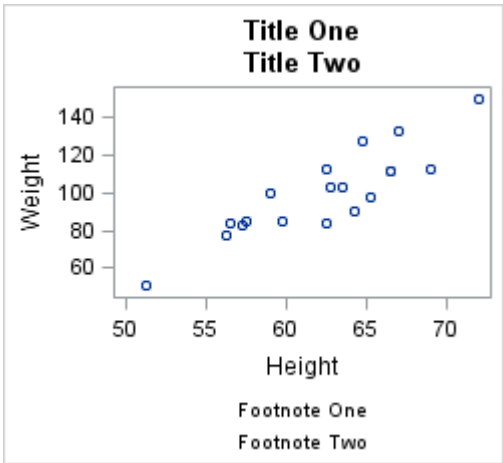
The LAYOUT OVERLAY Statement

The LAYOUT OVERLAY statement builds a 2-D, 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 about 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 the following:

- 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 such as the following:

  - 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 aspect ratio of the rectangular area of the plot wall (ASPECTRATIO= option)

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

Statements You Can Use in an OVERLAY Block

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

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 standalone plot statement, such as SCATTERPLOT or SERIESPLOT, in order to produce a usable graph. See Chapter 6, “Overview of the GTL Statements,” on page 31 for lists of standalone and dependent statements.

For example, if you execute the following template, it produces an empty graph.

```
proc template;
  define statgraph test;
    begingraph;
      layout overlay;
        referenceline x=10;
      endlayout;
```

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

The following message also appears in the SAS log:

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

A REFERENCELINE statement can be used only within 2-D overlay-type layouts (OVERLAY, OVERLAYEQUATED, or PROTOTYPE) with a standalone plot statement. (See "REFERENCELINE" in *SAS Graph Template Language: Reference*.) The standalone plot statement must provide an axis data range that is sufficient for determining the reference line axis extents. In this example, a standalone plot is not provided in the REFERENCELINE statement's layout. As a result, the reference line position cannot be determined.

Restrictions on Statement Combinations

Certain combinations of contained statements produce unexpected results. Consider the template in the following example.

Example Code 11.1 *Statement Combination Example*

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

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

When this template is executed, the following message is written to the SAS log:

```
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 types. In this example, the first statement in the layout is LINECHART. The LINECHART statement treats the CATEGORY=column as a categorical variable (regardless of data type) and builds a DISCRETE (categorical) axis. Because LINECHART is the first statement in this example layout, it determines the layout's X-axis type, which is discrete in this case.

Subsequent plots must be compatible with a discrete X axis. The end result of this example is a graph containing only the box plot output.

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. If this example specified the SCATTERPLOT statement instead of REGRESSIONPLOT, no warning would be issued because a scatter plot can be displayed on either a discrete or linear X and Y axis.

In the following layout block, the REGRESSIONPLOT and LINECHART statements have been switched.

```
layout overlay;
  regressionplot x=age y=weight;
  linechart category=age response=weight;
endlayout;
```

When the template in Example Code 11.1 on page 100 is executed with this layout block, the following message is written to the SAS log:

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

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

Even though a SCATTERPLOT can be included on either LINEAR or DISCRETE axes, the combination in the following layout block is not valid:

```
layout overlay;
  scatterplot x=age y=weight;
  linechart category=age response=weight;
endlayout;
```

When the template in Example Code 11.1 on page 100 is executed with this layout block, the following message is written to the SAS log:

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

In this case, the SCATTERPLOT statement sets the X- or Y-axis type to be 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 statement requires a discrete axis. Therefore, the LINECHART is not displayed. If you switch the statements, both plots are drawn because after the X axis is fixed to be discrete, the scatter plot can display numeric values on a discrete axis.

When a character variable is used, the axis-type conflict often does not arise. The combination in the following layout block works regardless of statement order.

```
layout overlay;
  scatterplot x=sex y=weight;
  linechart category=age response=weight;
endlayout;
```

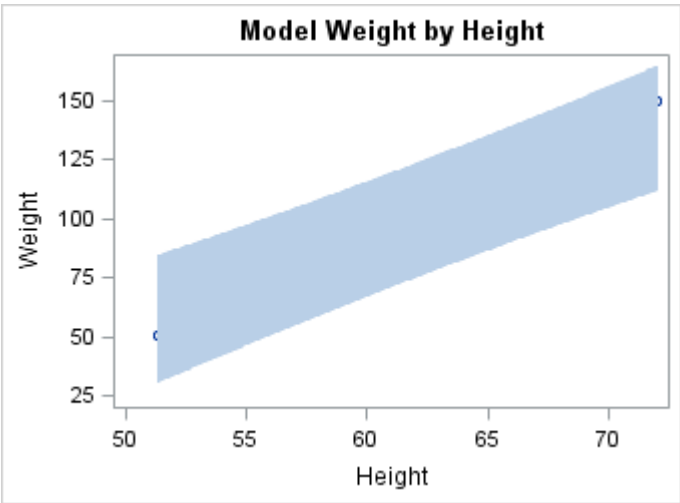
In either case, the discrete X axis displays a combination of Age values with a line above and Sex values with scatter points above.

Statement Order

The order in which you specify the plots within the LAYOUT OVERLAY block is important. The first plot that you specify in the layout block is drawn first. The second plot that you specify is then added on top of the first plot, and so on. It is possible for one plot's data to obscure the data beneath it. This situation is most likely when you are working with plots that produce filled areas, such as histograms and band plots. For example, the following template specifies the MODELBAND statement after the SCATTERPLOT and REGRESSIONPLOT statements.

```
proc template;
  define statgraph carsprofile;
    begingraph;
      entrytitle "Model Weight by Height";
      layout overlay;
        scatterplot x=height y=weight;
        regressionplot x=height y=weight /
          cli="cli";
        modelband "cli";
      endlayout;
    endgraph;
  end;
run;

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

Here is the output.



The filled model band plot is drawn after the scatter plot and regression plot, so the band plot obscures the data beneath it. To avoid this problem, specify the model band plot first. Another way to solve this problem is to make the model band plot partially transparent. Then the scatter plot markers and the regression line show through the band plot. However, as a best practice, specify any plot that creates filled areas first in the template definition, before you specify other graph elements.

Managing Axes in OVERLAY Layouts

Overview

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

- which axes are displayed
- the axis type of each axis (linear, time, and so on)
- 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 that 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.

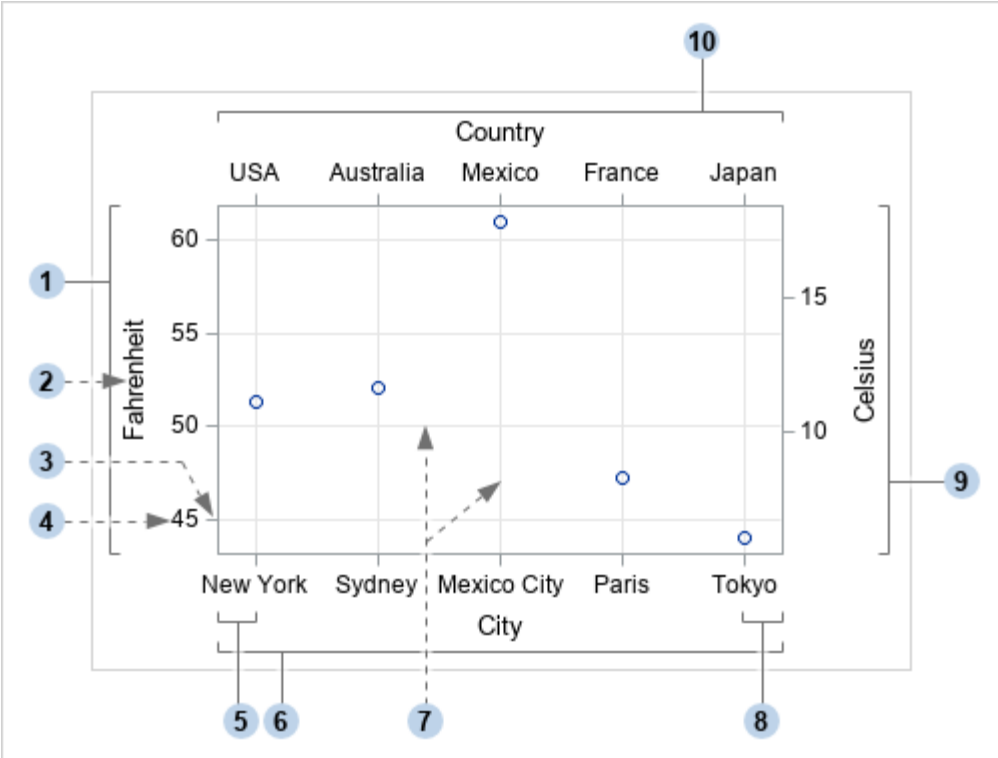
Axis Terminology

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=)

The following figure identifies the axis features in a typical plot.



1 Y axis
2 Y-axis label
3 Tick mark
4 Tick value
5 X-axis minimum offset
6 X axis
7 Axis grid lines
8 X-axis maximum offset
9 Y2 axis
10 X2 axis

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. Consider the following data:

Example Code 11.2 *Temperature Data*

```
data temps;
  input City $1-11 Country :$9. Fahrenheit;
  Celsius= (fahrenheit-32)*(5/9);
  cards;
New York     USA 52
Sydney       Australia 53
Mexico City  Mexico 64
Paris        France 47
Tokyo        Japan 43
;
run;
```

The following example plots the city and Fahrenheit temperature on the X and Y axes.

```
proc template;
 define statgraph axesXY;
  begingraph;
   entrytitle "X and Y Axes";
    layout overlay;
      scatterplot  x=City y=Fahrenheit;
    endlayout;
  endgraph;
 end;
run;

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

Here is the output.

Explicitly, the layout block means the following:

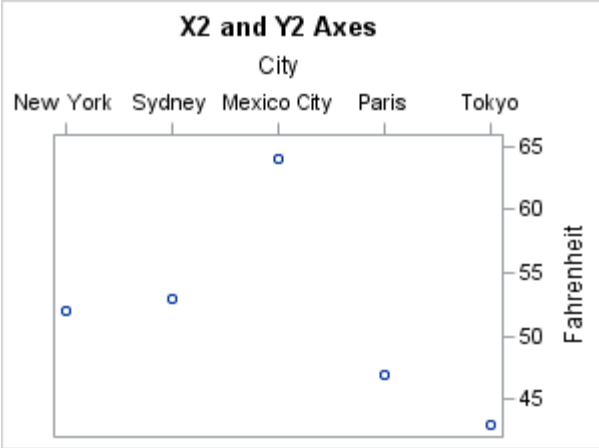
```
layout overlay;
  scatterplot x=city y=fahrenheit / xaxis=x yaxis=y;
endlayout;
```

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. The following layout block specifies the X2 and Y2 axes.

```
layout overlay;
```

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

Here is example output.



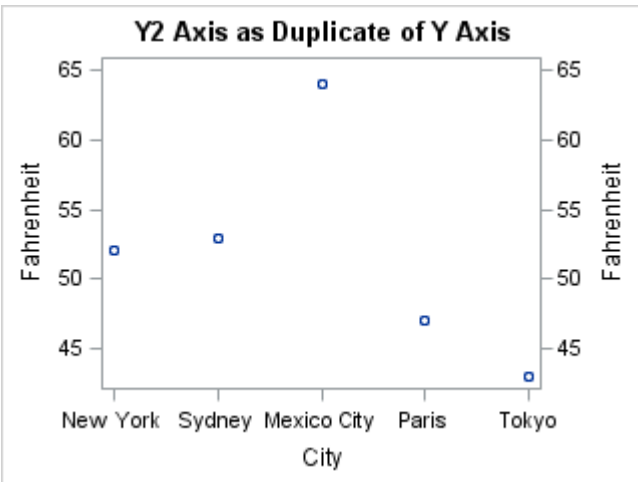
A single plot statement can activate one horizontal and one vertical axis. It cannot activate both horizontal or both vertical axes. Thus, to see both a Y and Y2 axis based on the same Y column, you could specify an additional plot statement as shown in the following layout block.

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

This layout block can be more compactly written as follows:

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

Here is example output.



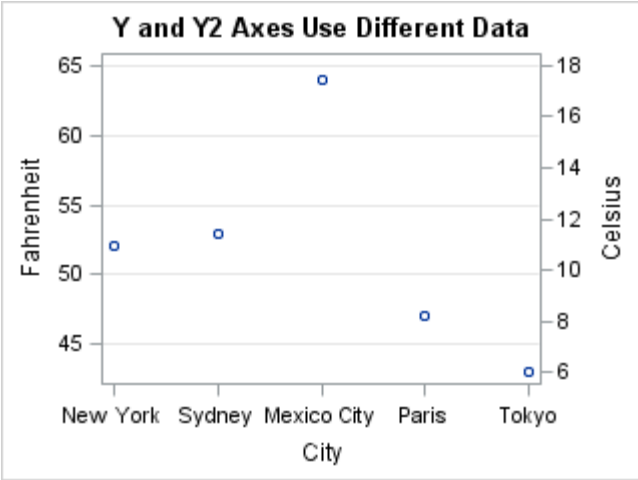
Note that this coding produces two overlaid scatter plots and that each plot has five markers. Because the five (X,Y) value pairs and the five (X,Y2) value pairs are identical, the Y and Y2 axes are identical and the markers are exactly superimposed. However, it is not necessary to create a second plot 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 113.

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. In the following example layout block, even though the Y and Y2 columns are different, the primary and secondary Y axes represent the same data range in different units.

```
layout overlay / yaxisopts=(griddisplay=on);
  scatterplot  x=City y=Fahrenheit;
  scatterplot  x=City y=Celsius / datatransparency=1 yaxis=y2;
endlayout;
```

Here is example output.

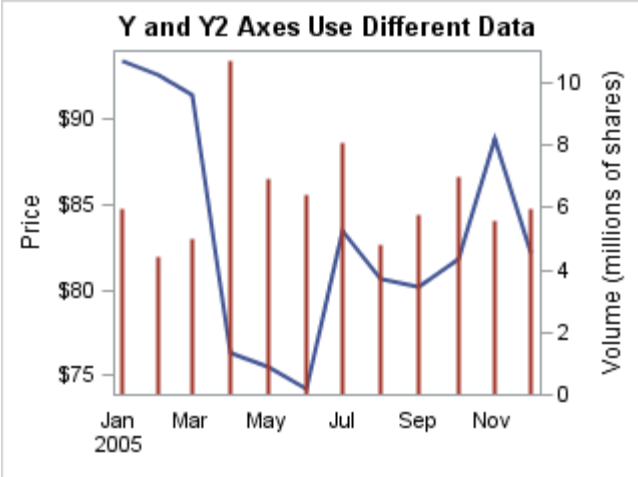


In such cases, the positioning of the tick values on each axis should be coordinated so that the grid lines represent the same temperature on each axis. "Axis Thresholds" on page 125 provides example code that shows how to coordinate the tick value positions.

In the following example, the primary and secondary Y axes represent different data ranges.

```
proc template;
  define statgraph overlayaxes.axesYY2;
    begingraph;
      entrytitle "Y and Y2 Axes Use Different Data";
      layout overlay / cycleattrs=true xaxisopts=(display=(tickvalues))
        y2axisopts=( offsetmin=0 label="Volume (millions of shares)");
        seriesplot  x=date y=close / lineattrs=(pattern=solid
thickness=2);
        needleplot  x=date y=eval(volume/10**6) /
          lineattrs=(pattern=solid thickness=2) yaxis=y2;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.stocks template=overlayaxes.axesYY2;
  where year(date)=2005 and Stock="IBM";
  format volume 3.;
  label close="Price";
run;
```

Here is the output.



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.

Avoiding Plot Axis 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 in a plot statement to ensure that plot is used to determine the axis type.
- Declare an axis type in 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 that it uses, regardless of its order within the layout block. This is useful when you want a certain stacking order for the plots, but do not want the first plot to set the axis features, such as axis type and default axis label.

In the following layout block, the BARCHART sets the X axis to be DISCRETE and the Y axis to be LINEAR:

```
layout overlay;
  scatterplot x=age y=weight;
  barchart category=age response=weight / primary=true;
endlayout;
```

All layouts that manage axes provide options that enable you to control the axis features. The following layout block shows how to declare an axis type for the X axis.

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

```
  barchart category=age response=weight;
endlayout;
```

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 "Specifying Axis Options" on page 113.

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 as shown in the following layout block.

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

When this template code is executed, the following message is written to the SAS log:

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

Reversing the order of the plot statements as shown in the following layout block also does not work.

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

When this template code is executed, the following message is written to the SAS log:

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

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 use only the X and Y axes, not X2 or Y2.

By default, you 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 template. In the template code, the PROC TEMPLATE block defines a style named AXIS_WALL, and then the ODS HTML statements sets the AXIS_WALL style as the active style for output that is directed to the HTML destination:

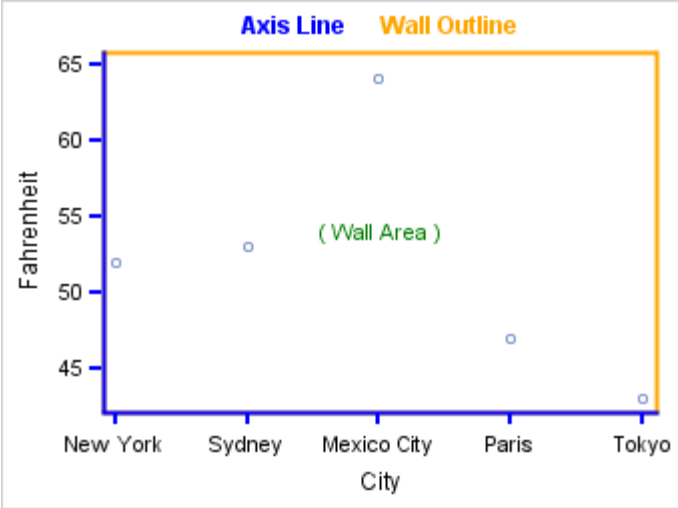
```
/* Specify a path for the ODS output */
filename odsout "output-path";

proc template;
 define style Styles.axis_wall_style;
 parent=styles.htmlblue;
 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 _all_ close;
ods html path=odsout file="AxisWallStyle.html" style=axis_wall_style;
```

If the following code 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.

```
proc template;
 define statgraph axis_wall1;
  begingraph /  border=true;
    entrytitle textattrs=(color=blue) "Axis Line      "
      textattrs=(color=orange) "Wall Outline";
    layout overlay / walldisplay=(fill outline);
      scatterplot  x=City y=Fahrenheit / datatransparency=.5;
      entry textattrs=(color=green) "( Wall Area )";
    endlayout;
  endgraph;
 end;
run;

proc sgrender data=temps template=axis_wall1;
run;
ods html close;
ods html; /* Not required in SAS Studio */
```

Note: See Example Code 11.2 on page 105.

Here is the output.



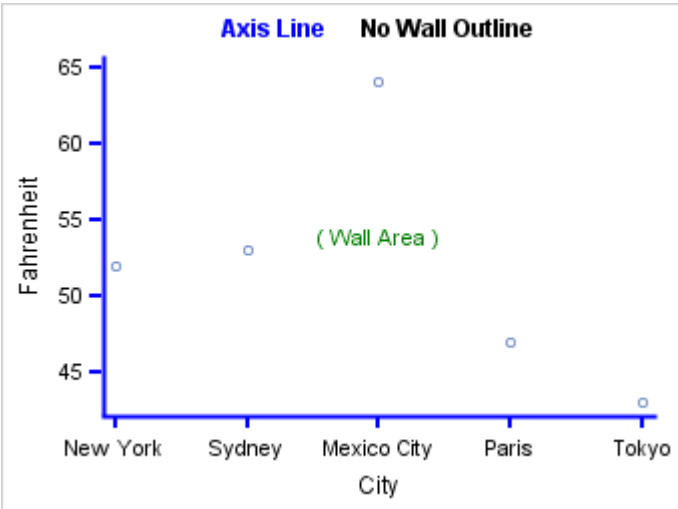
Most styles set the axis lines and the wall outline to be the same color, line pattern, and thickness, so it is impossible to see the difference. Sometimes you might not want to see the wall outline, or you might want to change the wall color. These types of changes can be set in a custom style or with the WALLDISPLAY= option on the LAYOUT OVERLAY statement. For example, the GTL default for the wall is to display the wall fill and outline. The following code fragment shows how to use the style template to turn off the wall outline:

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

Unless the WALLDISPLAY= option is specified in the LAYOUT OVERLAY statement, the FRAMEBORDER=OFF attribute in the style turns off the wall outline. This next code fragment shows how to use the WALLDISPLAY= option on the LAYOUT OVERLAY statement to turn off the wall outline:

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

Here is an example.



The WALLDISPLAY= option overrides the FRAMEBORDER= attribute in the current style.

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
	LineThickness	Dimension	Grid line thickness
	ContrastColor	Color	Grid line color
GraphLabelText	Color	Color	Axis label text color
	Font	font-specification[1]	Axis label font
GraphValueText	Color	Color	Axis tick value text color
	Font	font-specification[1]	Axis tick value font

1 A style font-specification includes 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

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

Here are some examples.

- The following example displays the label text in bold:

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

- The following example displays grid lines:

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

- The following example specifies a dot pattern for grid lines:

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

- The following example makes the 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 Template Language: Reference*.

Specifying Axis Options

To set axis options on the LAYOUT OVERLAY statement, you use the following syntax:

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

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

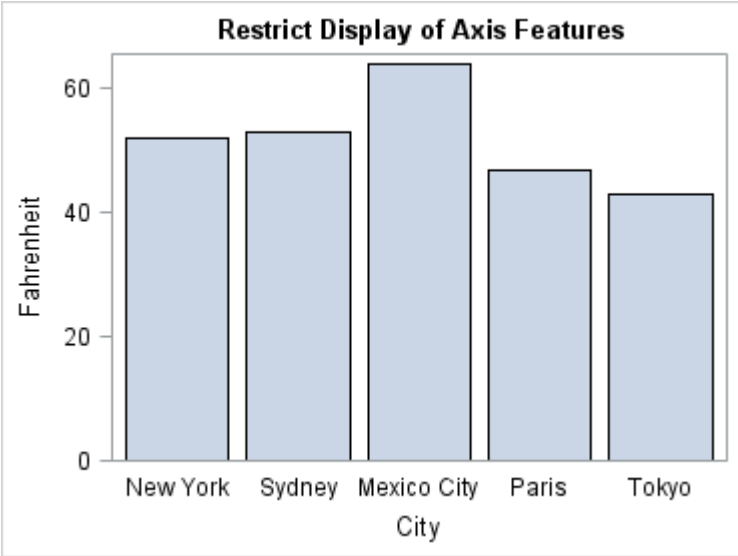
One of the basic options that you can set for any axis is DISPLAY= *keyword* | (*feature-list*). Four features are available for the *feature-list*: LINE, TICKS, TICKVALUES, and LABEL. The keywords STANDARD and ALL are equivalent to specifying the full list: (LINE TICKS TICKVALUES LABEL). You can also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

Example: Some plots do not need TICKS on all axes. The DISPLAY= axis option in the following example eliminates the ticks on the X axis by omitting the TICKS value in the *feature-list*.

```
proc template;
 define statgraph DisplayOpts;
  begingraph;
   entrytitle "Restrict Display of Axis Features";
    layout overlay / xaxisopts=(display=(label tickvalues line));
      barchartparm x=City y=Fahrenheit / orient=vertical;
    endlayout;
  endgraph;
 end;
run;

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

Note: See Example Code 11.2 on page 105.

Here is the output.



We now return to the common situation where you want a duplicated Y2 axis. The following layout block shows the most efficient way to do it:

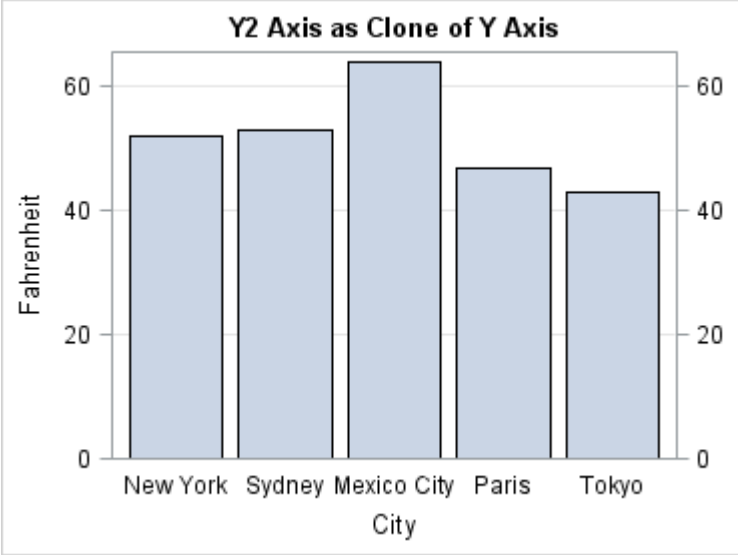
```
layout overlay / yaxisopts=( displaysecondary=standard );
  barchartparm category=city response=fahrenheit;
endlayout;
```

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 that you want to be displayed. The values available for the DISPLAYSECONDARY= option are the same as those of the DISPLAY= option. The following layout block specifies that the secondary axis label is not to 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 category=city response=fahrenheit;
endlayout;
```

Here is example output.



Axis Type

To determine axis types, the OVERLAY container examines all of the standalone 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 standalone plot, or a plot is designated as PRIMARY, the rules are as follows:

- If the plot statement that is mapped to an axis treats data values as discrete (such as the CATEGORY= column of the BARCHART or the X= column of the 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 151 for examples. Otherwise, the numeric axis type is LINEAR, the general numeric axis type. See “LINEAR Axes” on page 131 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 158 for examples.
- If a TYPE= *axis-type* option is specified, that is the type used. Plots that cannot support that axis type are not 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 GTL to use. Note the following:

- 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 are 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.

Here are some examples:

- For the following layout block, the BARCHART is considered the primary plot because it is the first standalone plot that is specified in the layout and no other plot has been set as the primary plot.

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

  A BARCHART requires a discrete X axis. You cannot change the axis type. It does not matter whether Quarter is a numeric or character column. Because the SERIESPLOT can use a discrete axis, the overlay is successful.

- For the following layout block, the first SERIESPLOT is considered primary.

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

  If the Quarter column is numeric and has a date format, then the X-axis type is TIME. If the column is numeric, but does not have a date format, then the axis type is LINEAR. If the column is character, then the axis type is DISCRETE.

- 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.

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

  It does not matter whether Quarter is a numeric or character column.

- For the following layout block, the SERIESPLOT is the primary plot.

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

  If Quarter is a character column, a discrete axis is used and the overlay is successful. However, if Quarter is a numeric column and either a TIME or LINEAR axis is used, the BARCHART overlay fails, and the following message is written to the log:

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

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 131, “TIME Axes” on page 151, and “LOG Axes” on page 158.

Axis Labels

Default Axis Labels

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 112. The following examples show how axis labels are determined and how to set an axis label.

Consider the following DATA step, which generates bacteria and virus growth test data.

Example Code 11.3 *Growth Data*

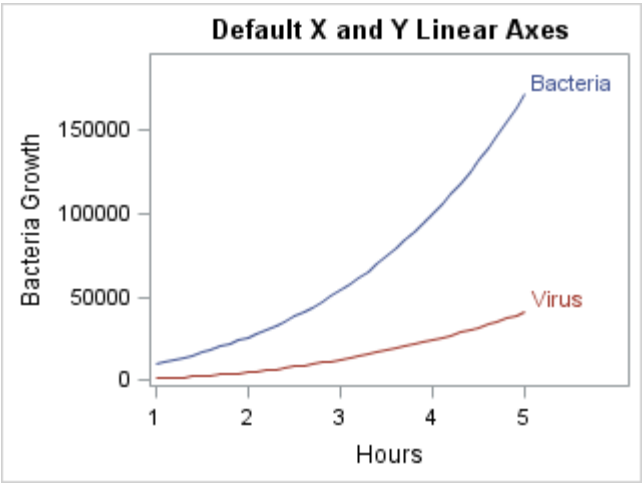
```
data growth;
  do Hours=1 to 5 by .1;
    Growth = 10**hours;
    Bacteria = 1000*10**( sqrt(Hours ));
    Virus = 1000*10**(log(hours));
    label bacteria = "Bacteria Growth" virus="Virus Growth";
    output;
  end;
run;
```

To plot the growth trend for both Bacteria and Virus in the same graph, you 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.

```
proc template;
  define statgraph axislabeldefault1;
```

```
      begingraph;
        entrytitle "Default X and Y Linear Axes";
        layout overlay / cycleattrs=true;
          seriesplot x=Hours y=Bacteria/ curvelabel="Bacteria";
          seriesplot x=Hours y=Virus / curvelabel="Virus";
        endlayout;
      endgraph;
    end;
  run;

  proc sgrender data=growth template=axislabeldefault1;
  run;
```

Here is the output.

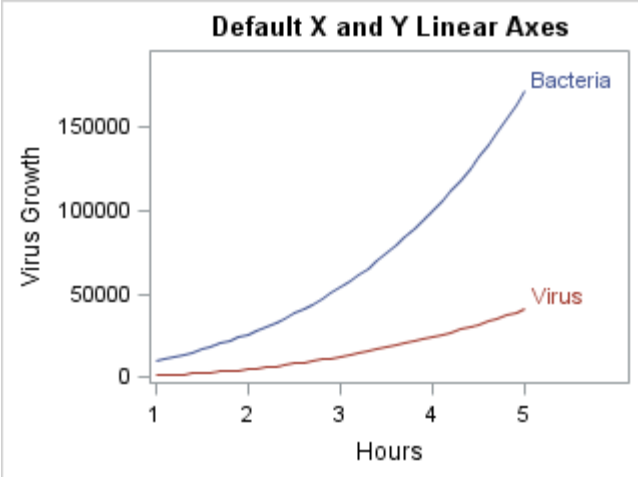


If you designate another plot statement as "primary," its X= and Y= columns are used to label the axes. The PRIMARY= option is useful when you desire a certain stacking order of the overlays, but you want the axis characteristics to be determined by a plot statement that is not the default primary plot statement. In the following layout block, 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;
```

Here is example output.

Figure 11.1 *Primary Plot Determines Default Axis Label*



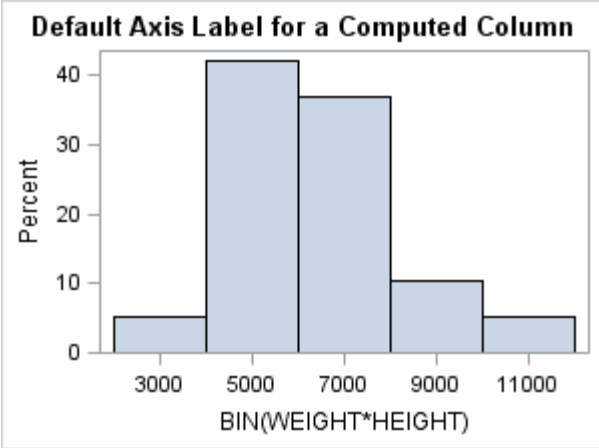
In the previous two examples, allowing the primary plot to determine the Y-axis label did not result in an appropriate label because a more generic label is needed. To achieve this, you must set the axis label yourself with the LABEL= option as shown in the following layout block.

```
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;
```

For computed columns, an axis label is manufactured from the input column(s) and the functional transformation that is applied to the input column(s). For example, you can use an EVAL expression to compute the product of WEIGHT and HEIGHT as a new column that can be used as a required argument as shown in the following example.

```
proc template;
  define statgraph axislabeldefault3;
    begingraph;
      entrytitle "Default Axis Label for a Computed Column";
      layout overlay;
        histogram eval(weight*height);
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.class template=axislabeldefault3;
run;
```

In this example, the manufactured label is BIN(WEIGHT*HEIGHT) as shown in the following figure.

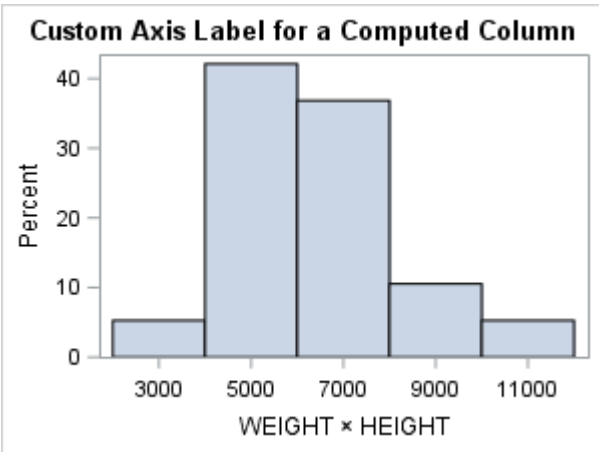


Overriding the Default Axis Label

You can use the axis option LABEL= to override the default label for an axis with your own label. For example, for a computed column, you can replace the manufactured label with a more appropriate label as shown in the following layout block.

```
layout overlay / xaxisopts=(label="WEIGHT (*ESC*){unicode '00D7'x} HEIGHT");
   histogram eval(weight*height);
endlayout;
```

Here is the output.

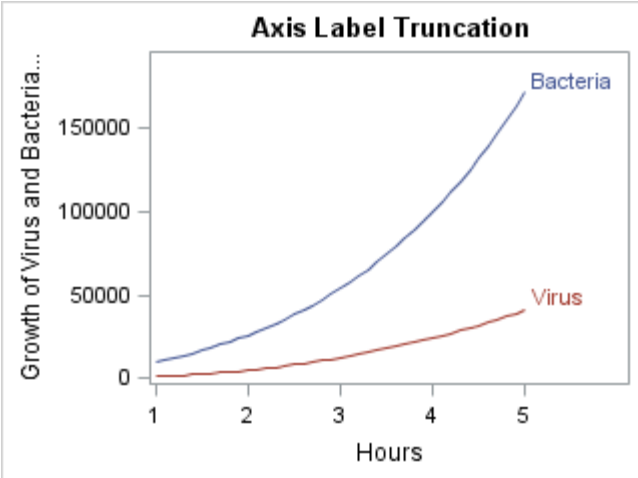


In this example, the manufactured label BIN(WEIGHT*HEIGHT) is replaced with WEIGHT x HEIGHT to denote the product of WEIGHT and HEIGHT.

Making Long Axis Labels Fit

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 does not 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. The following figure shows an example.



As a remedy for these situations, you can specify a shorter alternate label with the SHORTLABEL= option, or you can split the label into multiple lines by setting the LABELFITPOLICY= option to SPLIT or SPLITALWAYS.

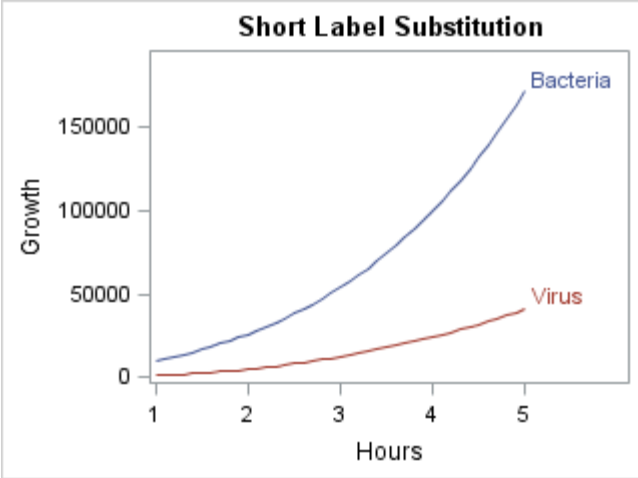
A short label is displayed whenever the default label or the LABEL= string does not fit and label splitting is not enabled. Here is an example.

Example Code 11.4 *Short Axis Label Example*

```
proc template;
  define statgraph shortaxislabel;
    begingraph;
      entrytitle "Short Label Substitution";
      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;
    endgraph;
  end;
run;

ods graphics / width=320px;
proc sgrender data=growth template=shortaxislabel;
run;
ods graphics / reset=width;
```

Note: See Example Code 11.3 on page 117.

Here is the output.



Because the specified label does not fit the allotted space, the short label is used instead.

Instead of a short label, you can enable label splitting, which automatically splits the axis label into multiple lines. To enable label splitting, set LABELFITPOLICY= to SPLIT or SPLITALWAYS. The SPLIT policy splits the label into multiple lines as needed to make it fit in the available space. The SPLITALWAYS policy always splits the label into multiple lines regardless of the available space. By default, the label is split on a blank space.

Note: The SHORTLABEL= option is ignored when LABELFITPOLICY= is set to SPLIT or SPLITALWAYS.

When LABELFITPOLICY= is set to SPLIT, by default, the label is split on a blank space only where a split is needed to make the label fit. When this option is set to SPLITALWAYS, the label is split on every occurrence of a blank space in the label. If you want to split the label on a character other than a blank space, use the LABELSPLITCHAR= option to specify one or more characters on which to split the label. The list must be specified as a quoted string with no space between the characters. The characters are case sensitive, and order is not significant. For example, to split a label on a blank space, a comma, or a semicolon, specify:

```
labelsplitchar=" ,;"
```

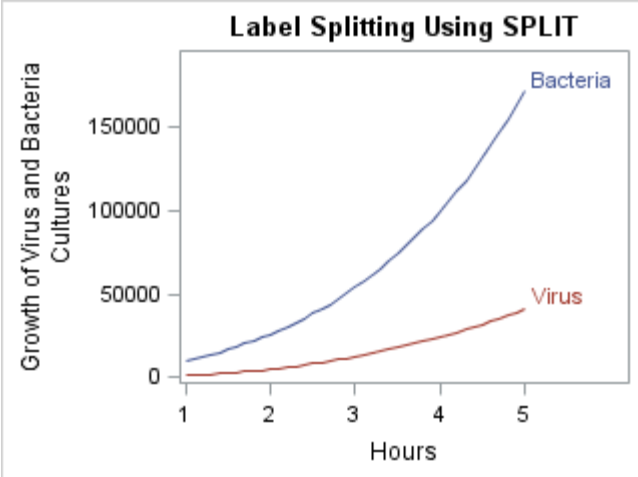
A split character on which a split occurs is removed from the displayed label by default. A split character on which a split does not occur remains in the displayed label. If you want all of the split characters to remain in the displayed label, use the LABELSPLITCHARDROP=FALSE option.

Each line of a split label is centered in the allotted space by default. You can use the LABELSPLITJUSTIFY= option to justify each line of the label left or right for the X and X2 axes, or top or bottom for the Y and Y2 axes.

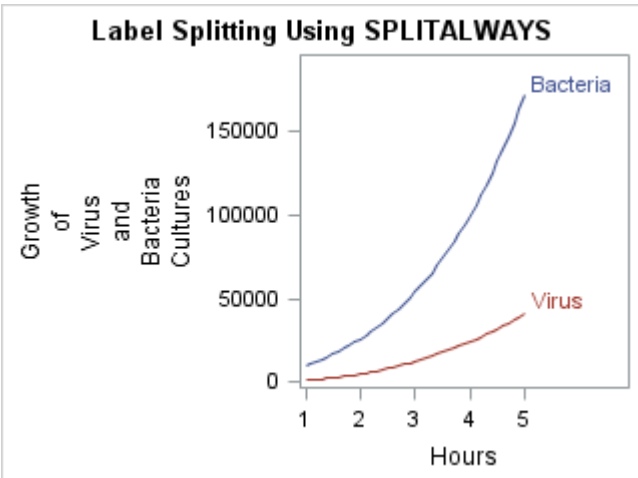
Here is the layout block in Example Code 11.4 on page 121 modified to use label splitting instead of a short label for the Y-axis label.

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

Here is example output.



In this example, the SPLIT policy is used, which splits the label on a blank space only where needed. In this case, only one split is needed after “and.” If the SPLITALWAYS policy is used instead, the label is split on every occurrence of a blank space as shown in the following figure.



Positioning the Axis Labels

By default, axis labels are centered in the axis area as shown in Figure 11.1 on page 119. You can use the LABELPOSITION= option to change the position of the axis labels for the OVERLAY, DATALATTICE, DATAPANEL, and LATTICE layouts. For the OVERLAY, DATALATTICE, and DATAPANEL layouts, you can specify one of the following values for LABELPOSITION=:

CENTER	centers the axis label in the axis area (default).
DATACENTER	for OVERLAY layouts, centers the axis label in the axis tick display area. For DATALATTICE and DATAPANEL layouts, this value repeats the axis label for each row or column and

	centers each label in the axis tick display area of its row or column.
TOP or BOTTOM	orients the label horizontally at the top or bottom of the axis area.
LEFT or RIGHT	positions the label to the left or right of the axis area.

For the LATTICE layout, you can specify one of the following values for LABELPOSITION=:

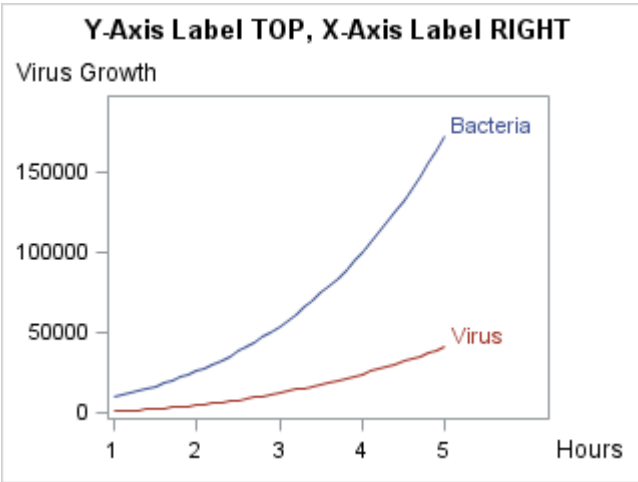
CENTER	centers the axis label in the axis area (default).
DATACENTER	centers each row or column axis label in its axis tick display area.

In the following example, the Y-axis label is positioned at the top of the axis area, and the X-axis label is positioned to the right of the axis area in an OVERLAY layout.

```
proc template;
  define statgraph axislabeldefault2;
    begingraph;
      entrytitle "Y-Axis Label TOP, X-Axis Label RIGHT";
      layout overlay / cycleattrs=true
        xaxisopts=(labelposition=right)
        yaxisopts=(labelposition=top);
        seriesplot x=Hours y=Bacteria/ curvelabel="Bacteria";
        seriesplot x=Hours y=Virus / curvelabel="Virus"
primary=true;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=growth template=axislabeldefault2;
run;
```

Note: See Example Code 11.3 on page 117.

Here is the output.



Using the LABELPOSITION= option can result in axis label collisions in some cases. For example, in the previous example, if LABELPOSITION=BOTTOM is used for the Y axis and LABELPOSITION=LEFT is used for the X axis, the axis labels collide. When you select a position, make sure that it does not collide with any other labels.

Axis Tick Values

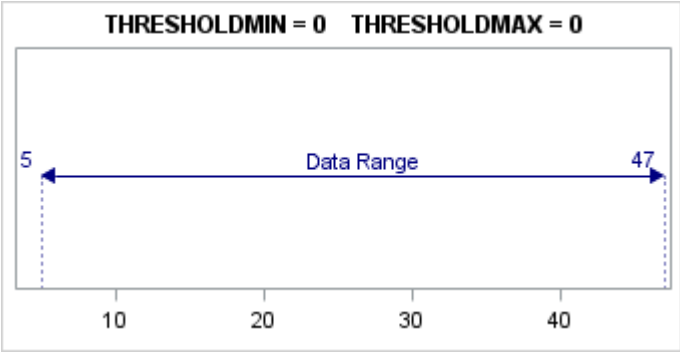
The tick values for LINEAR and TIME axes are calculated according to an internal algorithm that produces good tick values by default. This algorithm can be modified or bypassed with axis options. For examples, see “LINEAR Axes” on page 131, “Discrete Axes” on page 140, and “LOG Axes” on page 158.

By default, the font characteristics of the tick values are set by the current style, and the tick mark values progress along the axis from the least value to the highest value. You can set alternative font characteristics with the TICKVALUEATTRS= option. For more information, see “Axis Appearance Features Controlled by the Current Style” on page 112.

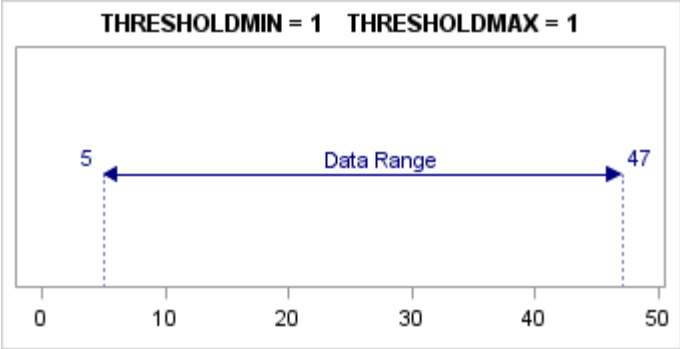
Note: For large tick values on a linear axis, you can specify a scale for the values to save space. See “Scaling the Tick Values on a Linear Axis” on page 134.

Axis Thresholds

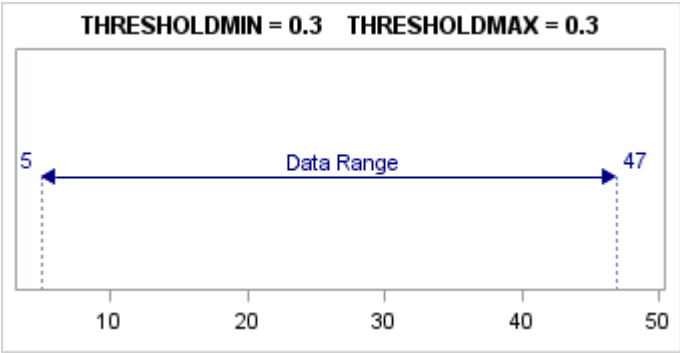
For LINEAR and LOG axes only, part of the default axis construction computes a small number of "good" tick values for the axis. This list might include "encompassing" tick values that go beyond the data range on both the lower or upper side of the axis. The THRESHOLDMIN= and THRESHOLDMAX= options of 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 are always 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 are always 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 is 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 is 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.

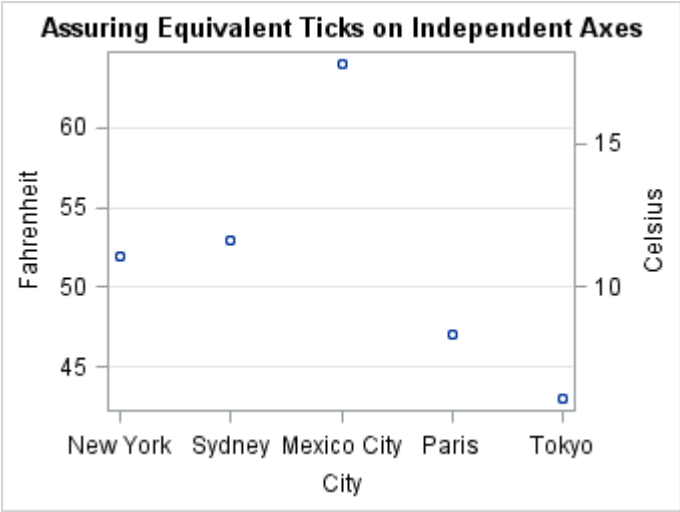
Thresholds are important when you want the Y and Y2 (or X and X2) axes ticks marks to be at the same locations on different scales. By preventing "encompassing" ticks from being drawn, you can ensure that the axis ranges for the two axes correctly align. The following example accepts the default minimum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

```
proc template;
  define statgraph axisThreshold1;
    begingraph;
      entrytitle "Assuring Equivalent Ticks on Independent Axes";
      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;
      endgraph;
    end;
  run;

  proc sgrender data=temps template=axisThreshold1;
  run;
```

Note: See Example Code 11.2 on page 105.

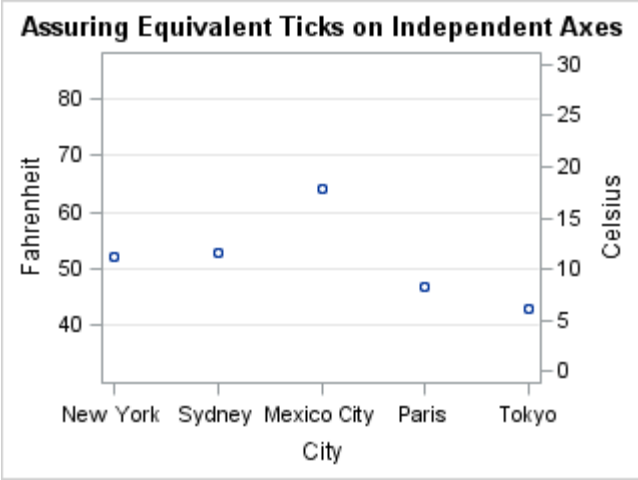
Here is the output. Notice that the five scatter points for each plot are superimposed exactly.



In the following layout block, 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;
```

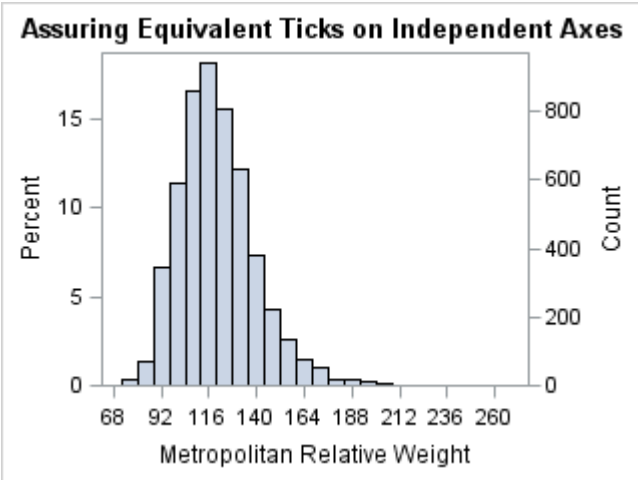
Here is example output.



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

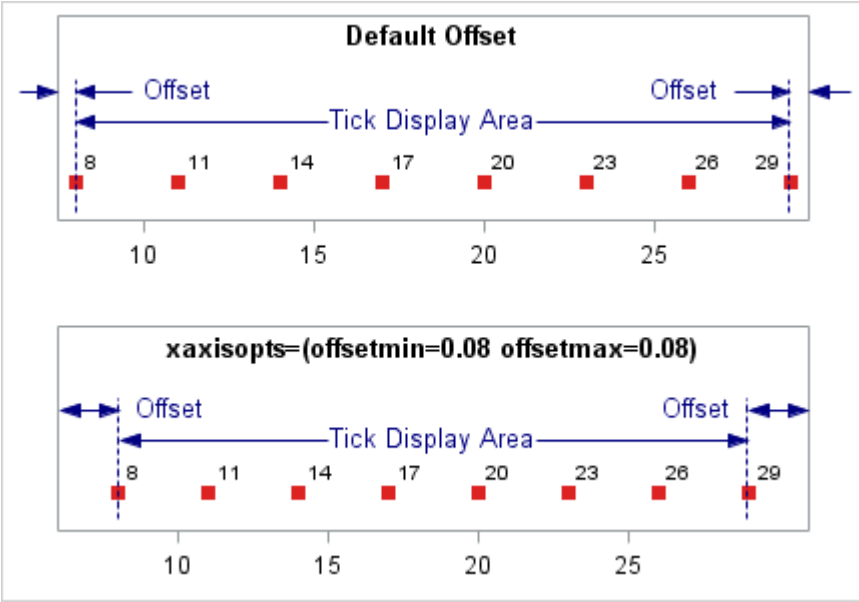
```
proc template;
  define statgraph overlayaxes.axisthreshold3;
    begingraph;
      entrytitle "Assuring Equivalent Ticks on Independent Axes";
      layout overlay /
        yaxisopts=(linearopts=(thresholdmin=0 thresholdmax=0))
        y2axisopts=(linearopts=(thresholdmin=0 thresholdmax=0));
        histogram mrw  / scale=percent;
        histogram mrw / yaxis=y2 scale=count;
      endlayout;
    endgraph;
   end;
run;

proc sgrender data=sashelp.heart template=overlayaxes.axisthreshold3;
run;
```

Here is the output.

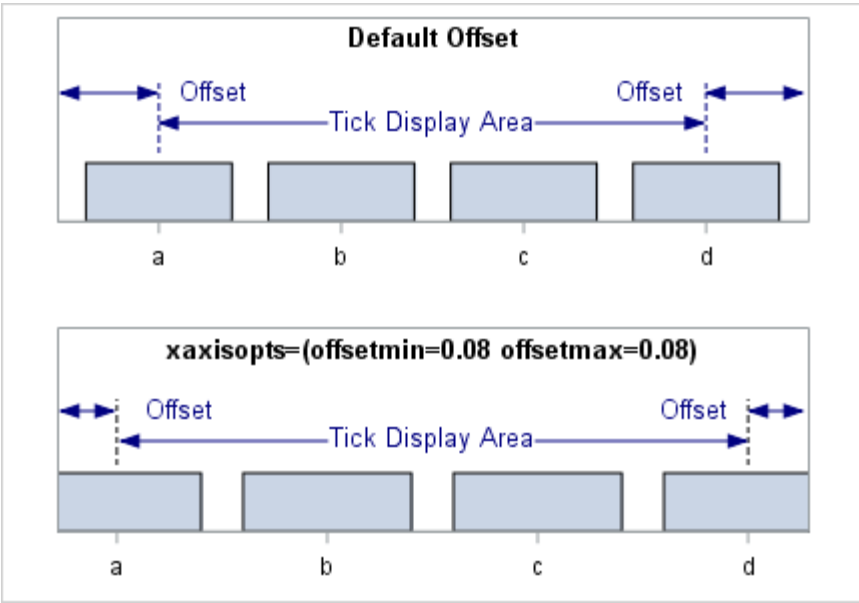


Axis Offsets

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 bar charts, offset space is added to ensure that the first and last box or bar is not clipped.



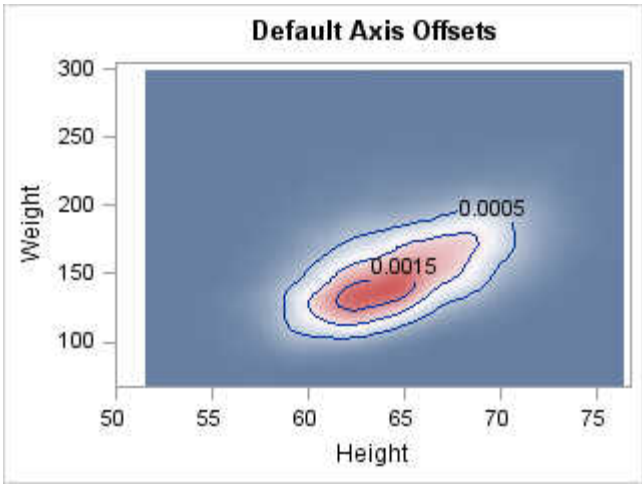
The OFFSETMIN= option on a layout statement controls the distance from the beginning of the axis to the first tick mark (or to the first minimum data value). The

OFFSETMAX= option controls the distance between last tick (or maximum data value) and the end of the axis. You can set both OFFSETMIN= and OFFSETMAX= to AUTO, AUTOCOMPRESS, or a numeric range. The default offset is AUTO. AUTO automatically provides enough offset to display markers and other graphical features. AUTOCOMPRESS provides enough offset to keep the minimum and maximum tick marks from extending beyond the axis length. The numeric range is 0–1. This range is used to calculate the offset as a percentage of the full axis length.

For some plots, the axis offsets are not desirable. To illustrate this, consider the following contour plot example.

```
proc template;
  define statgraph overlayaxes.axisoffsets1;
    begingraph;
      entrytitle "Default Axis Offsets";
      layout overlay;
        contourplotparm x=height y=weight z=density;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.gridded template=overlayaxes.axisoffsets1;
run;
```

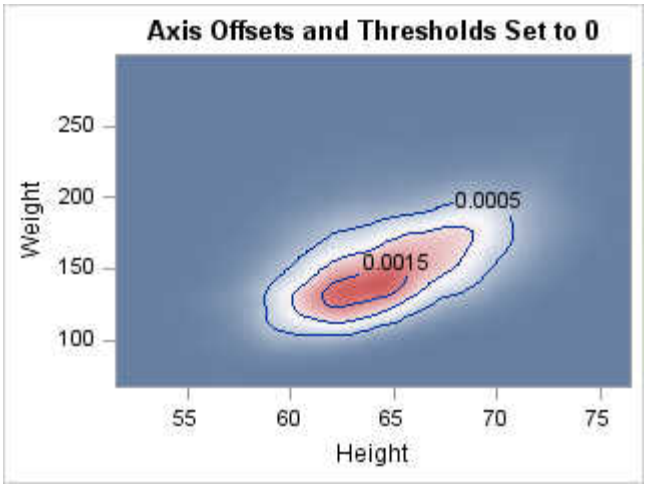
Here is the output.



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. To eliminate the "extra gaps at the ends of the axes, set axis offsets and thresholds to 0, as shown in the following layout block.

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

An offset is a value in the range 0–1 that represents a percentage of the length of the axis. Here is example output.



LINEAR Axes

Setting the Data Range and Tick Values on a Linear Axis

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 are used as long as those values are within the actual range of the data. To extend (or reduce) the axis data range, you can use the VIEWMIN= and VIEWMAX= suboptions of the LINEAROPTS= option. You can also use the TICKVALUEPRIORITY=TRUE option, which automatically extends the axis range to accommodate the values specified by the TICKVALUELIST= or TICKVALUESEQUENCE= option.

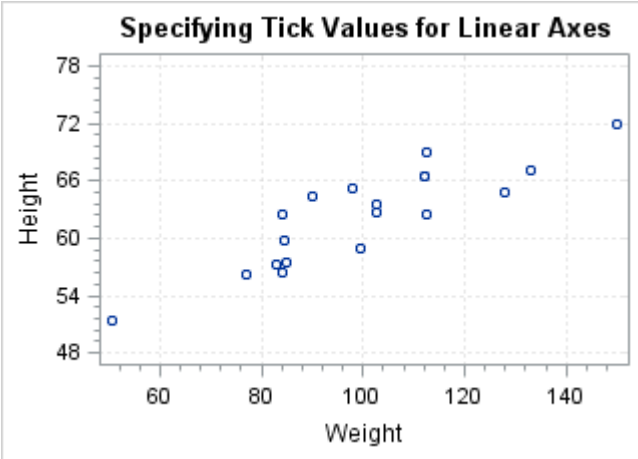
Here is an example that uses the TICKVALUELIST= option to specify the tick values. It uses the VIEWMIN= and VIEWMAX= options to extend the axis range to include the specified values.

Example Code 11.5 *TICKVALUELIST Example*

```
proc template;
  define statgraph heightchart;
    begingraph;
      entrytitle "Specifying Tick Values for Linear Axes";
      layout overlay /
        xaxisopts=(type=linear griddisplay=on
          gridattrs=(pattern=dot color=lightgray)
          linearopts=(minorticks=true))
        yaxisopts=(type=linear tickvaluehalign=left
          griddisplay=on gridattrs=(pattern=dot color=lightgray)
          linearopts=(viewmin=48 viewmax=78
          tickvaluelist=(48 54 60 66 72 78)
          minorticks=true));
        scatterplot x=weight y=height;
```

```
      endlayout;
    endgraph;
  end;
run;

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

Here is the output.



With this method, if you change the values specified by the TICKVALUELIST= option, you might have to change the VIEWMIN= and VIEWMAX= option values to accommodate the new values. If you use the TICKVALUEPRIORITY=TRUE option instead of the VIEWMIN= and VIEWMAX= options, the axis range is adjusted automatically.

When you use the TICKVALUELIST= option, you can use the TICKVALUEDISPLAYLIST= option to specify string values to display on the axis instead of the specified tick values. You must specify the TICKVALUELIST= option in order to use the TICKVALUEDISPLAYLIST= option. The string values specified by the TICKVALUEDISPLAYLIST= option map one-to-one positionally to the values specified by the TICKVALUELIST=. Thus, the number of string values that you specify in the TICKVALUEDISPLAYLIST= option must exactly match the number of values that you specify in the TICKVALUELIST= option. If you specify too many display values, the excess values are ignored. If you specify too few, blank tick values result on the axis.

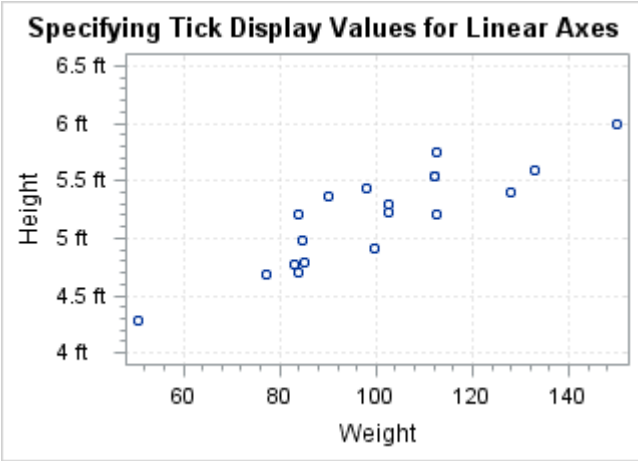
Here is the Example Code 11.5 on page 131 modified to use the TICKVALUEDISPLAYLIST= option to display the height values in feet on the Y axis.

```
proc template;
  define statgraph heightchart;
    begingraph;
      entrytitle "Specifying Tick Display Values for Linear Axes";
      layout overlay /
        xaxisopts=(type=linear griddisplay=on
          gridattrs=(pattern=dot color=lightgray)
          linearopts=(minorticks=true))
        yaxisopts=(type=linear tickvaluehalign=left
          griddisplay=on gridattrs=(pattern=dot color=lightgray)
          linearopts=(tickvaluepriority=true minorticks=true
            tickvaluelist=(48 54 60 66 72 78)
            tickdisplaylist=("4 ft" "4.5 ft" "5 ft" "5.5 ft"
```

```
                "6 ft" "6.5 ft")));
          scatterplot x=weight y=height;
        endlayout;
      endgraph;
    end;
  run;

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

Notice that the TICKVALUELIST= and TICKVALUEDISPLAYLIST= options specify the same number of values. Notice also that the values map positionally.

Here is the output.



Formatting the Tick Values on a Linear Axis

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 the formats that are not supported, see Appendix 6, “SAS Formats Not Supported,” on page 703.

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

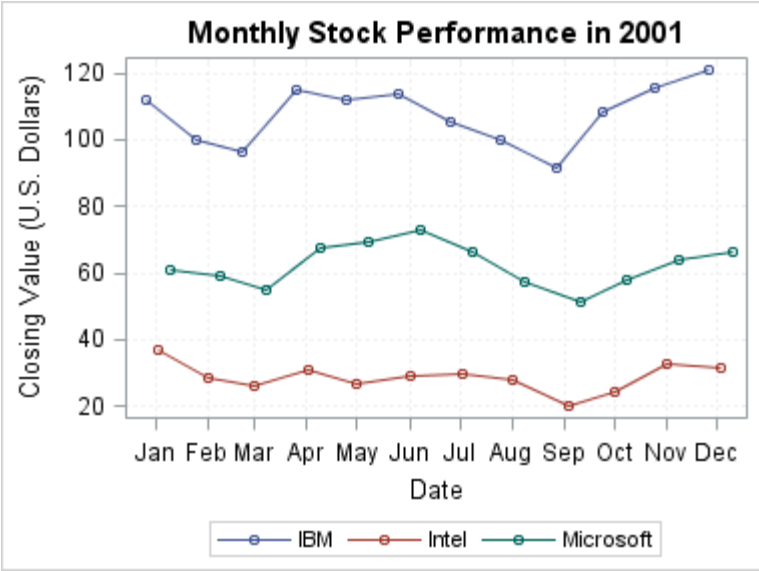
```
linearopts=(tickvalueFormat=data)
```

There are special options to control tick values. INTEGER=TRUE calculates good integers to use as tick values given the range of the data.

Here is an example that overrides the dollar format on column Close in Sashelp.Stocks with a decimal format.

```
proc template;
  define statgraph barchart;
   begingraph;
    entrytitle "Monthly Stock Performance in 2001";
    layout overlay /
      xaxisopts=(griddisplay=on gridattrs=(pattern=dot)
        timeopts=(tickvalueformat=monname3. viewmax='01DEC2001'd))
      yaxisopts=(griddisplay=on gridattrs=(pattern=dot)
        label="Closing Value (U.S. Dollars)"
        linearopts=(tickvalueformat=5.0));
      seriesplot x=date y=close / name="scatter"
        display=all group=stock groupdisplay=cluster;
      discretelegend "scatter";
    endlayout;
   endgraph;
  end;

proc sgrender data=sashelp.stocks template=barchart;
  where year(date) = 2001;
run;
```

Here is the output.



Scaling the Tick Values on a Linear Axis

For large tick values on linear axes, you can specify a tick value scale in order to save space. This feature applies to linear axes only. You can use a named scale such as millions, billions, or trillions, or a scientific notation scale expressed as 10^*n*. To extract a scale automatically from the tick values, include the

EXTRACTSCALE=TRUE option in the TICKVALUEFORMAT= option list. By default, for tick values of 999 trillion or less, a named scale is used. For values over 999 trillion, a scientific notation scale is used. For large tick values, the scale factor is set to ensure that the absolute value of the largest value is greater than 1. For small fractional tick values, the scale factor is set to ensure that the absolute value of the smallest value is greater than 1.

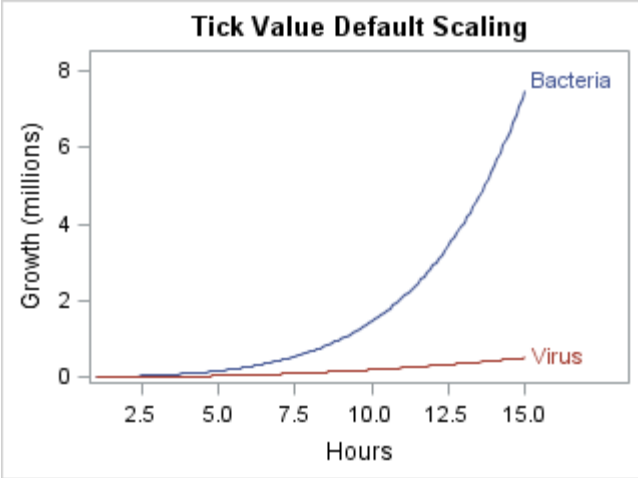
Note: The scale that is extracted by the EXTRACTSCALE= option is derived from the English locale.

To specify a scientific notation scale for values less than 999 trillion, include the EXTRACTSCALETYPE=SCIENTIFIC option in the TICKVALUEFORMAT= option list. The scale that is used is appended to the axis label.

Here is an example that specifies the default scale for the growth example that is shown in "Making Long Axis Labels Fit" on page 120 for 15 hours of growth data.

```
data growth15hr;
  do Hours=1 to 15 by .1;
    Growth = 10**hours;
    Bacteria = 1000*10**( sqrt(Hours ));
    Virus = 1000*10**(log(hours));
    label bacteria = "Bacteria Growth" virus="Virus Growth";
    output;
  end;
run;

proc template;
  define statgraph axisdefaultscale;
    begingraph;
      entrytitle "Tick Value Default Scaling";
      layout overlay / cycleattrs=true
        yaxisopts=(label="Growth of Virus and Bacteria Cultures"
          shortlabel="Growth"
          linearopts=(tickvalueformat=(extractscale=true)));
        seriesplot x=Hours y=Bacteria/ curvelabel="Bacteria";
        seriesplot x=Hours y=Virus / curvelabel="Virus";
      endlayout;
    endgraph;
  end;
run;

ods graphics / width=320px;
proc sgrender data=growth15hr template=axisdefaultscale;
run;
ods graphics / reset=width;
```

Here is the output.



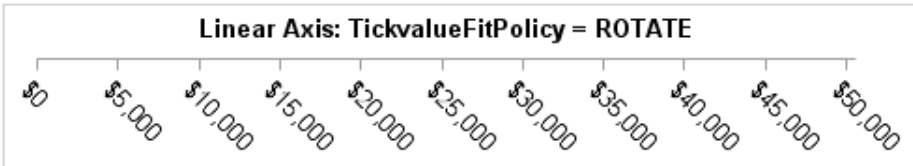
Notice that the tick value scale name is automatically appended to the axis label, which is the short label in this case.

Fitting the Tick Values on a Linear Axis

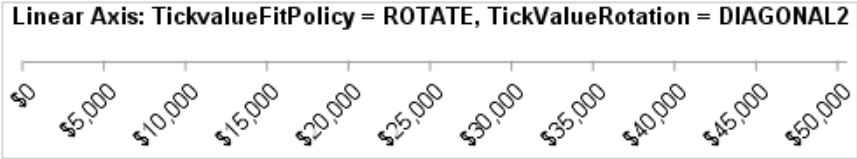
When the tick value text becomes too crowded, the tick values that are displayed on the axes are scaled automatically for fit. For example, the axis below comfortably shows eleven tick values:

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

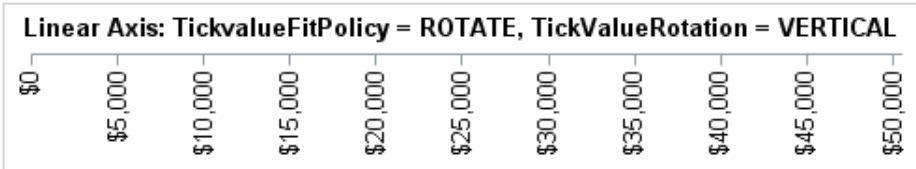
For the X and X2 axes only, you can set TICKVALUEFITPOLICY=ROTATE or TICKVALUEFITPOLICY=ROTATEALWAYS, which angles the tick value text 45 degrees by default:



Starting with SAS 9.4M5, you can set TICKVALUEROTATION=DIAGONAL2 to angle the tick value text –45 degrees instead:



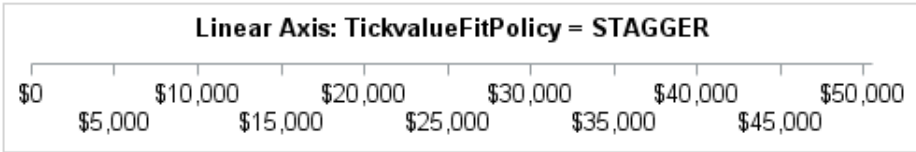
When you set TICKVALUEFITPOLICY=ROTATE or TICKVALUEFITPOLICY=ROTATEALWAYS, you can set TICKVALUEROTATION=VERTICAL to rotate the tick value text vertically instead of diagonally:



Note: When the tick values are rotated vertically, they read from bottom up as shown.

Note: The ROTATE and ROTATEALWAYS policies, and the TICKVALUEROTATION= option apply to the X and X2 axes only.

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



For the X and X2 axes, 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. For the Y and Y2 axes, you can set TICKVALUEFITPOLICY to NONE or THIN. For information about the tick value fit policies that can be used with a linear axis in each of the applicable layouts, see “Tick Value Fit Policy Applicability Matrix” on page 699.

Creating a Broken Linear Axis

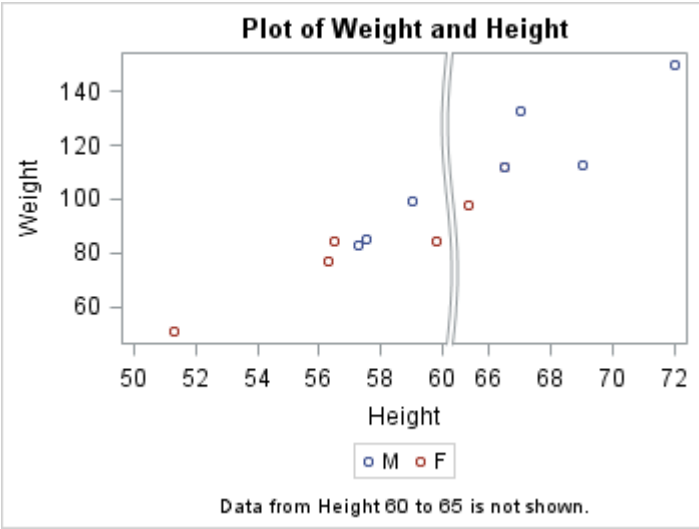
In some cases, you might want to remove portions of a plot where the data is sparse or is of little interest. Starting with SAS 9.4M1, you can use the INCLUDERANGES= option to specify ranges of data that you want to include on a linear axis. All data and axis tick values that fall outside of the ranges that you specify are clipped from

the output. At each point where a gap in the axis exists, break lines are drawn perpendicular to the axis to indicate the break. The axis tick value sequence also reflects the gaps.

Here is an example of a weight and height plot in which the INCLUDERANGES= option is used to remove the data for heights between 60 and 65 inches.

```
proc template;
  define statgraph scatterplot;
    begingraph;
      entrytitle "Plot of Weight and Height";
      entryfootnote "Data from Height 60 to 65 is not shown.";
      layout overlay / xaxisopts=(
        linearopts=(includeranges=(50-60 65-72)));
        scatterplot x=height y=weight / name="scatter" group=sex;
        discretelegend "scatter";
      endlayout;
    endgraph;
  end;
run;

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

The INCLUDERANGES= option specifies ranges as a space-separated list of *start–end* value pairs. In this example, ranges 50–60 and 65–72 are specified, which creates a break between 60 and 65. Here is the output.



Notice the break lines and the gap in the X-axis tick value sequence where the break occurs. All data markers that fall between 60 and 65 are clipped from the output.

The following restrictions apply to broken axes:

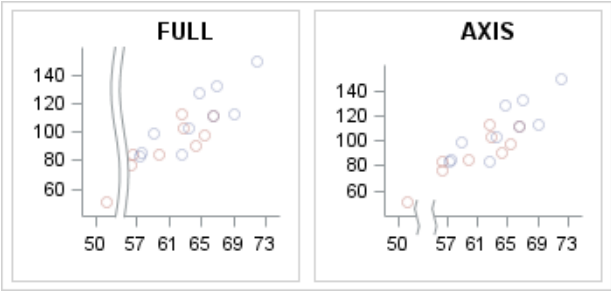
- A broken axis is valid for linear and time axes in an OVERLAY layout only.
- Only one axis can be broken.
- Each range specified must be nonzero. A zero range such as 12–12 is considered invalid.
- When plots are associated with the X and X2 axes or with the Y and Y2 axes, neither axis can be broken.

- When an axis is broken, data tips and selectable graphical elements are not supported.

For more information about the INCLUDERANGES= option, see "Options for Linear Axes Only" in *SAS Graph Template Language: Reference*.

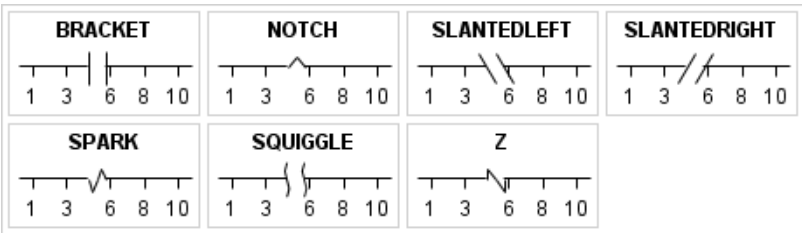
Starting with SAS 9.4M3, you can change the default break indicator. Rather than break lines that span the entire data display, you can specify a break symbol that appears on the axis line only. The BEGINGRAPH statement AXISBREAKTYPE= option enables you to specify the break type. Set it to FULL to specify full-display break lines or set it to AXIS to specify a break symbol only on the axis line. The following figure shows an example of each.

Figure 11.2 *Axis Break Types FULL and AXIS*



You can specify AXIS only when the axis line or the plot wall outline is displayed. Otherwise, AXIS is ignored and FULL is used instead. If you specify AXIS when the secondary axis line or the plot wall outline is displayed, the break symbol is displayed on both the primary axis and the secondary axis. Otherwise, the break symbol is displayed only on the primary axis as shown in Figure 11.2 on page 139.

By default, a pair of squiggle lines is used to indicate the breakpoint on the axis line. You can use the BEGINGRAPH statement AXISBREAKSYMBOL= option to specify one of the break symbols shown in the following figure.



For more information about the BEGINGRAPH statement AXISBREAKTYPE= and AXISBREAKSYMBOL= options, see "BEGINGRAPH" in *SAS Graph Template Language: Reference*.

Discrete Axes

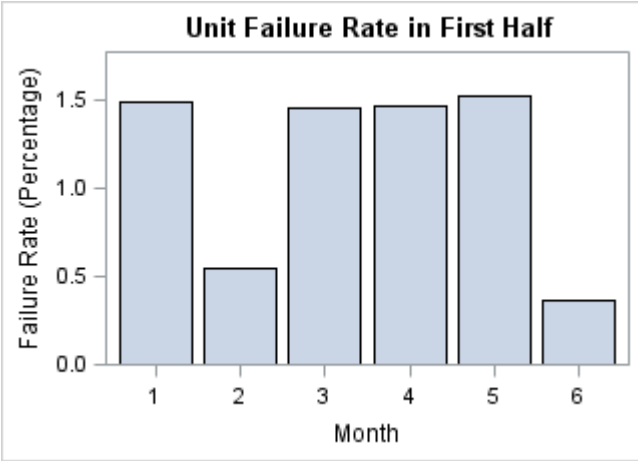
Setting the Tick Values on a Discrete Axis

On a discrete axis, the data values that are represented in the graph are displayed by default as tick values on the axis. If the graph data is a SAS data set, character tick values are arranged along the axis in the order in which they are used in the graph, and numeric values are arranged in ascending order. If the plot data is a CAS in-memory table, both character and numeric tick values are arranged in ascending order along the axis. In some cases, the resulting tick-value order might not be desirable, especially if the graph consists of multiple plots or if the data contains missing values. In that case, you can include the TICKVALUELIST=(*value-list*) option in the DISCRETEOPTS= option to set the order of the axis tick values. You can also use the TICKVALUELIST= option to subset the tick values or to add values that are not included in the data. The values in value-list must correspond to the formatted values of the data.

Here is an example of a template that uses the TICKVALUELIST= option to display the first six months of annual failure data.

```
/* Create a test data set of annual failure data */
data failrate;
  do month=1 to 12 by 1;
    failures=2*ranuni(12345);
    label failures="Failure Rate (Percentage)"
      month="Month";
    output;
  end;
run;

/* Create a template to display the first six months of data */
proc template;
  define statgraph firsthalf;
    begingraph;
      entrytitle "Unit Failure Rate in First Half";
      layout overlay /
        xaxisopts=(
        discreteopts=(tickvaluelist=("1" "2" "3" "4" "5" "6")));
        barchart category=month response=failures;
      endlayout;
    endgraph;
  end;
run;

/* Plot the data */
proc sgrender data=failrate template=firsthalf;
run;
```

Here is the output.



Note: This graph can also be accomplished by subsetting the data.

By default, the tick marks appear on the midpoints. You can include the TICKTYPE=INBETWEEN option in the DISCRETEOPTS= option list to position the tick marks between the midpoints instead of on the midpoints.

Formatting the Tick Values on a Discrete Axis

Starting with SAS 9.4M3, you can use the TICKVALUEFORMAT= option to apply a character format to the original tick values on a discrete axis. Numeric formats are not supported. The original tick values can be a list of formatted values from the data column or a list of tick values from the TICKVALUELIST= option. Also starting with SAS 9.4M3, ODS Graphics supports Unicode values in user-defined character and numeric formats.

You can use a user-defined character format to modify the tick values on a discrete axis. For example, you can use abbreviations or special symbols to shorten long tick values. You can also use a user-defined character format to duplicate tick values by consolidating two or more tick values into one. Here is a simple example that uses a character format to duplicate tick values Y and N on the category axis of a horizontal bar chart of survey results. The survey measures support for a fictitious zoning ordinance, ZOR327A. The results are broken out by age group. This example also demonstrates how to use Unicode values in a user-defined format to help shorten tick values. Here is the code for this example.

```
/* Create the survey data set */
data ZOR327A_survey;
  input category $1-17 result;
datalines;
Age 20 to 29:    0
Yes1             0.41
No1              0.59
Age 30 to 39:    0
Yes2             0.53
No2              0.47
Age 40 to 49:    0
```

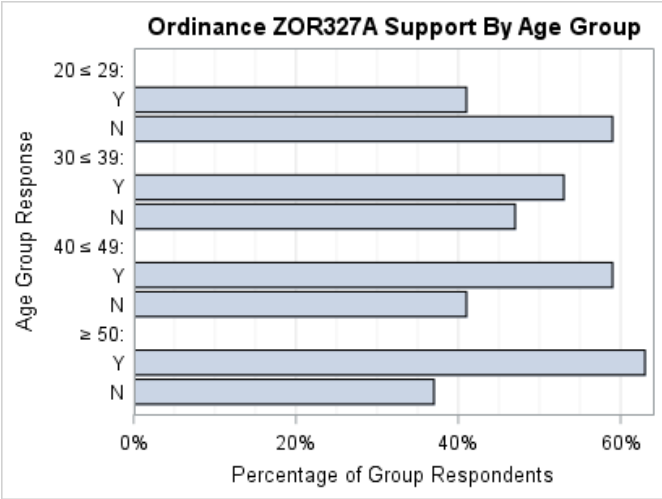
```
Yes3              0.59
No3               0.41
Age 50 and Over:  0
Yes4              0.63
No4               0.37
;
run;

/* Create a format for the category tick values */
proc format;
  value $catTickValues
    "Yes1", "Yes2", "Yes3", "Yes4" = "Y"
    "No1", "No2", "No3", "No4" = "N"
    "Age 20 to 29:" = "20 (*ESC*){unicode '2264'x} 29:"
    "Age 30 to 39:" = "30 (*ESC*){unicode '2264'x} 39:"
    "Age 40 to 49:" = "40 (*ESC*){unicode '2264'x} 49:"
    "Age 50 and Over:" = "(*ESC*){unicode '2265'x} 50:";
run;

/* Define the template for the graph */
proc template;
  define statgraph barchart;
    begingraph;
      entrytitle "Ordinance ZOR327A Support By Age Group";
      layout overlay /
        xaxisopts=(label="Percentage of Group Respondents"
          griddisplay=on linearopts=(minorgrid=true minortickcount=3))
        yaxisopts=(display=(label tickvalues) reverse=true
          label="Age Group Response"
          discreteopts=(tickvalueformat=$catTickValues.));
        barchartparm category=category response=result /
          orient=horizontal;
      endlayout;
    endgraph;
  end;
run;

/* Render the graph */
proc sgrender data=ZOR327A_survey template=barchart;
  format result percent.;
run;
```

In the plot data, in order to plot the data properly, each age group must have a unique category for the Yes and No statistics. Yes*n* and No*n* are used in this example, where *n* is unique for each age group. The $catTickValues format is used to display a Y and N tick value for each age group and to shorten the age group values. To duplicate the Y and N tick values, format $catTickValues consolidates all of the Yes*n* category values into tick value Y, and all of the No*n* category values into tick value N. To help shorten the age-group tick values, format $catTickValues uses the Unicode less-than-or-equal-to and greater-than-or-equal-to symbols. Notice that (*ESC*) is used to escape each Unicode value. To specify a Unicode value in a user-defined format, you must use the (*ESC*) escape sequence as shown in this example. You cannot use a user-defined ODS escape character in this case.

Format $catTickValues is applied to the chart category axis tick values using the TICKVALUEFORMAT= axis option.

Note: Applying a format that duplicates values to a character column might produce unexpected results. To use such a format to duplicate axis tick values, specify the format in the TICKVALUEFORMAT= option for the axis.

Here is the result.



Fitting Tick Values on a Discrete Axis

To avoid tick-value collisions, a discrete axis uses several of the fit policies that a linear axis uses. These policies include THIN, ROTATE, ROTATEALWAYS, STAGGER, ROTATETHIN, STAGGERTHIN, and STAGGERROTATE. For information about these policies, see "Fitting the Tick Values on a Linear Axis" on page 136. A discrete axis supports additional fit policies that enable you to do the following:

- (starting with SAS 9.4M5) display the tick values vertically as stacked letters with STACKALWAYS. See "Displaying the Discrete Axis Tick Values Vertically" on page 144.
- extract the axis values into a legend with the EXTRACT or EXTRACTALWAYS fit policy. See "Extracting Discrete Axis Tick Values into a Legend" on page 144.
- split the axis values into two or more lines with the SPLIT or SPLITALWAYS fit policy. See "Splitting the Discrete Axis Tick Values" on page 146.
- truncate the axis values with the TRUNCATE fit policy. See "Truncating the Discrete Axis Tick Values" on page 147.

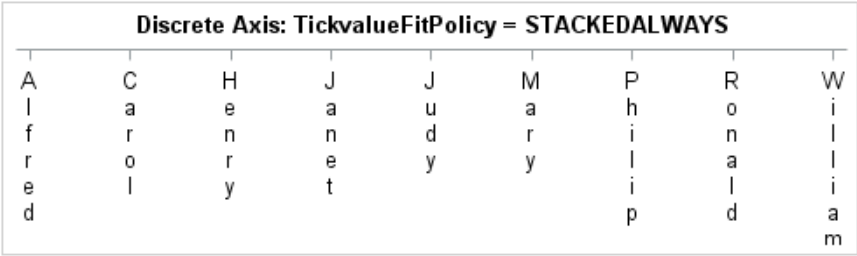
A discrete axis also supports the following additional compound fit policies:

ROTATEALWAYSDROP	SPLITTHIN	TRUNCATESTAGGER
SPLITALWAYSTHIN	STAGGERTRUNCATE	TRUNCATETHIN
SPLITROTATE	TRUNCATEROTATE	

Starting with SAS 9.4M5, STACKEDALWAYSTHIN is also supported.

Displaying the Discrete Axis Tick Values Vertically

Starting with SAS 9.4M5, you can set TICKVALUEFITPOLICY=STACKEDALWAYS to display the tick values on the X or X2 axis vertically as stacked letters. You can use this policy as an alternative to ROTATE with TICKVALUEROTATION=VERTICAL, which rotates the tick values 90-degrees. The following figure shows an example.



If you also require tick-value thinning, use STACKEDALWAYSTHIN instead.

Extracting Discrete Axis Tick Values into a Legend

For a large number of tick values, you can set TICKVALUEFITPOLICY=EXTRACT or TICKVALUEFITPOLICY=EXTRACTALWAYS to move the axis values to an AXISLEGEND. In that case, on the axis, the values are replaced with consecutive integers, and an AXISLEGEND is added that cross references the integer values on the axis to the actual axis values.

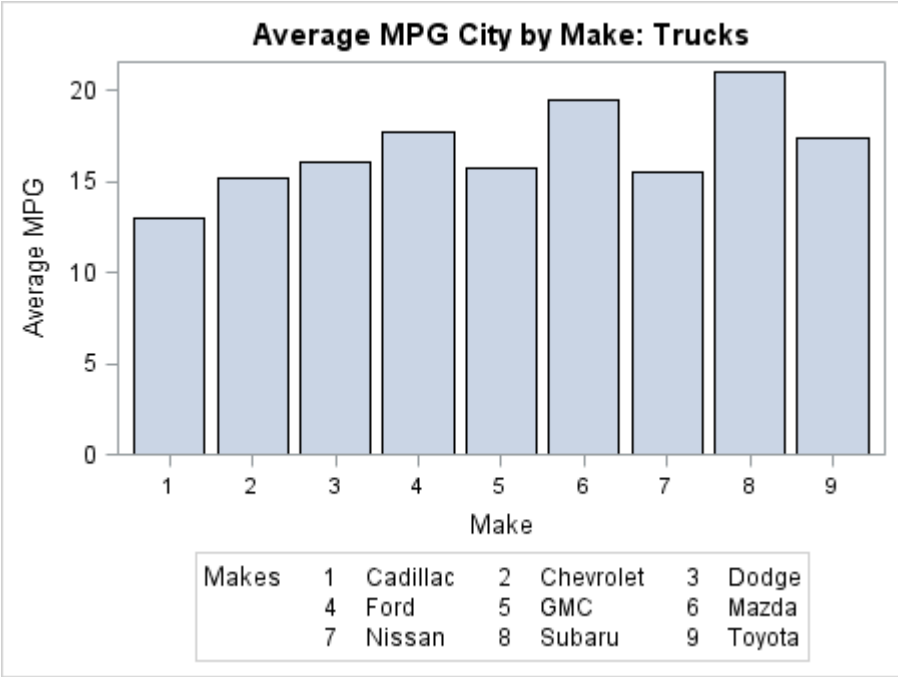
Note: If the axis type is not DISCRETE, the TICKVALUEFITPOLICY=EXTRACT and TICKVALUEFITPOLICY=EXTRACTALWAYS options are ignored.

When you set TICKVALUEFITPOLICY to EXTRACT or EXTRACTALWAYS, you must also include the NAME= option in your axis option list and an AXISLEGEND "*name*" statement in your layout block. In the AXISLEGEND statement, you must set *name* to the value that you specified with the NAME= option in the axis options list. Here is an example.

```
proc template;
  define statgraph discretefitpolicyextract;
    begingraph / border=false designwidth=450px designheight=400px;
      entrytitle "Average MPG City by Make: Trucks";
      Layout overlay /
        yaxisopts=(label="Average MPG")
        xaxisopts=(label="Make"
          name="axisvalues"
```

```
          tickvalueattrs=(size=9pt)
          discreteopts=(tickvaluefitpolicy=extract));
        barchart category=make response=mpg_city/orient=vertical
          stat=mean;
        axislegend "axisvalues" / pad=(top=5 bottom=5) title="Makes";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.cars template=discretefitpolicyextract;
  where type="Truck";
run;
```

Here is the output.



In this example, the name AXISVALUES is used to associate the AXISLEGEND statement with the X axis. The graph size is set to 450 pixels wide by 400 pixels high. Since there is not enough room on the X axis to fit the make names, the names are extracted to an axis legend as shown.

In most cases, the EXTRACT policy moves the axis values to an axis legend only if a collision occurs. If no collision occurs, the values are displayed on the axis in the normal manner. In this example, if you increase the width of the graph to 640 pixels and rerun the program, the legend disappears. In contrast, the EXTRACTALWAYS policy moves the axis values to a legend regardless of whether a collision occurs.

Note: With the exception of TICKVALUEFITPOLICY=EXTRACTALWAYS, the TICKVALUEFITPOLICY= is never applied unless a tick value collision situation is present. That is, you cannot force tick values to be rotated, staggered, or moved to an axis legend if there is no collision situation.

For information about the tick value fit policies that can be used with a discrete axis in each of the applicable layouts, see "Tick Value Fit Policy Applicability Matrix" on page 699.

Splitting the Discrete Axis Tick Values

For long tick values, you can set the TICKVALUEFITPOLICY= option to SPLIT. The SPLIT option splits the tick values to make them fit the available space. You can set this option to SPLITALWAYS to always split the tick values regardless of the available space. For the X and X2 axes, the amount of available space is the width allocated to each midpoint value. For the Y and Y2 axes, it is the width allocated to the axis.

By default, the tick values are split at a blank space that occurs where a split is needed. The value is split until the label is successfully fitted or no more blank spaces exist in the value string. Each blank space on which a split occurs is dropped from the displayed value. Blank spaces on which a split does not occur are retained in the displayed value. Each line of the value is centered in the available space.

If you want to split the values on a character other than a blank space or on more than one character, you can use the TICKVALUESPLITCHAR= option to specify a list of one or more split characters. You must specify the list as a quoted string with no space between the characters. The characters are case sensitive, and order is not significant. For example, to split a tick value on a blank space, a comma, or a semicolon, specify:

```
tickvaluesplitchar=" ,;"
```

When you specify multiple characters, each character in the list is treated as a separate split character unless they appear consecutively in the tick value. In that case, all of the characters together are treated as a single split character.

If you want to change the justification of each split line from CENTER, you can use the TICKVALUESPLITJUSTIFY= option to specify LEFT or RIGHT instead.

For example, consider the following axis values:

Davidson, Richard
Johnston, Miranda
McMillian, Joseph
Robertson, Mary Ann
Stenovich, Timothy

Notice that the last and first names are separated by a comma followed by a space. This example will make the following changes:

- Split the last and first names into two lines.
- Left-justify both lines.
- Retain the comma between the last and first names.
- Discard the space that occurs after the comma.

You can use the TICKVALUEFITPOLICY=SPLIT option to split the values on the space and discard the space by default. However, to left-justify the names, you must specify the TICKVALUESPLITJUSTIFY=LEFT option. The following figure shows the result.

Notice that the value Robertson, Mary Ann splits only once even though there are two occurrences of the split character in the value. By default, a split occurs on a split character only if a split is needed at that character to fit the label. In this case, because a split is not needed at the second blank character, it remains in the displayed value. If you want to force a split at every occurrence of a split character, use the TICKVALUEFITPOLICY=SPLITALWAYS option. In that case, the value Robertson, Mary Ann, splits into three left-justified lines.

If you want to keep the split characters in the tick values, use the TICKVALUESPLITCHARDROP=FALSE option. In that case, where each split occurs, the split character remains as the last character in the current line, and the characters that follow wrap to the new line.

The compound policies SPLITTHIN and SPLITALWAYSTHIN apply the split policy first, and then apply the thin policy if the split policy does not work.

Truncating the Discrete Axis Tick Values

You can set TICKVALUEFITPOLICY=TRUNCATE to shorten the tick values. Axis values that are greater than 12 characters in length are truncated to 12 characters. Ellipses are added to the end of the truncated values to indicate that truncation has occurred. When there is sufficient room on the axis to fit the full axis values, no truncation occurs. For example, if you decide to truncate the values from the previous example rather than split them. If you specify the TICKVALUEFITPOLICY=TRUNCATE option and remove the TICKVALUESPLITJUSTIFY=LEFT option, the values are truncated as shown in the following figure.

The compound policies STAGGERTRUNCATE, TRUNCATEROTATE, TRUNCATESTAGGER, and TRUNCATETHIN apply the first fit policy, and then apply the second fit policy if the first policy does not work.

Setting Alternating Wall Color Bands for Discrete Intervals

For a discrete axis, you can use the COLORBANDS= option in your DISCRETEOPTS= option list to specify alternating wall color bands for each of the

discrete axis intervals. The alternating bands can help improve the readability of complex plots. You can specify attributes for the odd or even bands. The odd bands begin with the first axis interval. The even bands begin with the second interval.

Note: If you specify the COLORBANDS option for more than one axis, such as both the X and Y axes, in an overlay, you might get unexpected results.

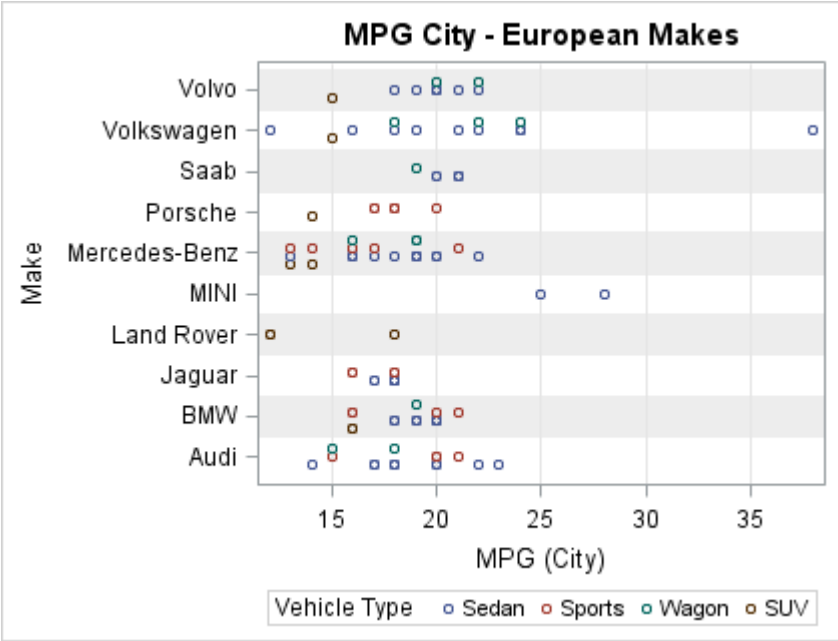
Include the COLORBANDSATTRS= option in your DISCRETEOPTS= option list to specify the fill color and transparency of the bars. You can specify either a style element or a list of fill options. If you specify fill options, you must enclose the options in parenthesis.

TIP You can add a border around the color bands by including the TICKTYPE=INBETWEEN option in the DISCRETEOPTS= option list and the GRIDDISPLAY=ON and GRIDATTRS= options in the XAXISOPTIONS= or YAXISOPTIONS= options list.

Here is an example that adds alternating light-gray color bands to the discrete Y axis of a plot. The bands begin on the second (even) interval.

```
proc template;
  define statgraph colorbands;
    begingraph;
      entrytitle "MPG City - European Makes";
      layout overlay /
        xaxisopts=(griddisplay=on)
        yaxisopts=(type=discrete offsetmin=0.07 offsetmax=0.07
          discreteopts=(colorbands=EVEN
          colorbandsattrs=(transparency=0.6 color=lightgray)));
        scatterplot y=make x=mpg_city / name="sp"
          group=type groupdisplay=cluster;
        discretelegend "sp" / title="Vehicle Type";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.cars template=colorbands;
  where origin="Europe";
run;
```

Here is the output.

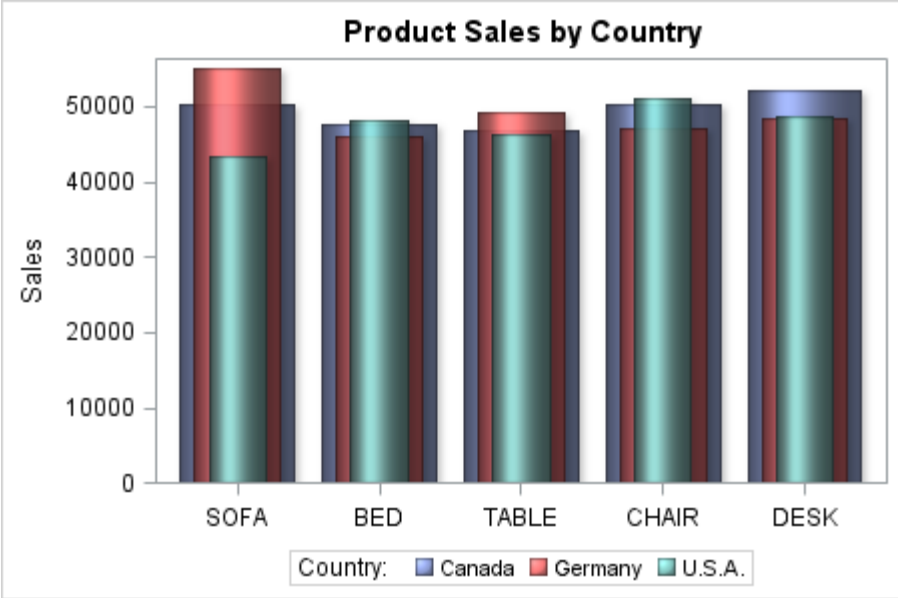


Offsetting Graph Elements from the Category Midpoint

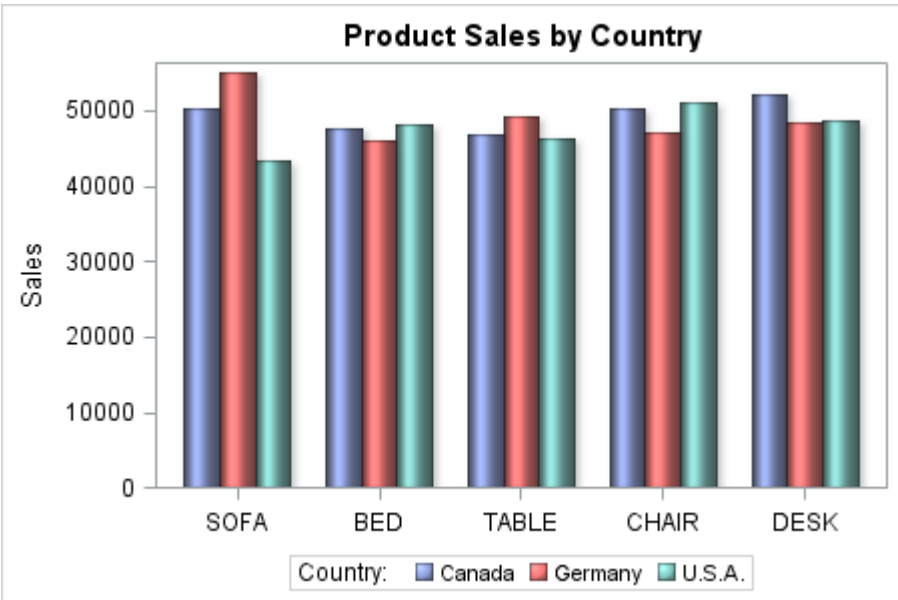
When you overlay plots that support discrete data, you can use the DISCRETEOFFSET= option to offset the graph elements from the midpoints on the discrete axis. Plots that support discrete data on one axis include:

BARCHART	BOXPLOTPARM	SCATTERPLOT
BARCHARTPARM	SERIESPLOT	DROPLINE
BOXPLOT	STEPPLOT	

By default, the graph elements, such as bars or markers, are positioned on the midpoint tick marks as shown in the following example. In this example, three bar charts are drawn using transparency and varying bar widths so that you can see how the bars are overlaid. The DATATRANSPARENCY= option is used to control the transparency, and the BARWIDTH= option is used to control the bar widths.



Instead of overlaying the bars on each tick mark, you can use the DISCRETEOFFSET= and BARWIDTH= options in the plot statement to move the bars away from the midpoint and size them to form a cluster of side-by-side bars or overlapped bars on each tick mark. The DISCRETEOFFSET= option can be set to a value ranging from –0.5 to 0.5. The value specifies the offset as a percentage of the distance between the midpoints on the axis. Here is the previous example modified to use the DISCRETEOFFSET= and BARWIDTH= options to form a cluster of side-by-side bars around each tick mark on the X axis.



Here is the code that generates this graph.

```
/* Extract the sales data for each country from SASHELP.PRDSALE. */
data sales;
  set sashelp.prdsale(keep=country actual product);
  if (country eq "CANADA") then canada=actual;
  else if (country eq "GERMANY") then germany=actual;
  else if (country eq "U.S.A.") then usa=actual;
```

```
run;

/* Create the graph template. */
proc template;
  define statgraph offset;
    begingraph;
    entrytitle "Product Sales by Country";
    layout overlay / cycleattrs=true
      xaxisopts=(display=(tickvalues)) yaxisopts=(label="Sales");
      barchart category=product response=canada / stat=sum
name="canada"
        legendlabel="Canada" dataskin=sheen datatransparency=0.1
        barwidth=0.25 discreteoffset=-0.25;
      barchart category=product response=germany / stat=sum
name="germany"
        legendlabel="Germany" dataskin=sheen datatransparency=0.1
        barwidth=0.25;
      barchart category=product response=usa / stat=sum name="usa"
        legendlabel="U.S.A." dataskin=sheen datatransparency=0.1
        barwidth=0.25 discreteoffset=0.25;
      discretelegend "canada" "germany" "usa" / title="Country:"
        location=outside;
    endlayout;
    endgraph;
  end;
run;

/* Generate the graph. */
proc sgrender data=sales template=offset;
run;
```

In this example, the DISCRETEOFFSET=–0.25 option in the first BARCHART statement moves its bars to the left of each tick mark by 25% of the distance between the tick marks. The second BARCHART statement uses the default offset (0), which positions its bars on the tick marks. The DISCRETEOFFSET=0.25 option on the third BARCHART statement positions its bars to the right of each tick mark by 25% of the distance between the tick marks. The BARWIDTH=0.25 option in each of the BARCHART statements sizes the bars to form a cluster with no space between the bars as shown. You can adjust the DISCRETEOFFSET= and BARWIDTH= option values to overlap the bars or add space between the bars in each cluster as desired.

TIME Axes

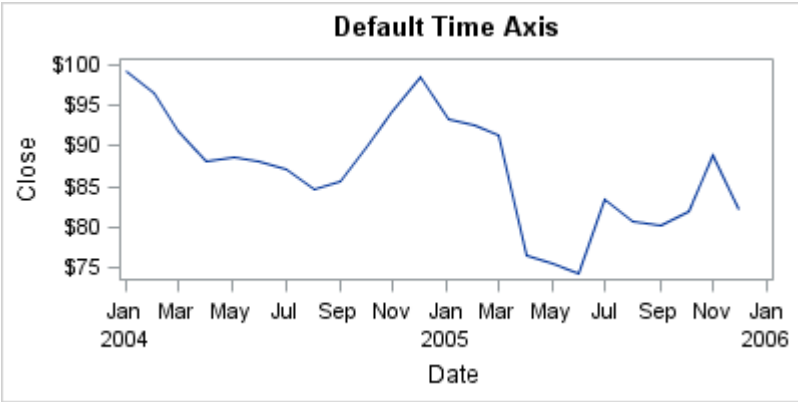
Overview of TIME Axes

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, a SAS date format is associated with the Date column. 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 column could be MMDDYY or any other date-type format. The actual format serves only as a hint and is not used directly, unless requested.

```
proc template;
  define statgraph timeaxis1;
    begingraph;
      entrytitle "Default Time Axis";
      layout overlay;
        seriesplot x=date y=close;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.stocks template=timeaxis1;
where stock="IBM" and
  date between "1jan2004"d and "31dec2005"d;
run;
```

Here is the output.



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. If the data range had been larger (for example, 1Jan1998 to 1Dec2005), the TIME axis would choose a larger interval, YEAR, to display by default.

Setting the Tick Values on a Time Axis

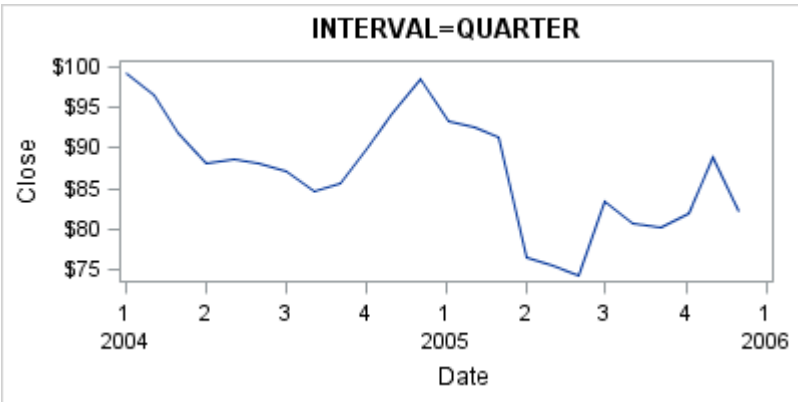
You can use the INTERVAL= option to select different date or time intervals to display. The default interval is AUTO, which chooses an appropriate interval, based on the data and the column format.

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 layout block specifies that tick values should occur at quarter intervals:

```
layout overlay / xaxisopts=(timeopts=(interval=quarter));
  seriesplot x=date y=close;
endlayout;
```

Here is example output.



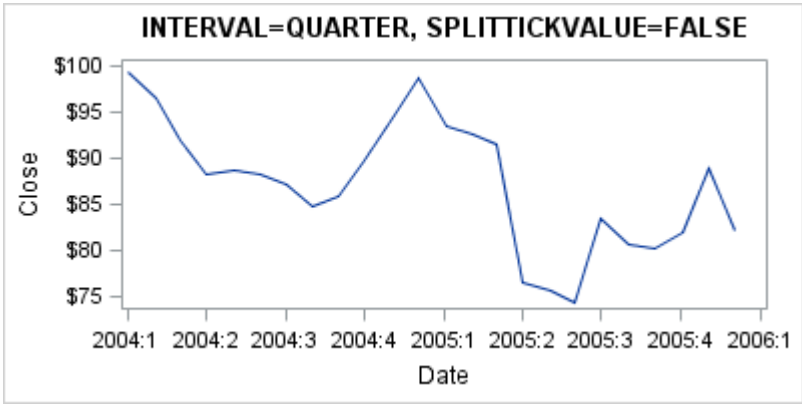
By default, the tick values are split to conserve space when possible. The following table shows the rules for splitting tick value.

Original Form	Split Form
DAY-MONTH-YEAR	DAY-MONTH YEAR
QUARTER-YEAR	QUARTER YEAR
MONTH-YEAR	MONTH YEAR
TIME-DATE	TIME DATE
TIME only	original form
YEAR, QUARTER, MONTH, or DAY only	original form
International format	original form

You can turn off the splitting feature with the SPLITTICKVALUE=FALSE option as shown in the following layout block.

```
layout overlay / xaxisopts=(timeopts=(interval=quarter
  splittickvalue=false));
  seriesplot x=date y=close;
endlayout;
```

Here is example output.



Notice that each tick value uses more space.

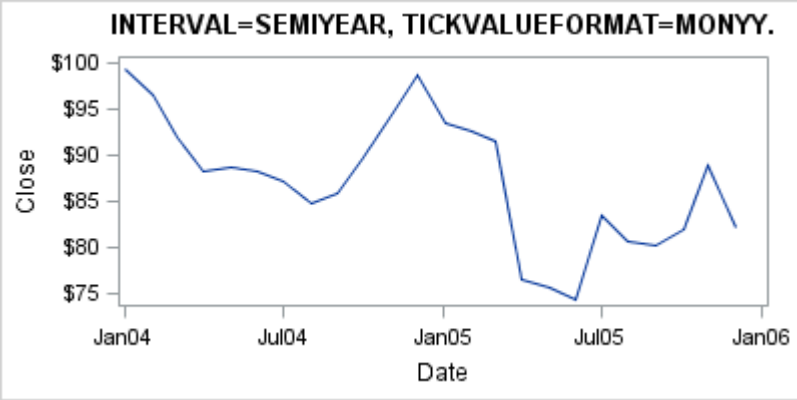
Formatting the Tick Values on a Time Axis

As with LINEAR axes, you can force a specific format for tick values with the TICKVALUEFORMAT= option. If you specify TICKVALUEFORMAT=DATA, the

format is associated with the column that is used. Or you can specify a format as shown in the following layout block.

```
layout overlay / xaxisopts=(timeopts=(interval=semiyear
  tickvalueformat=monyy.));
  seriesplot x=date y=close;
endlayout;
```

Here is example output.



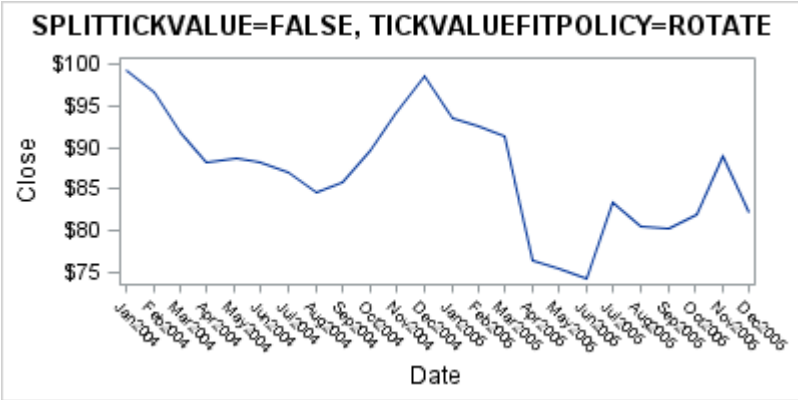
Note: GTL currently honors most but not every SAS format. For a list of the formats that are not supported, see Appendix 6, “SAS Formats Not Supported,” on page 703.

Fitting the Tick Values on a Time Axis

As with LINEAR axes, you can specify a tick value fitting policy for a TIME axis. The following policies are available: THIN, ROTATE, ROTATEALWAYS, STAGGER, ROTATETHIN, STAGGERTHIN, and STAGGERROTATE when tick values are not split. The default policy is THIN. The following layout block specifies the ROTATE policy and no label splitting.

```
layout overlay / xaxisopts=(timeopts=(interval=month
  splittickvalue=false tickvaluefitpolicy=rotate));
  seriesplot x=date y=close;
endlayout;
```

Here is example output.



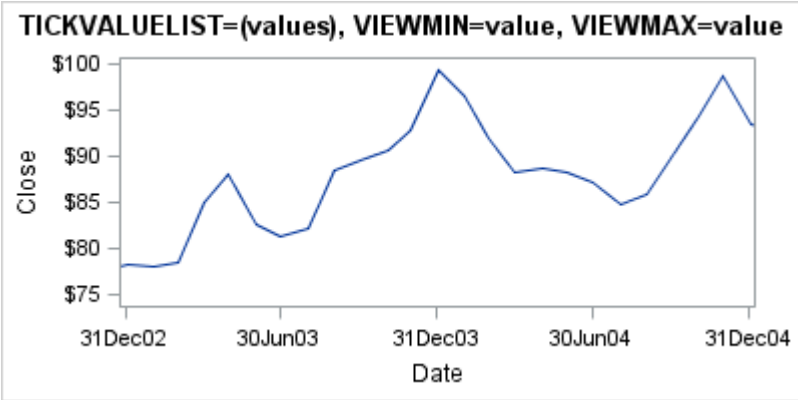
For information about the tick value fit policies that can be used with a time axis in each of the applicable layouts, see “Tick Value Fit Policy Applicability Matrix” on page 699.

Setting the Data Range on a Time Axis

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.

```
proc template;
  define statgraph timeaxis1;
    begingraph;
      entrytitle "TICKVALUELIST=(values), VIEWMIN=value,
VIEWMAX=value";
      layout overlay / xaxisopts=(timeopts=(tickvalueformat=data
          viewmin="31Dec2002"d viewmax="31Dec2004"d
          tickvaluelist=("31Dec2002"d "30Jun2003"d
          "31Dec2003"d "30Jun2004"d "31Dec2004"d)));
        seriesplot x=date y=close;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.stocks template=timeaxis1;
where stock="IBM" and date between "1dec2002"d and "31dec2005"d;
run;
```

Here is the output.



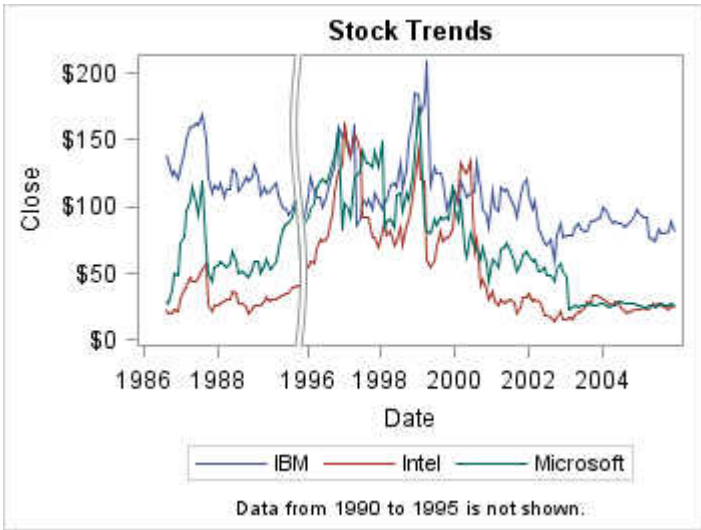
Creating a Broken Time Axis

In some cases, you might want to remove portions of a plot where the data is sparse or is of little interest. Starting with SAS 9.4M1, you can use the INCLUDERANGES= option to specify ranges of data that you want to include on a time axis. All data and axis tick values that fall outside of the ranges that you specify are clipped from the output. At each point where a gap in the axis exists, break lines are drawn perpendicular to the axis to indicate the break. The axis tick value sequence also reflects the gaps.

Here is an example of a closing price and date plot in which the INCLUDERANGES= option is used to remove the data between 1990 and 1995.

```
proc template;
  define statgraph stockplot;
    begingraph;
      entrytitle "Stock Trends";
      entryfootnote "Data from 1990 to 1995 is not shown.";
      layout overlay /
        xaxisopts=(timeopts=(includeranges=(
          '01jan1986'd-'31dec1989'd '01jan1996'd-'31dec2005'd)
          tickvalueformat=year4.));
        seriesplot x=date y=close / name="series" group=stock;
        discretelegend "series";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.stocks template=stockplot;
run;
```

The INCLUDERANGES= option specifies ranges as a space-separated list of *start–end* value pairs. In this example, ranges '01jan1986'd–'31dec1989'd and '01jan1996'd–'31dec2005'd are specified, which creates a break between 1989 and 1996. Here is the output.



Notice the break lines and the gap in the X-axis tick value sequence where the break occurs.

The following restrictions apply to broken axes:

- A broken axis is valid for linear and time axes in an OVERLAY layout only.
- Only one axis can be broken.
- When plots are associated with the X and X2 axes or with the Y and Y2 axes, neither axis can be broken.
- When an axis is broken, data tips and selectable graphical elements are not supported.

For more information about the INCLUDERANGES= option, see "Options for Time Axes Only" in *SAS Graph Template Language: Reference*.

Starting with SAS 9.4M3, you can change the default break indicator. Rather than break lines that span the entire data display, you can specify a break symbol that appears only on the axis line. For more information, see "Creating a Broken Linear Axis" on page 137.

LOG Axes

Overview of LOG Axes

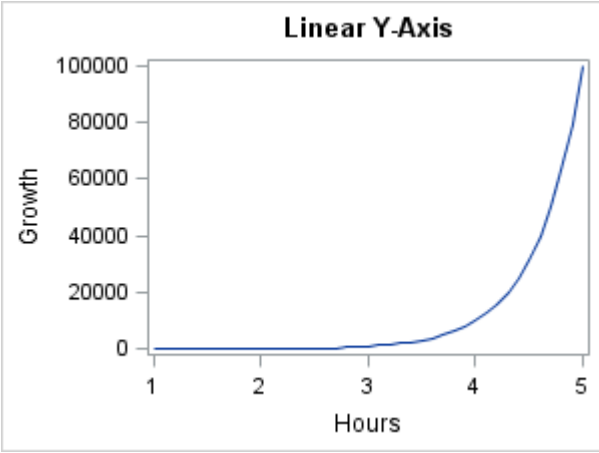
An axis displaying a logarithmic scale is very useful when your data values span orders of magnitude. For example, when you plot your growth data with a linear axis, you suspect that the growth rate is exponential.

```
proc template;
  define statgraph overlayaxes.logaxis1;
    begingraph;
      entrytitle "Linear Y-Axis";
      layout overlay;
        seriesplot x=Hours y=growth;
```

```
        endlayout;
      endgraph;
    end;
  run;

  proc sgrender data=growth template=overlayaxes.logaxis1;
  run;
```

Note: See Example Code 11.3 on page 117.

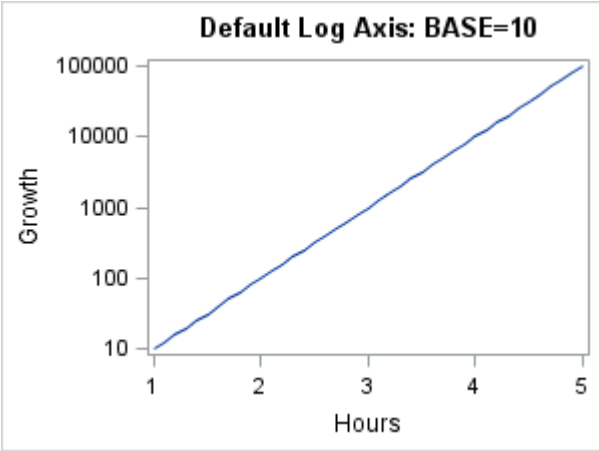
Here is the output.



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 as shown in the following layout block.

```
layout overlay / yaxisopts=(type=log);
  seriesplot x=Hours y=growth;
endlayout;
```

Any of the four axes can be a log axis. Here is example output.



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:

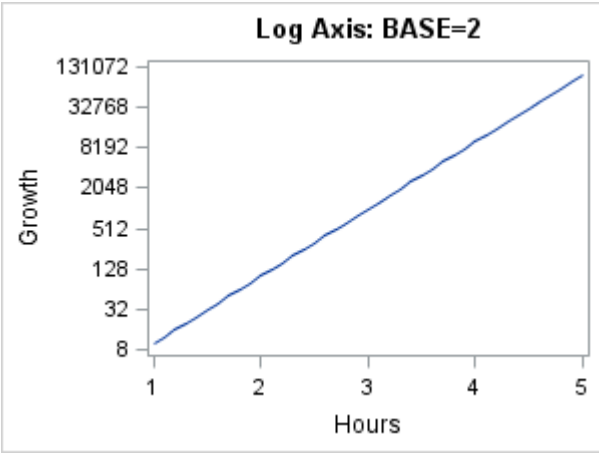
```
NOTE: Log axis cannot support zero or negative values in the data range. The axis
type will
      be changed to LINEAR.
```

Setting the Base on a Log Axis

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. The following layout block specifies base 2.

```
layout overlay / yaxisopts=(type=log logopts=(base=2));
  seriesplot x=Hours y=growth;
endlayout;
```

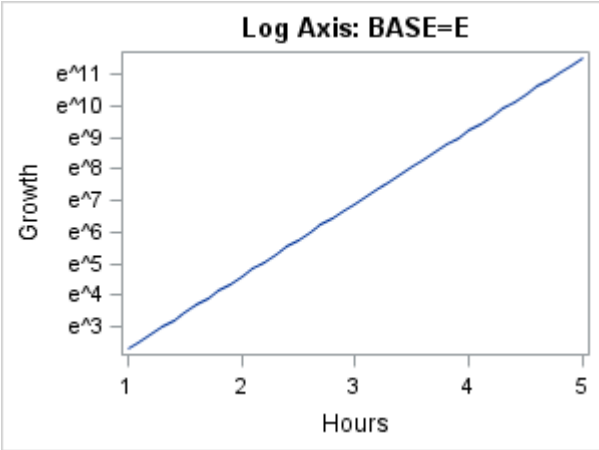
Here is example output.



The following layout block specifies base E.

```
layout overlay / yaxisopts=(type=log logopts=(base=e));
  seriesplot x=Hours y=growth;
endlayout;
```

Here is example output.



Setting the Tick Intervals on a Log Axis

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.

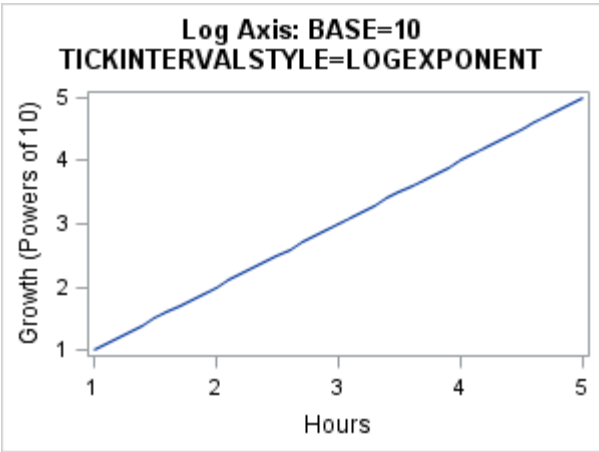
LOGEXPONENT
: Major ticks are placed at uniform intervals at integer powers of the base. The tick values are only the integer exponents for all bases.

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 as shown in the following layout block.

```
layout overlay / yaxisopts=(type=log label="Growth (Powers of 10)"
  logopts=(base=10 tickintervalstyle=logexponent));
  seriesplot x=Hours y=growth;
endlayout;
```

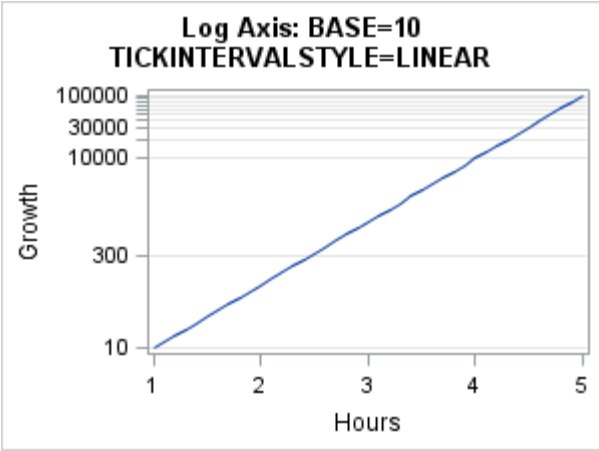
Here is example output.



When using TICKINTERVALSTYLE=LINEAR, it is visually helpful to turn on the grid lines as shown in the following layout block.

```
layout overlay / yaxisopts=(type=log  griddisplay=on
  logopts=(base=10 tickintervalstyle=linear));
  seriesplot x=Hours y=growth;
endlayout;
```

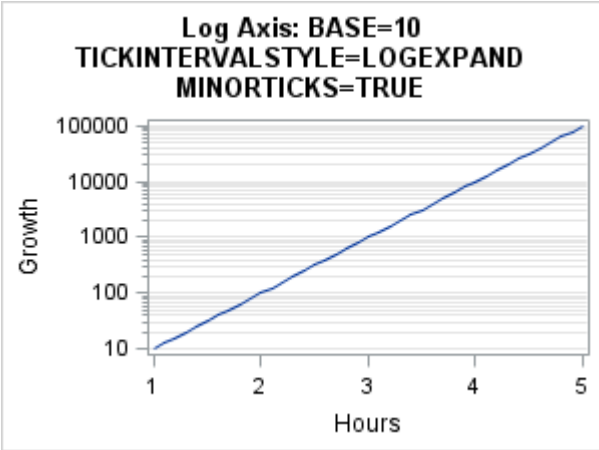
Here is example output.



When using BASE=10 and TICKINTERVALSTYLE=LOGEXPAND or TICKINTERVALSTYLE=LOGEXPONENT, you can add minor ticks and minor grid lines to emphasize the log scale as shown in the following layout block.

```
layout overlay / yaxisopts=(type=log griddisplay=on
  logopts=(base=10 tickintervalstyle=logexpand
  minorticks=true minorgrid=true));
  seriesplot x=Hours y=growth;
endlayout;
```

Here is example output.



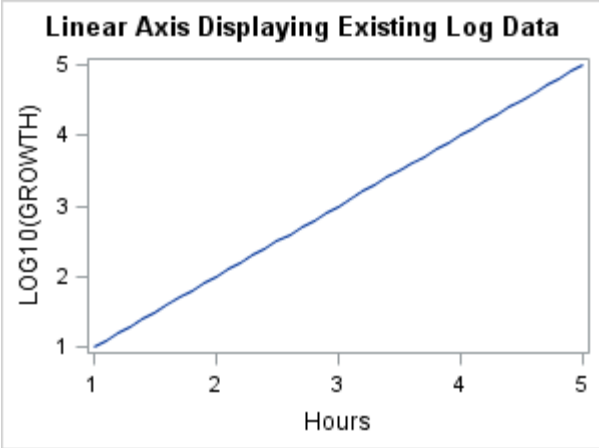
The data range of a log axis can be set with the VIEWMIN= and VIEWMAX= log options. It can also be set with the TICKVALUELIST= option when TICKVALUEPRIORITY=TRUE. See “Setting the Tick Values on a Log Axis” on page 163.

If your input data has already been transformed into log values, always use a LINEAR axis to display them as shown in the following layout block.

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

```
endlayout;
```

Do not use a LOG axis in this case. Here is example output.



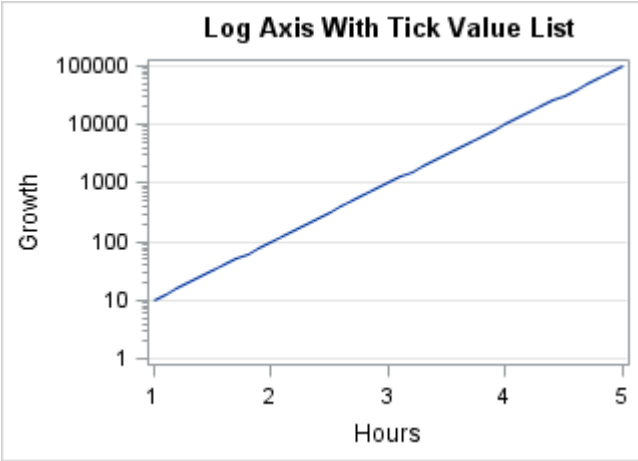
Setting the Tick Values on a Log Axis

You can use the TICKVALUELIST=(*values*) option on a log axis to specify the values that are to appear on the axis. By default, the values in the list that are within the data range or the range specified by the VIEWMIN= and VIEWMAX= options are displayed on the axis. All other values in the list are ignored. If you want the values specified in the list to take precedence, you can specify the TICKVALUEPRIORITY=TRUE. In that case, if the values are outside of the data range, the axis range is extended to include all of the values in the list, and the VIEWMIN= and VIEWMAX= options are ignored.

The following layout block specifies the values 1, 10, 100, 1000, 10000, and 100000.

```
layout overlay /
  yaxisopts=(griddisplay=on type=log
    logopts=(base=10 tickvaluepriority=true
    tickintervalstyle=logexpand
    tickvaluelist=(1 10 100 1000 10000 100000)
    minorticks=true));
  seriesplot x=Hours y=growth;
endlayout;
```

Because the data range is 10 to 100000, the TICKVALUEPRIORITY=TRUE option is specified to extend the lower axis range down to 1. Here is example output.



You can use the VALUETYPES= option to specify the TICKVALUELIST= option values in a scale and format other than that specified by the TICKINVERVALSTYLE= option. You can also use the VALUETYPES= option to specify VIEWMIN= and VIEWMAX= values. For example, in the previous example, you can specify VALUESTYPE=EXPONENT, and then list the TICKVALUELIST= option values as exponents of 10 rather than expanded values as shown in the following layout block.

```
layout overlay /
  yaxisopts=(griddisplay=on type=log
    logopts=(base=10 tickvaluepriority=true
    tickintervalstyle=logexpand
    valuestype=exponent
    tickvaluelist=(0 1 2 3 4 5)
    minorticks=true));
  seriesplot x=Hours y=growth;
endlayout;
```

The resulting graph is the same. This is particularly useful for large axis values. For more information about using the VALUESTYPE= option, see VALUESTYPE=.

Formatting the Tick Values on a Log Axis

Starting with SAS 9.4M3, you can force a specific format for tick values on a log axis with the TICKVALUEFORMAT= option. The TICKVALUEFORMAT= option is honored only when the tick-interval style is LOGEXPAND or LINEAR. It is ignored when the tick-interval style is LOGEXPONENT. Use the TICKINTERVALSTYLE= option to change the tick-interval style. To use the format that is associated with the column, specify TICKVALUEFORMAT=DATA. To use a different format, specify TICKVALUEFORMAT=*format*.

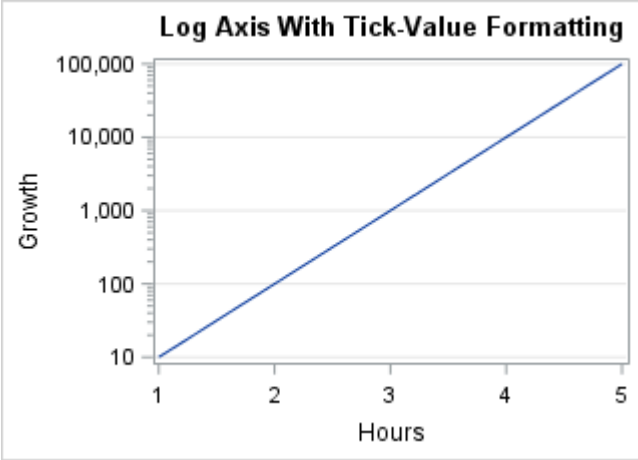
Note: GTL currently honors most but not all SAS formats. For a list of the formats that are not supported, see Appendix 6, “SAS Formats Not Supported,” on page 703.

When TICKINTERVALSTYLE=LOGEXPAND, the specified format is honored for base 2 and base 10 logarithmic scales. When TICKINTERVALSTYLE=LINEAR, the

specified format is honored for base 10, base 2, and base E logarithmic scales. Here is an example.

```
layout overlay /
   yaxisopts=(griddisplay=on type=log
      logopts=(base=10 tickintervalstyle=logexpand
      tickvalueformat=comma7.
      minorticks=true));
   seriesplot x=Hours y=growth;
endlayout;
```

Here is example output.



Avoiding Plot Data Conflicts

All plot statements have one or more required arguments that map input data columns 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 example:

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

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

The template in this example will compile with no compilation errors or warnings because no variables are checked at compile time. However, when the template is executed, the following warnings are written to the SAS log:

```
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 layout block, the wrong variable name was used for the GROUP= option.

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

In the data, the column is named Sex, not Gender. This is not regarded as an error condition—the bar chart is drawn without groups in this case.

Overlay Examples

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. This section shows you how to overlay statements and use other GTL features to create more complex plots.

Vertical and Horizontal Bar-Line Charts

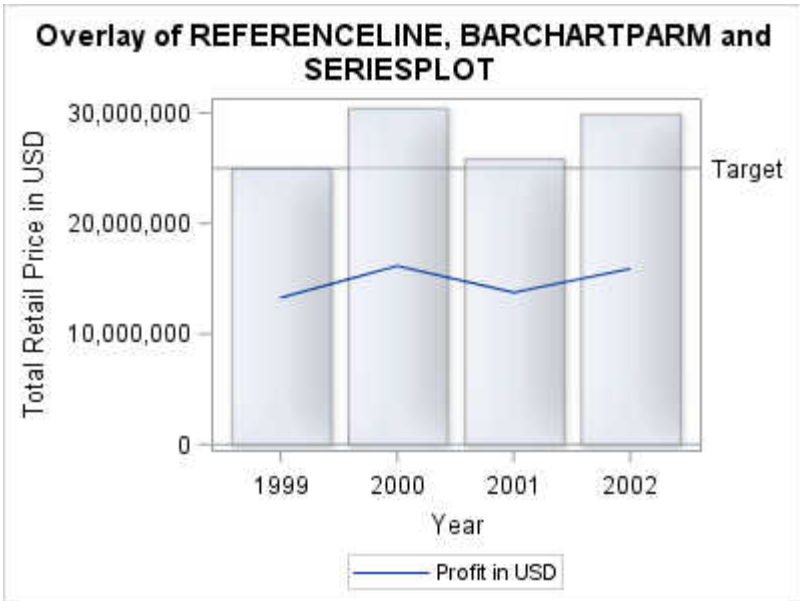
Bar-Line Charts

GTL does not provide a BARLINE statement that you can use to create a bar-line chart. However, you can create this type of chart by overlaying a series plot on a bar chart as shown in the following SAS code.

```
proc template;
  define statgraph barline;
    begingraph;
      entrytitle "Overlay of REFERENCELINE, BARCHARTPARM and
SERIESPLOT";
      layout overlay;
        referenceline y=25000000 / curvelabel="Target";
        barchartparm category=year response=retail / dataskin=matte
          fillattrs=(transparency=0.5)
```

```
          fillpatternattrs=(pattern=R1 color=lightgray);
        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;

/* generate the graph */
proc sgrender data=orsales template=barline;
  format retail profit comma12.;
run;
```

The output reflects the requested stacking order.



Horizontal Bar Charts

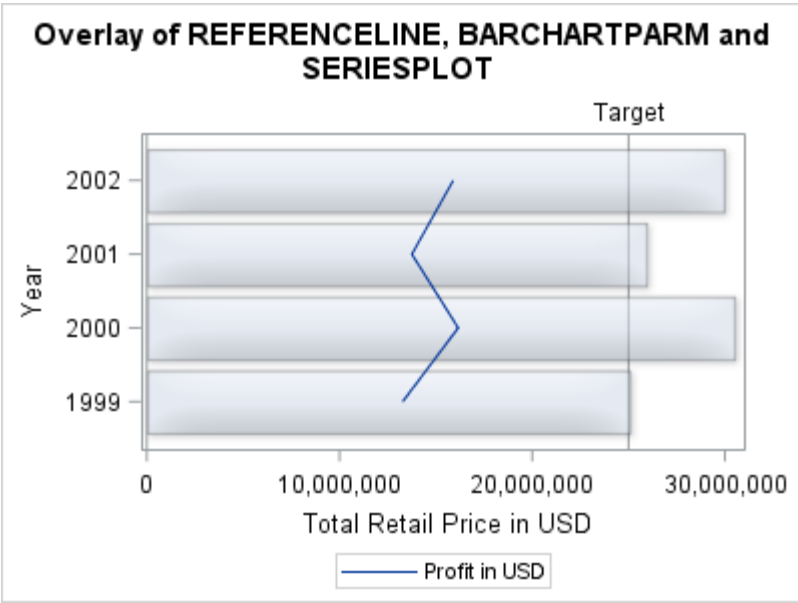
When creating a bar chart, it is sometimes desirable to rotate the chart from vertical to horizontal. GTL does not provide separate statements for vertical and horizontal charts—each is considered to be the same plot type with a different orientation. To create the horizontal version of the bar-line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

```
proc template;
  define statgraph barline2;
    begingraph;
```

```
      entrytitle "Overlay of REFERENCELINE, BARCHARTPARM and
SERIESPLOT";
      layout overlay;
        referenceline x=25000000 / curvelabel="Target";
        barchartparm category=year response=retail / orient=horizontal
          dataskin=matte fillattrs=(transparency=0.5)
          fillpatternattrs=(pattern=R1 color=lightgray);
        seriesplot x=profit y=year / name="series"
          legendlabel="Profit in USD";
          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;

/* generate the graph */
proc sgrender data=orsales template=barline2;
  format retail profit comma12.;
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 how it was, but the ORIENT=HORIZONTAL option was added to swap the axis mappings. The data set up and SGRENDER step are unchanged.

Here is the output.

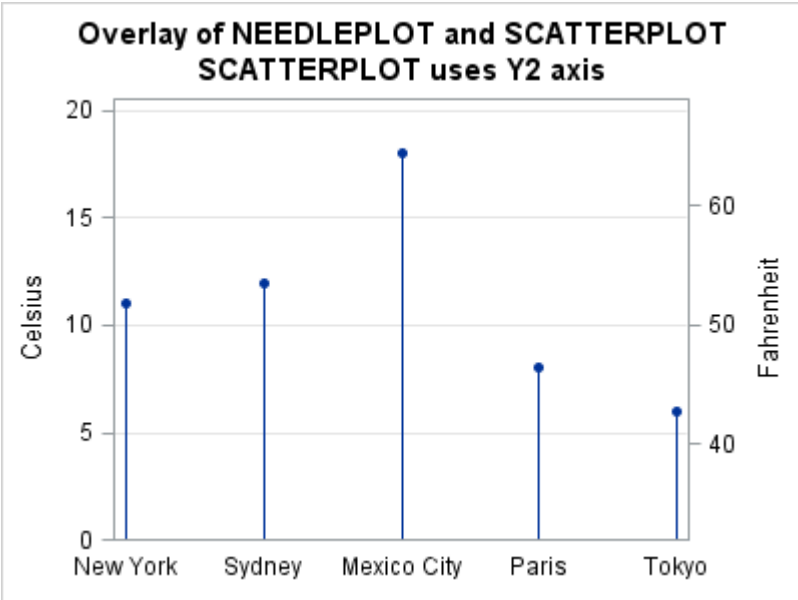


This same strategy would be used to create a horizontal box plot or histogram. If you wanted to reverse the ordering of the Y axis, you could add the REVERSE=TRUE option to the Y-axis options as shown in the following statement.

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

Plot with 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 where a Y2 axis opposes the Y axis to the right and an X2 axis opposes the X axis at the top of the layout container. Here is an example provides a complete program that generates this type of graph. In this example, Fahrenheit temperatures are displayed on a separate Y2 axis while Celsius temperatures are displayed on the Y axis as shown in the following figure.



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. Here is the template code for this example.

```
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;
```

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.

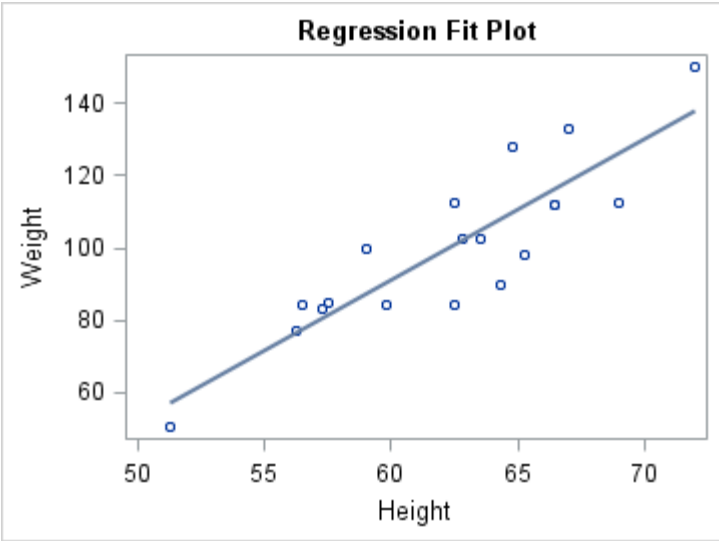
Plot with Fit Line

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 must be used with one or more layout statements. For simplicity, the most basic layout statement is used in this discussion: 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 in which the plot statements are specified, and the last one is drawn on top. Here is an example.

```
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;
```

Here is the output.

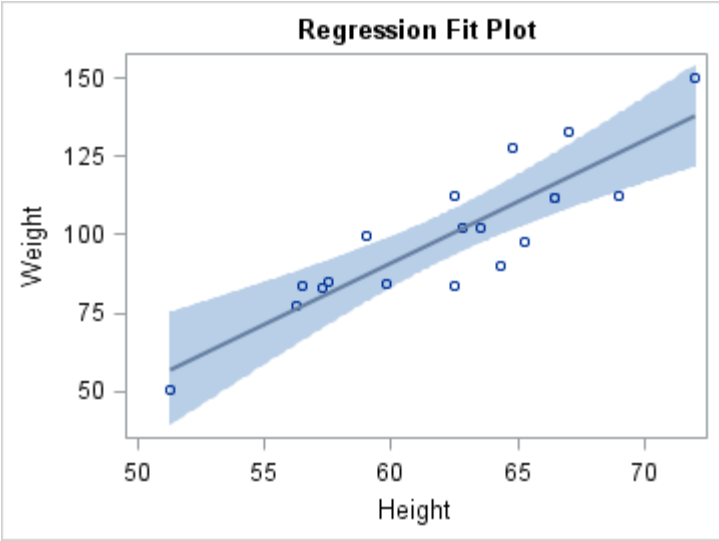


Plot with Fit Line with Confidence Bands

In the example in "Plot with Fit Line" on page 170, 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, and specify the unique name as its required argument as shown in the following layout block.

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

Notice that the MODELBAND statement appears first in the template, ensuring that the band appears behind the scatter points and fit line. A MODELBAND statement must be used with a REGRESSIONPLOT, LOESSPLOT, or PBSPLINEPLOT statement. Here is example output.



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 that you want. Here is an example of using non-computed, standalone plots to build the fit plot. First, choose a procedure to do the analysis. The following example uses the REG procedure.

```
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 columns and observations in Sashelp.Class plus, for each observation, the computed columns P, LCLM, and UCLM. Here is a partial listing of the Predict data set.

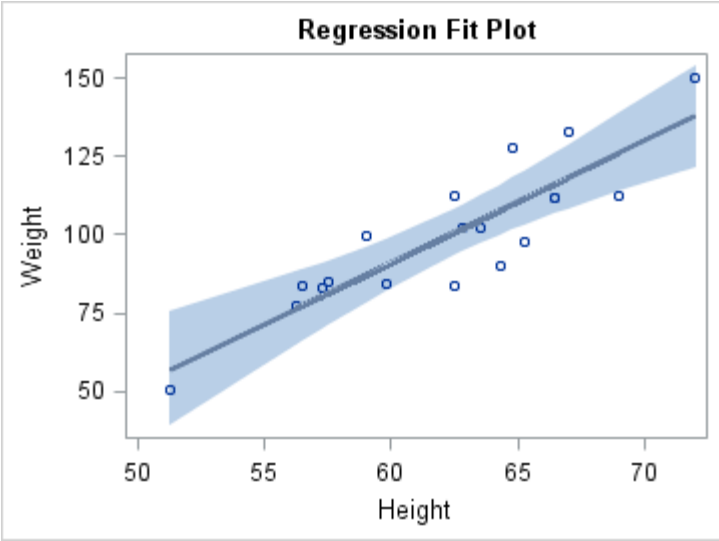
Obs	Name	Sex	Age	Height	Weight	p	lclm	uclm
1	Alfred	M	14	69.0	112.5	126.006	113.554	138.458
2	Alice	F	13	56.5	84.0	77.268	65.782	88.755
3	Barbara	F	13	65.3	98.0	111.580	102.899	120.261
4	Carol	F	14	62.8	102.5	101.832	94.336	109.329
5	Henry	M	14	63.5	102.5	104.562	96.897	112.226
6	James	M	12	57.3	83.0	80.388	69.782	90.993
7	Jane	F	12	59.8	84.5	90.135	81.762	98.509
…								

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

```
proc template;
  define statgraph fit;
    begingraph;
      entrytitle "Regression Fit Plot";
```

```
      layout overlay;
        bandplot x=height
          limitupper=uclm
          limitlower=lclm /
          connectorder=axis
          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;
```

Here is the output.



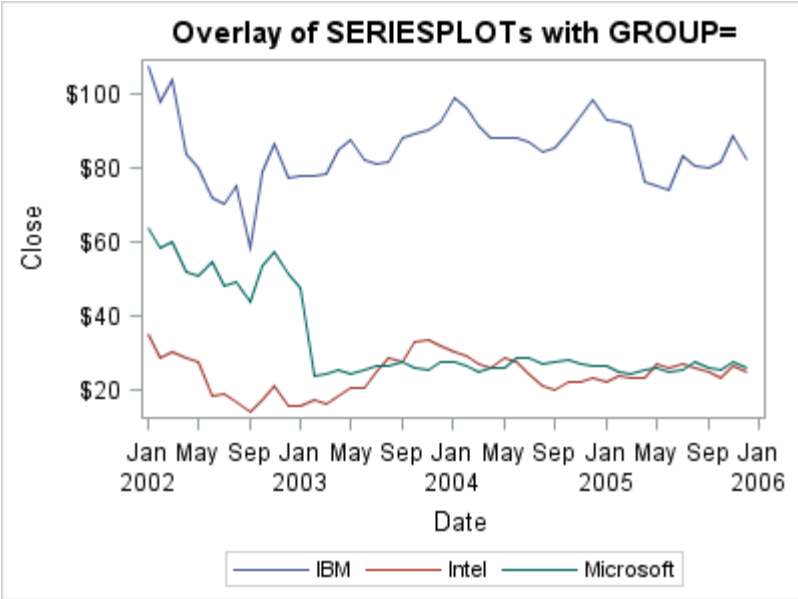
Plot of Grouped Data

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 as shown in the following example.

```
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 from the HTMLBlue style are used:



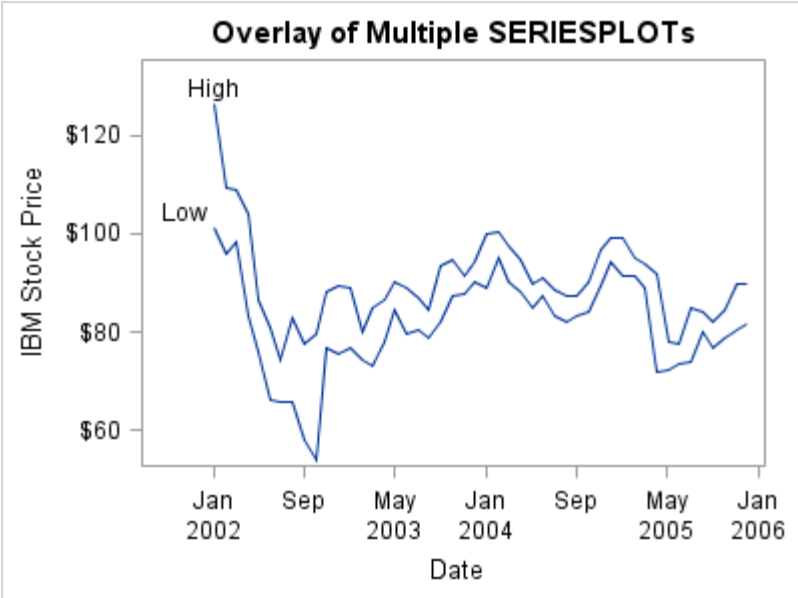
Using Overlays to Graph Multiple Response Variables

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 as shown in the following example.

```
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;
```

Here is the output.



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

Plot with Insets

In some cases, you might want to inset additional information in your graph to further explain the plots or the plot data. Such information might include descriptive text, a small table of additional statistics, or information about the plot values displayed along a plot axis. GTL provides several ways in which can inset supplemental information in your graphs. For more information, see Chapter 21, “Adding Insets to Your Graph,” on page 383.

Plot Appearance

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 and labels in the example in “Using Overlays to Graph Multiple Response Variables” on page 174, you can use the LINEATTRS= and CURVELABELATTRS= options as shown in the following layout block:

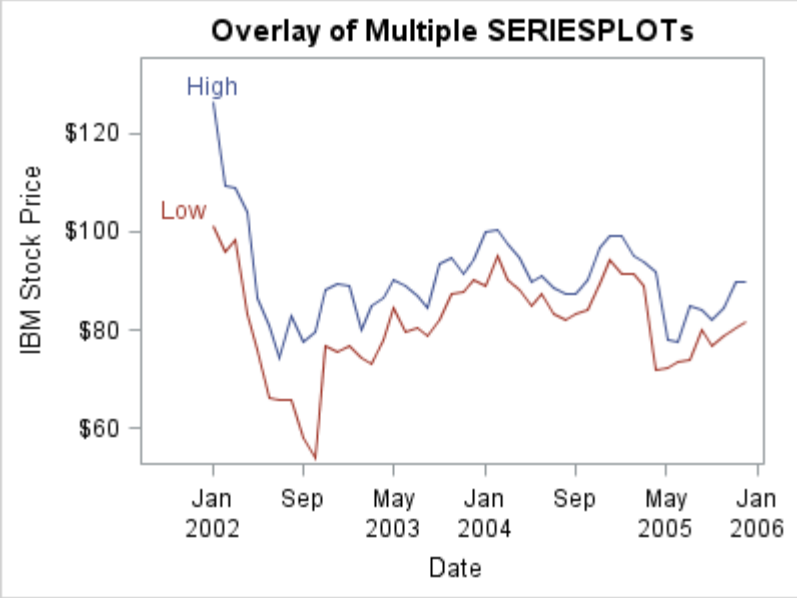
```
layout overlay / yaxisopts=(label="IBM Stock Price");
  seriesplot x=date y=high / curvelabel="High"
    lineattrs=GraphData1
    curvelabelattrs=GraphData1;
```

```
  seriesplot x=date y=low  / curvelabel="Low"
    lineattrs=GraphData2
    curvelabelattrs=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 about how set the appearance features of plots, see Chapter 27, “Managing Your Graph’s Appearance,” on page 491.

12

Creating Overlay Graphs with Equated Axes Using the OVERLAYEQUATED 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

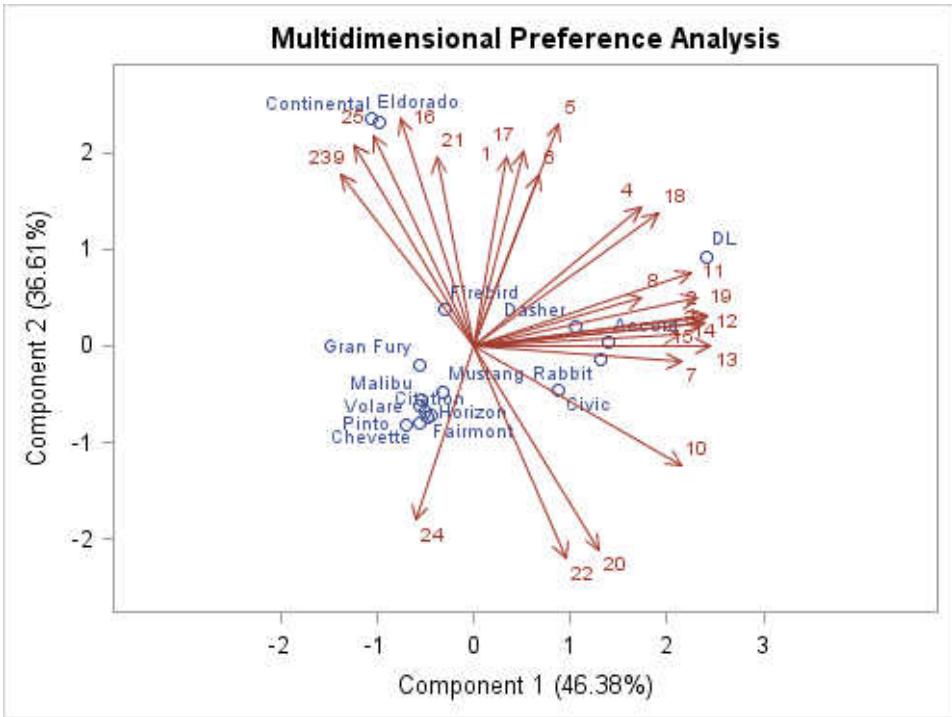


Figure 12.2 Sample Plot from PROC LOGISTIC

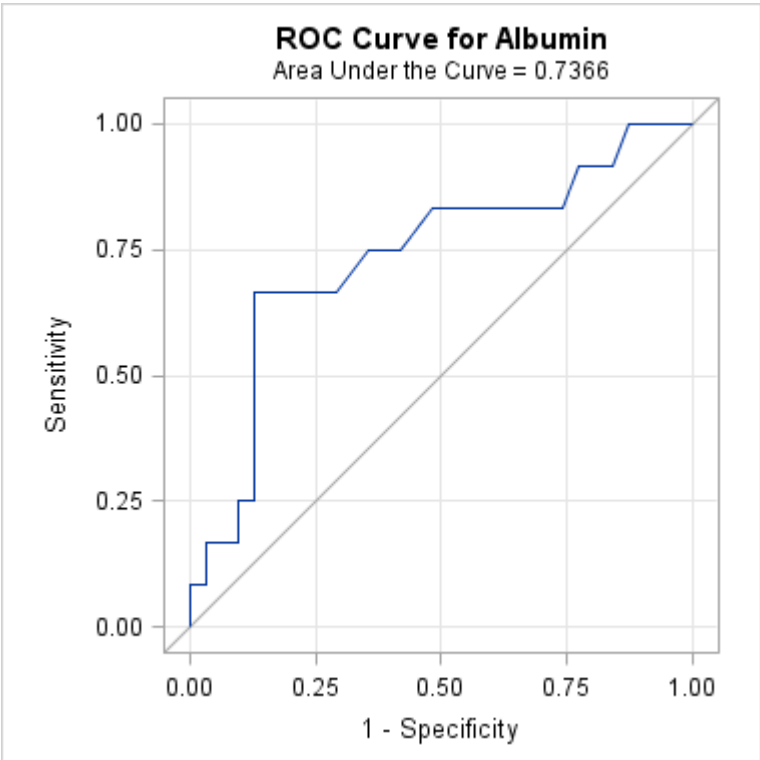
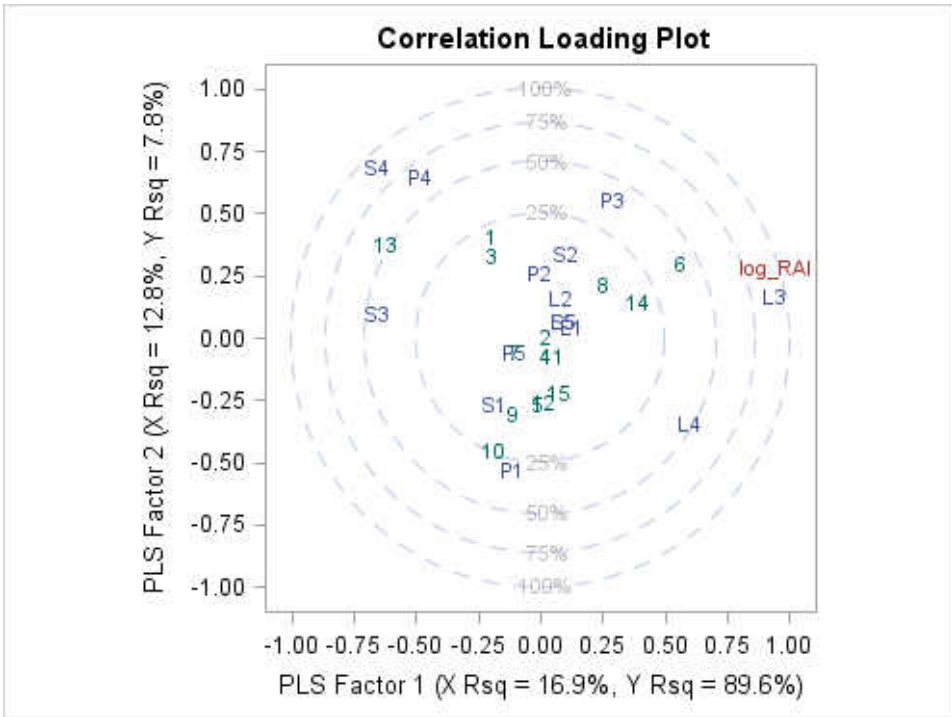


Figure 12.3 Sample Plot from PROC PLS

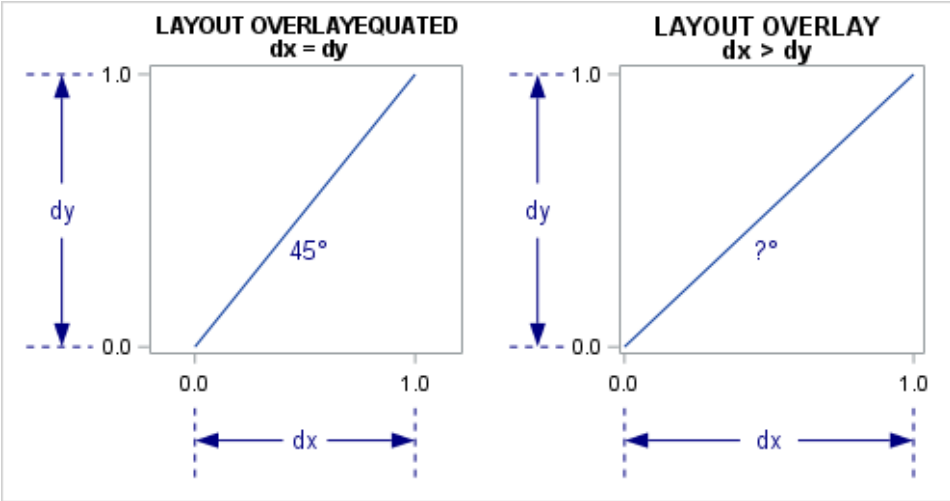


Whenever the same units of measure are used on both axes, it is desirable that the distance displayed between the same data interval be the same on both axes. To achieve this effect, you must use an OVERLAYEQUATED layout.

For specifying plot statements, the OVERLAYEQUATED layout is similar to the OVERLAY layout: you can specify one or more 2-D 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 180.)

The following figure illustrates how a series plot might be displayed when it is specified within an OVERLAYEQUATED layout rather than an OVERLAY layout:



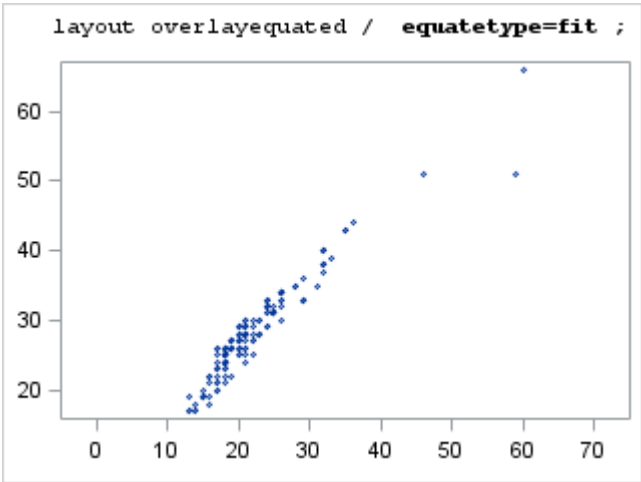
Managing Axes in OVERLAYEQUATED Layouts

Types of Equated Axes

The EQUATETYPE= option of the LAYOUT OVELAYEQUATED statement manages the display of the axes. The following values are available:

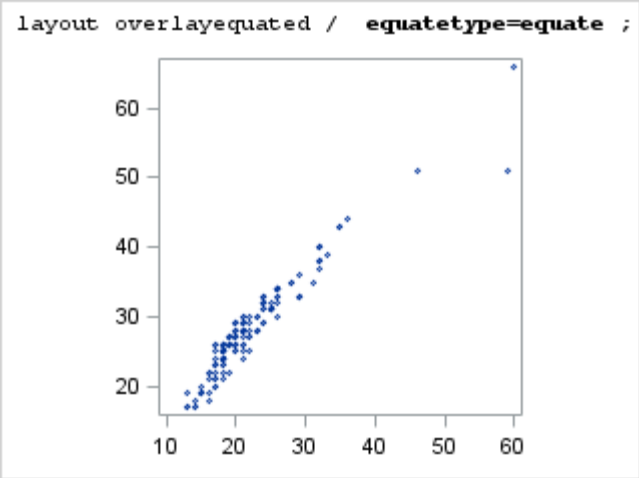
FIT

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.



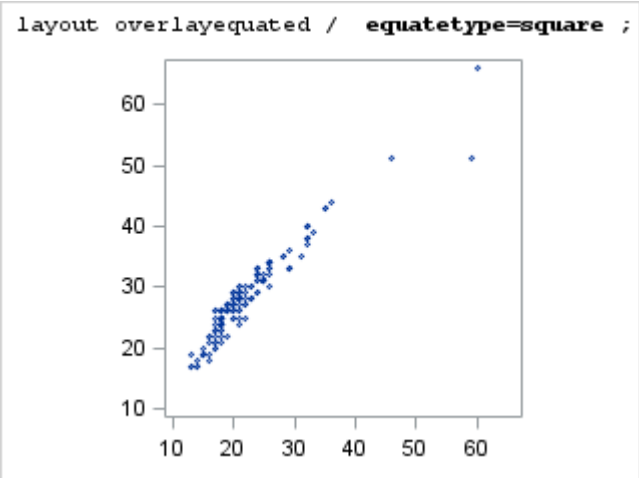
EQUATE

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.

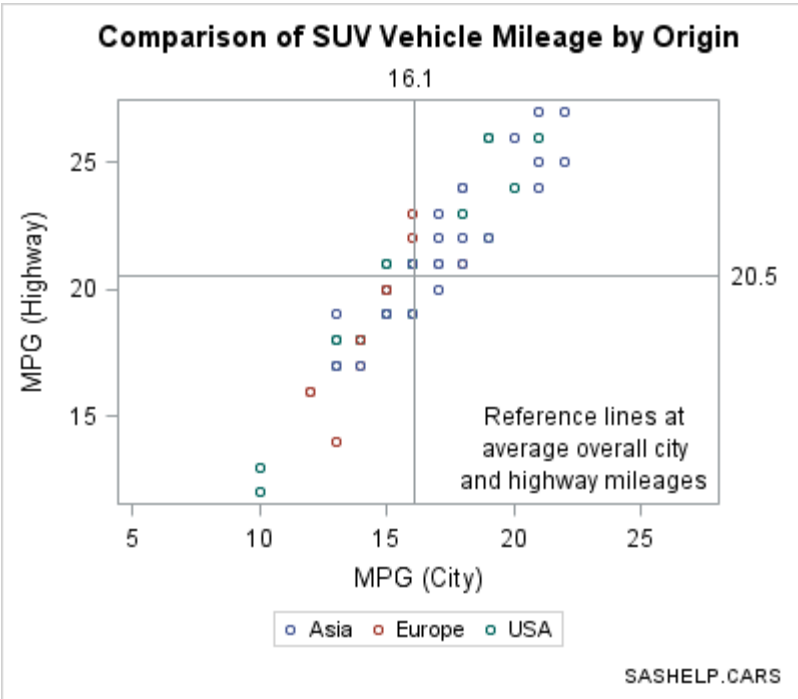


The following example template uses the EQUATETYPE= option:

```
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;
```

Here is the output.



Note: This program uses several features, such as run-time 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 suboptions 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. The XAXIS= and YAXIS= options are ignored.

"Managing Axes in OVERLAY Layouts " on page 103 discusses many of the axis options that are available for managing graph axes.

Equated Overlay Layout Examples

To illustrate how to use the equated layout, a simplified version of the PPPLOT template that is supplied with PROC UNIVARIATE is used, 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 dynamic variables to pass in values for options, but in essence, the following template is equivalent. The dynamic variables 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:

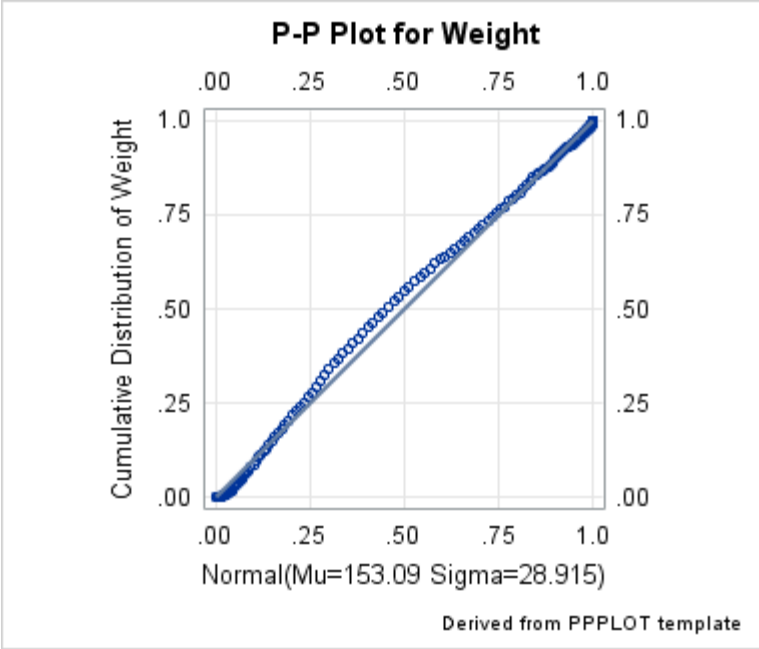
```
ods graphics on;
ods select ppplot;
ods output ppplot=ppdata;
proc univariate data=sashelp.heart;
  var weight;
  ppplot / normal square;
run;
quit;
proc sgrender data=ppdata
  template=pp_plot;
run;
```

Here is the output.

The following OVERLAYEQUATED statement modifies the equated axes:

```
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));
```

Here is example output.


P-P Plot for Weight
.00
.25
.50
.75
1.0
Cumulative Distribution of Weight
Normal(Mu=153.09 Sigma=28.915)
Derived from PPPLOT template

13

Creating Overlay 3-D Graphs Using the OVERLAY3D Layout

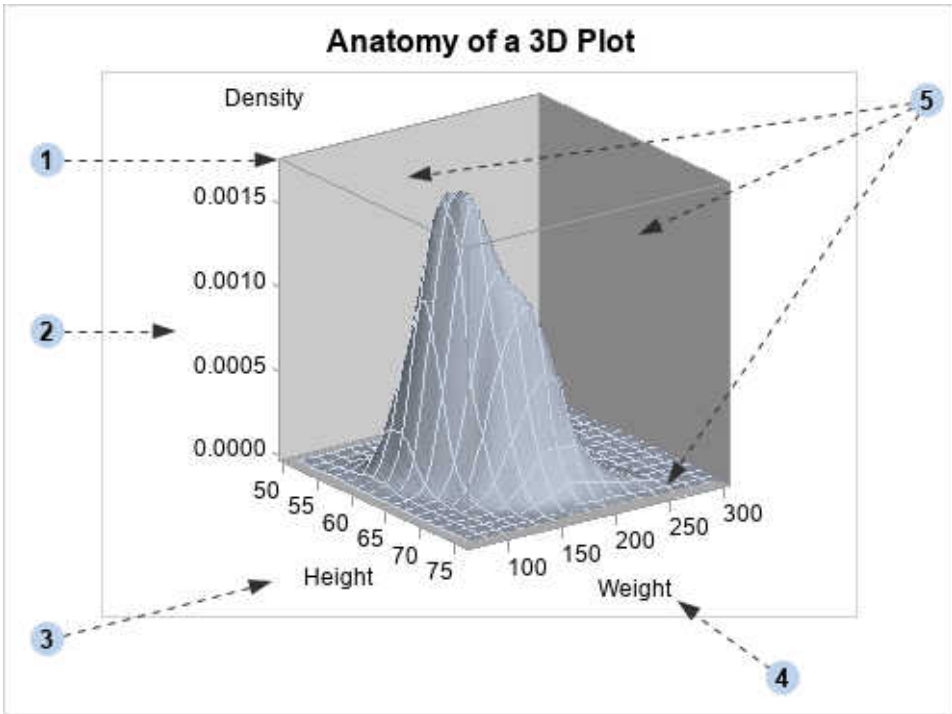
The LAYOUT OVERLAY3D Statement

GTL has one layout for 3-D graphics: the LAYOUT OVERLAY3D statement. Two 3-D plot statements can be placed within this layout: BIHISTO3DPARM and SURFACEPLOTPARM. No 2-D 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 3-D representation of a surface or a bi-variate histogram (possibly overlaid together). The 3-D layout has features that 2-D 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 3-D plot:



1 Cube
2 Z axis
3 X axis
4 Y axis
5 Wall

Data Requirements for 3-D Plots

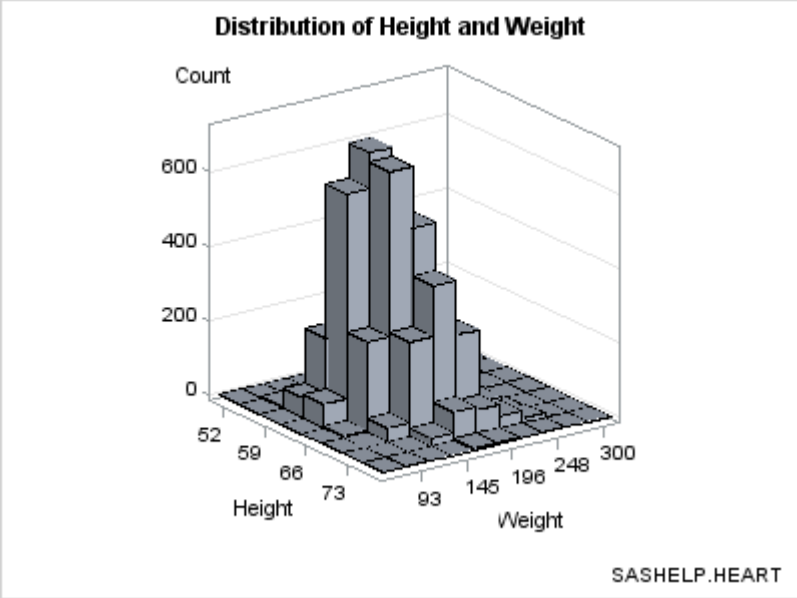
Overview of the Data Requirements for 3-D Plots

Both of the plot statements that can be used in the OVERLAY3D layout are parameterized plots.(See “Overview of the Plot Types” on page 51). 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 type 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. That means that their data ranges have been divided into equal sized intervals (bins), and that the 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 column in the output data set.

Here is a partial listing of the Kde data set.

Obs	value1	value2	count
1	51.5000	67.000	1
2	51.5000	92.889	0
3	51.5000	118.778	0
4	51.5000	144.667	0
5	51.5000	170.556	0
6	51.5000	196.444	0
7	51.5000	222.333	0
8	51.5000	248.222	0
9	51.5000	274.111	0
10	51.5000	300.000	0
11	55.0714	67.000	1
12	55.0714	92.889	8
13	55.0714	118.778	16
14	55.0714	144.667	11
15	55.0714	170.556	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.

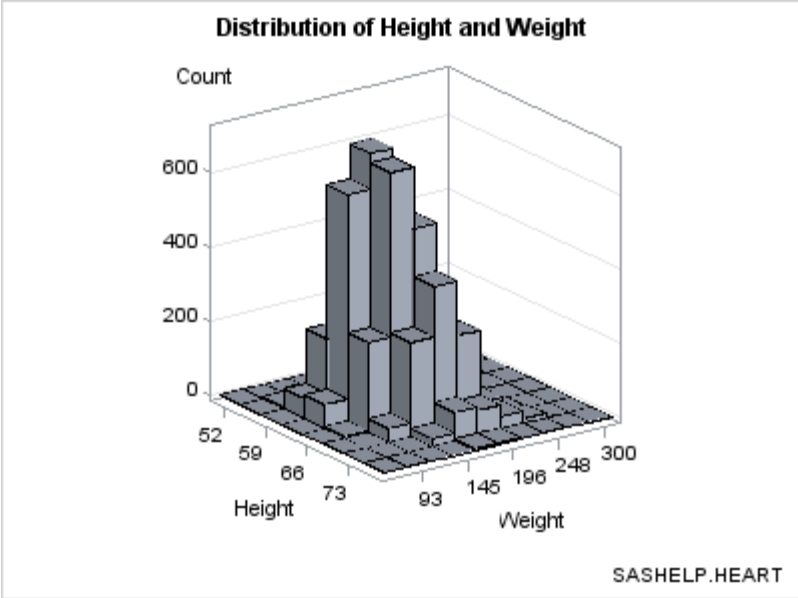
Example Code 13.1 *KDE Data Plot Template*

```
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;
```

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 labeled 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."

Executing this template generates the following output.

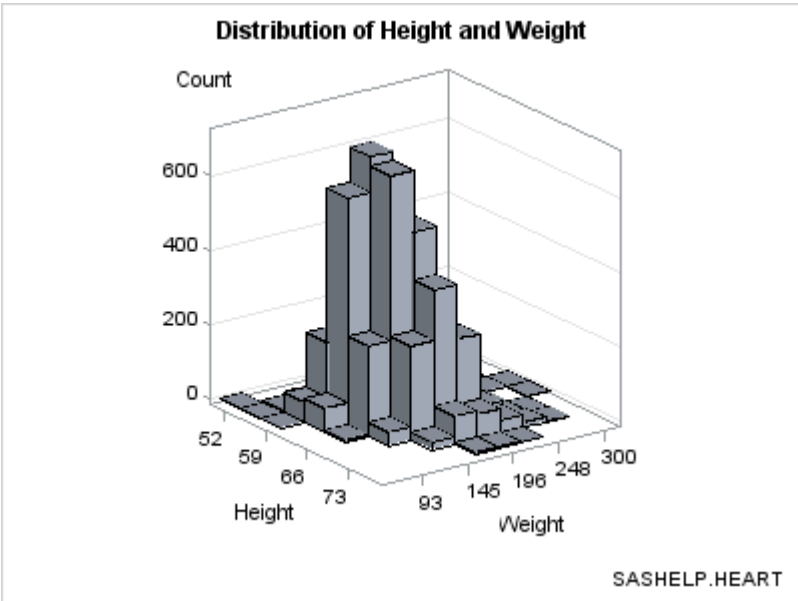
```
proc sgrender data= kde template=bihistogram1a;
  label value1="Height" value2="Weight";
run;
```



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 versus portions of the grid with no data. To render the template in Example Code 13.1 on page 190 for only nonzero values:

```
proc sgrender data=kde template=bihistogram1a;
  where count > 0;
  label value1="Height" value2="Weight";
run;
```

Here is the output.



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:

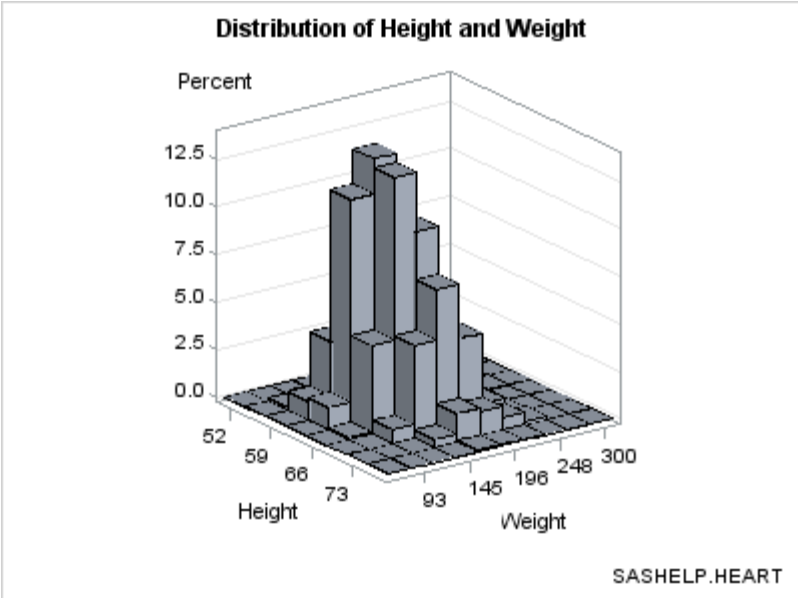
```
proc kde data=sashelp.heart;
```

```
  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;
```

To render the template in Example Code 13.1 on page 190 with the transformed data:

```
proc sgrender data=kde template=bihistogram1a;
   label value1="Height" value2="Weight";
run;
```

Here is the output.



Setting Bin Width. Another technique for binning data is to set a bin width and compute the number of observations in each bin. This technique ensures that the bins are of a good size. In the following DATA step, 5 is the bin width for HEIGHT and 25 for WEIGHT.

```
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 as shown in the following example.

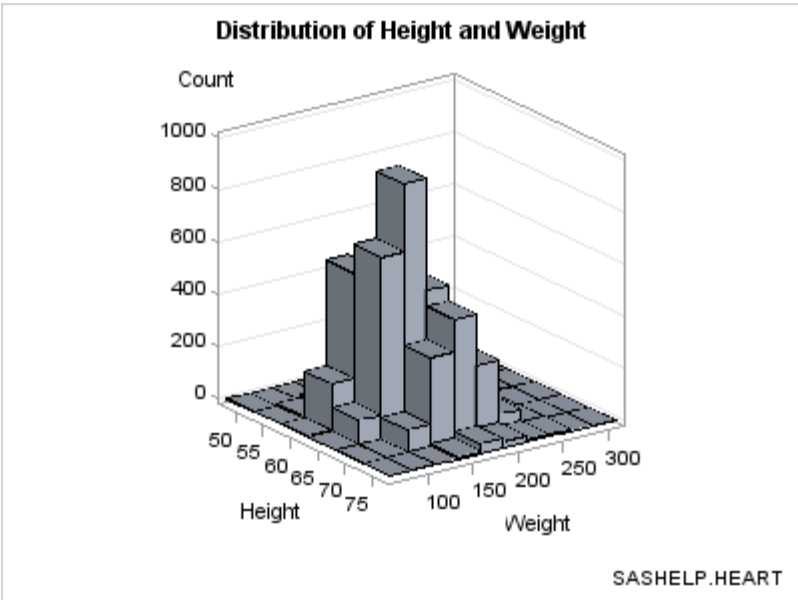
```
proc summary data=heart nway completetypes;
  class height weight;
```

```
  var height;
  output out=stats(keep=height weight count) N=Count;
run;
```

Notice that the COMPLETETYPES option forces all possible combinations of the two variables to be displayed, even if no data exists for a particular crossing. The template can be simplified as shown in the following because the bin midpoints are equally spaced integers.

```
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;
```

For this selection of bin widths, 6 bins were produced for HEIGHT and 10 for WEIGHT. Here is the output.



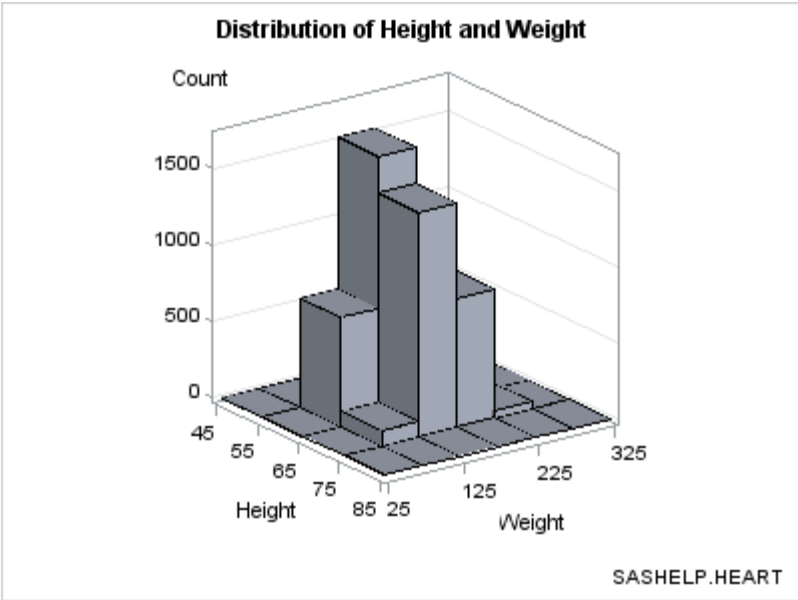
If you prefer to see the axes labeled with the bin endpoints rather the bin midpoints, you can use the ENDLABELS=TRUE setting on the BIHISTOGRAM3DPARM statement. Note that the ENDLABELS= option is independent of the XVALUES= and YVALUES= options.

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;
```

Here is the output.



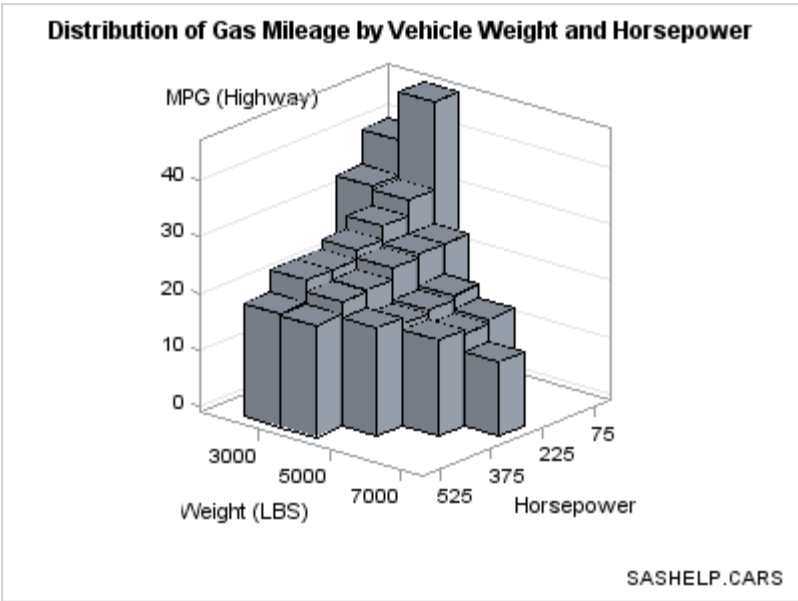
If you choose bin widths that are too small, "gaps" might be displayed among axis ticks values, which might cause the following message:

```
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 3-D data summarization of a response variable Z, which must have nonnegative values, by two numeric classification variables that are equally 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. Here is an example.

```
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;
```

Here is the output.



Producing Surface Plots

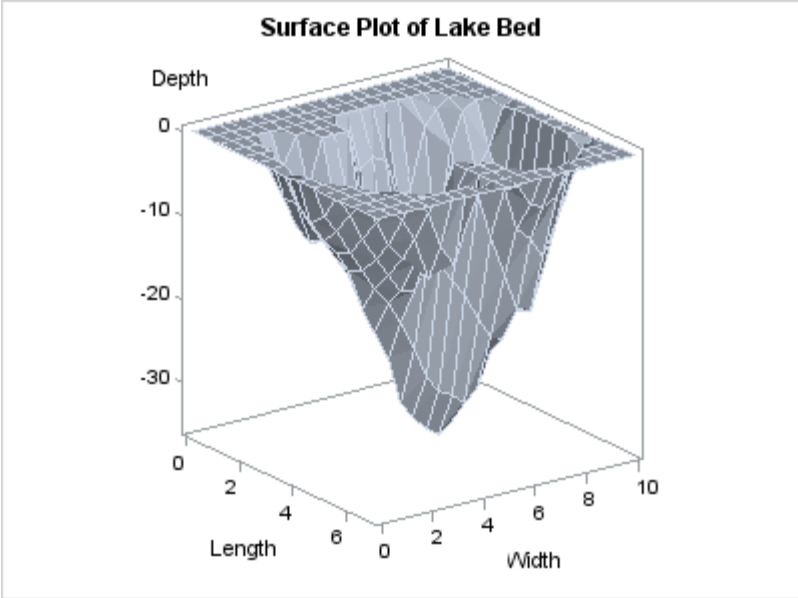
A surface plot shows points that are defined by three continuous numeric variables and connected with a polygon mesh. A polygon mesh is a collection of vertices, edges, and faces that defines the shape of a polyhedral object, which simulates the surface. In order for a surface to be drawn, the input 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 3-D 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 as shown in the following example.

```
proc template;
   define statgraph surfaceplotparm;
     begingraph;
       entrytitle "Surface Plot of Lake Bed";
       layout overlay3d / cube=false;
         surfaceplotparm x=length y=width z=depth;
       endlayout;
     endgraph;
   end;
 run;

 ods graphics / antialiasmax=5700;
 proc sgrender data=sashelp.lake template=surfaceplotparm;
 run;
```

Here is the output.



The SURFACEPLOTPARM statement assumes that the response and 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. The sort direction for Y should be ascending. The sort direction of X can be either ascending or descending.

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. Here is a partial listing of the Sashelp.Lake data set.

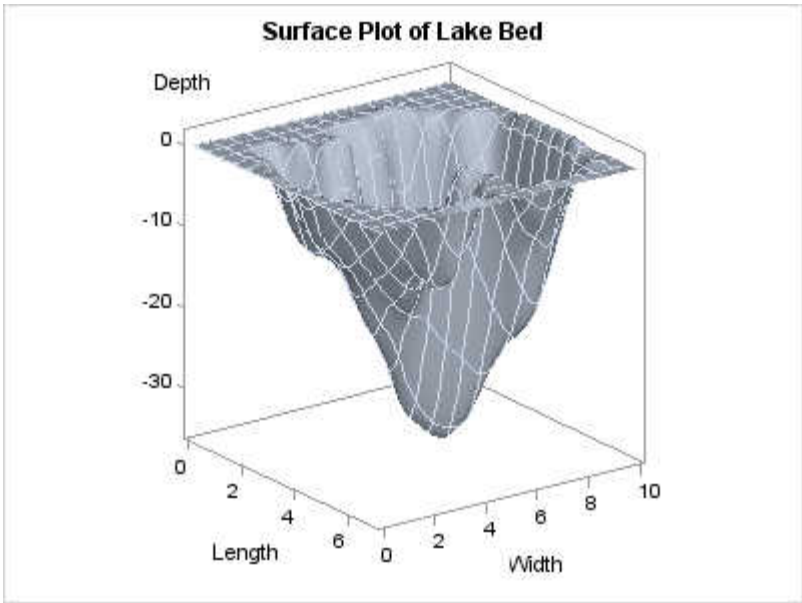
Obs	Width	Length	Depth
1	0	0.0	0.00000
2	0	0.5	0.00000
3	0	1.0	0.00000
4	0	1.5	-0.00598
5	0	2.0	0.00000
6	0	2.5	0.00000
7	0	3.0	0.00000
8	0	3.5	-0.21287
9	0	4.0	0.00000
10	0	4.5	0.00000
11	0	5.0	0.00000
12	0	5.5	-0.00299
13	0	6.0	0.00000
14	0	6.5	0.00000
15	0	7.0	0.00000
…			

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.

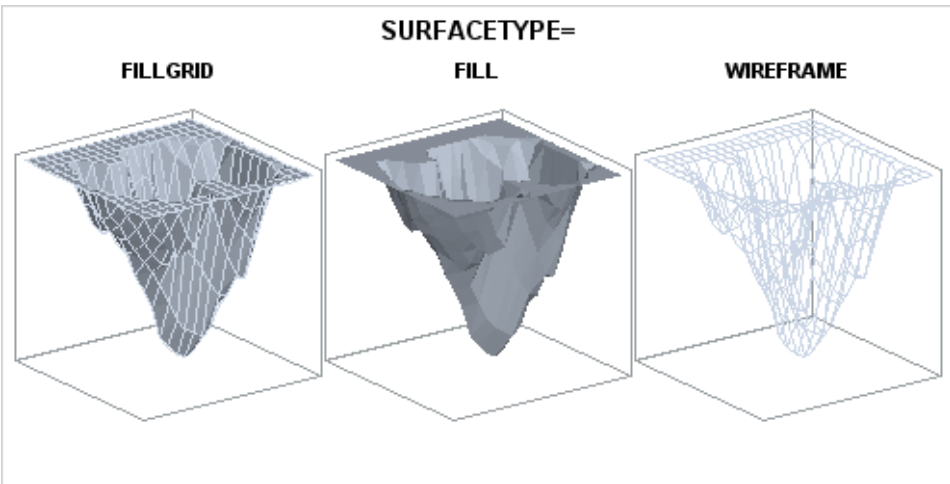
```
proc g3grid data=sashelp.lake out=spline;
  grid width*length = depth / naxis1=75 naxis2=75 spline;
run;

proc sgrender data=spline template=surfaceplotparm;
run;
```

By increasing the grid size and specifying a SPLINE interpolation, a smoother surface is rendered. Here is the output.



The SURFACETYPE= option offers three different types of surface rendering:

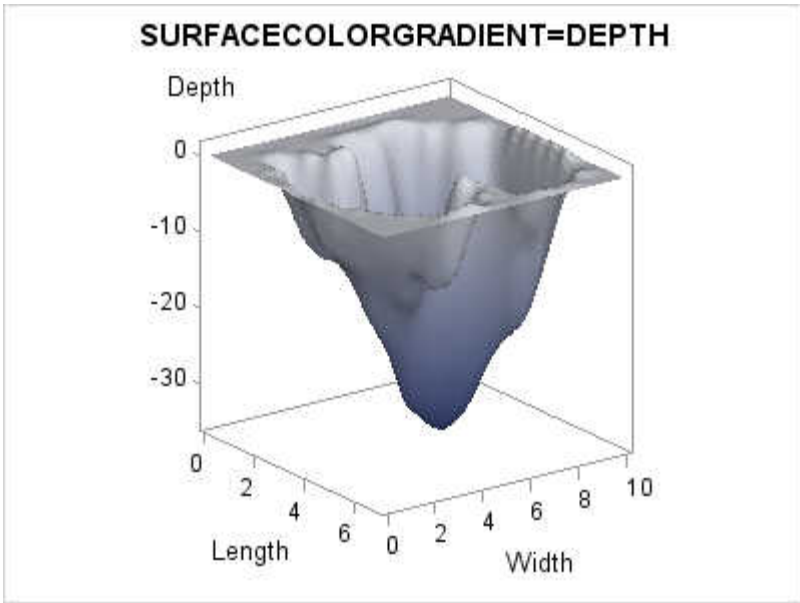
FILLGRID	a filled surface with grid outlines (the default)
FILL	a filled surface without grid outlines
WIREFRAME	an unfilled (see through) surface with grid outlines



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 column.

```
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.

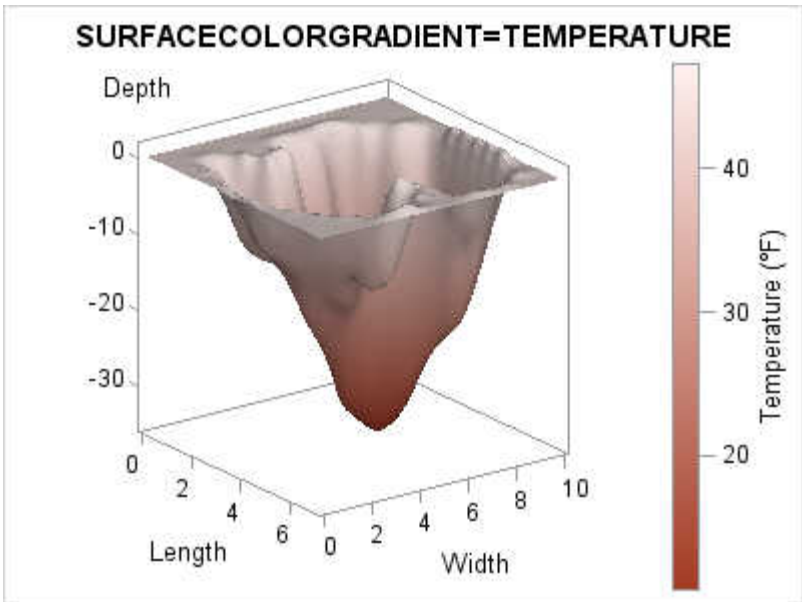


Using Color to Show an Additional Response Variable. The SURFACECOLORGRADIENT= option does not have to use the Z= variable. In the next example, another variable, TEMPERATURE is used.

```
ods escapechar="^"; /* Define an escape character */
```

```
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 (^{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;
```

Here is the output.



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 20, “Adding Legends to Your Graph,” on page 337.

Managing Axes in OVERLAY3D Layouts

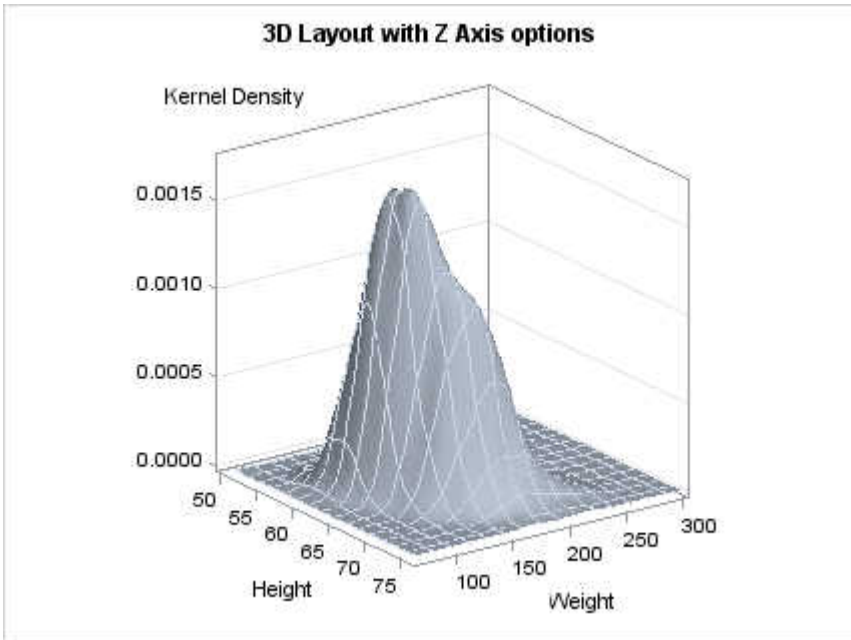
Axes for the OVERLAY3D layout are similar to axes for the OVERLAY layout, although the following exceptions apply to OVERLAY3D layouts:

- An additional ZAXISOPTS=( ) option is available for managing the Z axis.
- All three axis types can be either LINEAR or TIME. A DISCRETE or LOG 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 block 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;
```

Here is example output.



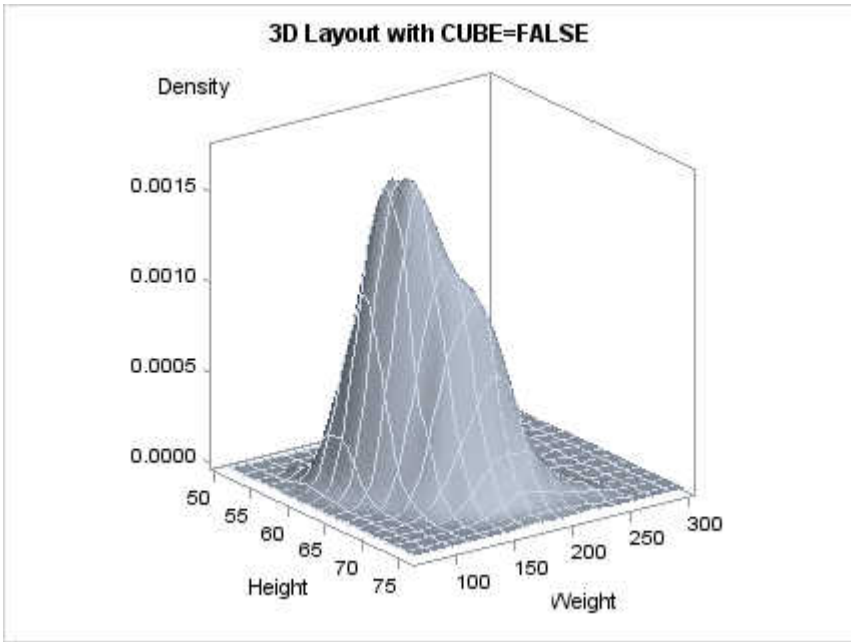
Display Features of the OVERLAY3D Layout

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. The following example shows you how to disable the lines.

```
proc template;
  define statgraph nocube;
    begingraph;
      entrytitle "3D Layout with CUBE=FALSE";
      layout overlay3d / cube=false;
        surfaceplotparm x=height y=weight z=density;
      endlayout;
    endgraph;
  end;
run;

ods graphics / antialiasmax=3600;
proc sgrender data=sashelp.gridded template=nocube;
run;
```

Here is the output.

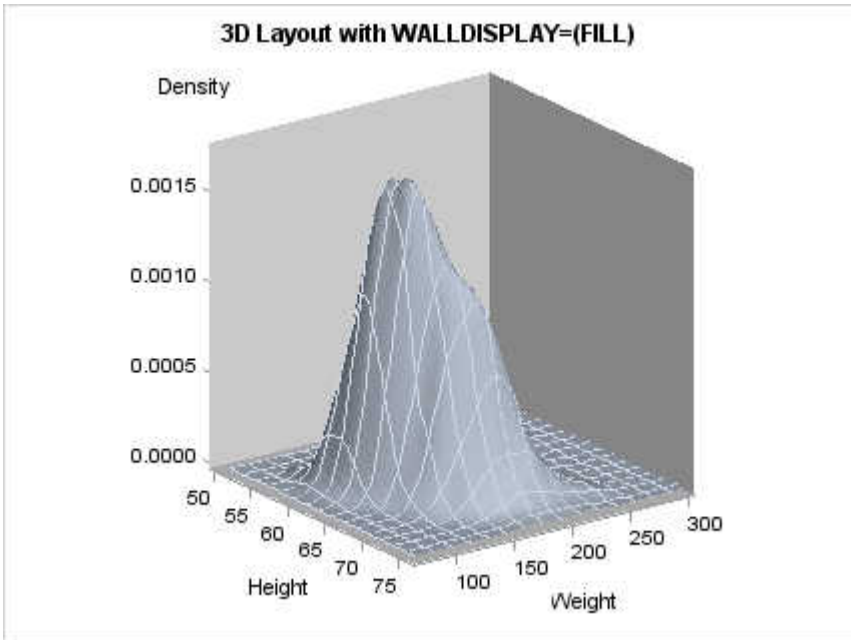


Displaying a Fill in the Graph Walls

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 as shown in the following layout block.

```
layout overlay3d / cube=false
  walldisplay=(fill);
  surfaceplotparm x=height y=weight
    z=density;
endlayout;
```

When filled, the wall lighting is adjusted to give a 3-D effect, based on the graph viewpoint. Here is example output.



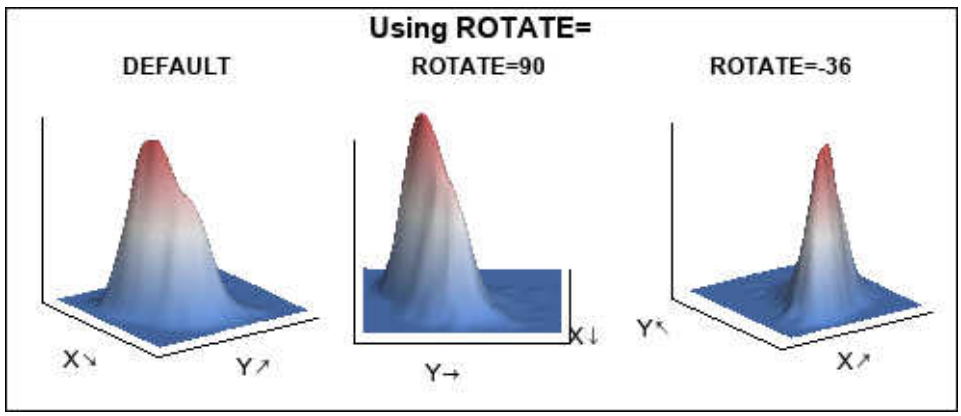
Defining a Viewpoint

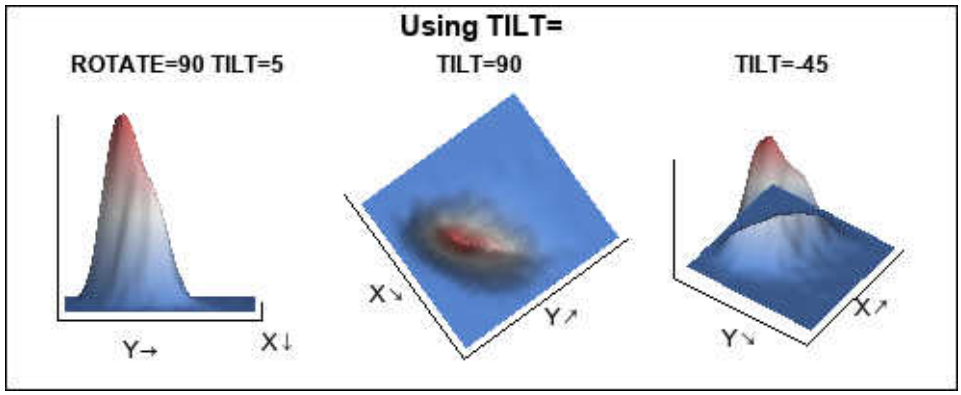
Representing a 3-D graph statically in two dimensions often obscures details that are better viewed from a different viewpoint. Three options on the LAYOUT OVERLAY3D statement can be independently set to obtain a different viewpoint.

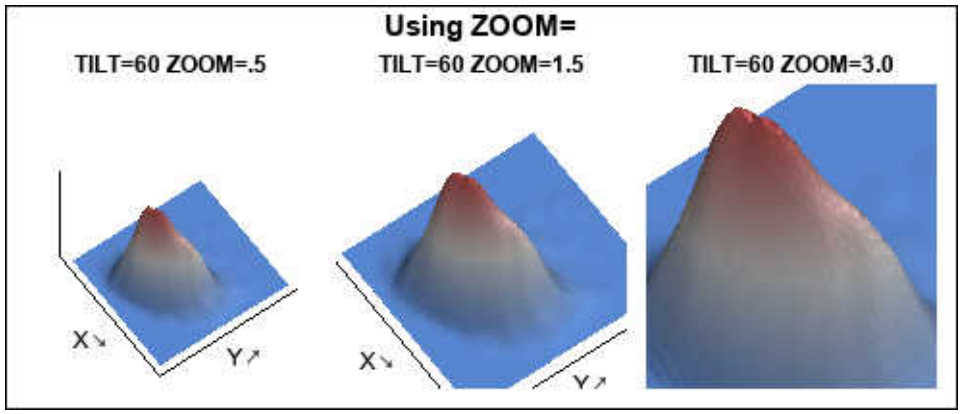
Option	Value Range	Default	Description
ROTATE=	-360 to 360	54	Specifies the angle of rotation. Rotation is measured in a clockwise direction about a

Option	Value Range	Default	Description
			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.






Using ZOOM=
TILT=60 ZOOM=.5
TILT=60 ZOOM=1.5
TILT=60 ZOOM=3.0
X↘
Y↗
X↘
Y↗

14

Creating Gridded Graphs Using the GRIDDED Layout

The LAYOUT GRIDDED Statement

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 about the LATTICE layout, see Chapter 15, “Creating Lattice Graphs Using the LATTICE Layout,” on page 221.

Typical applications of GRIDDED layouts are to create:

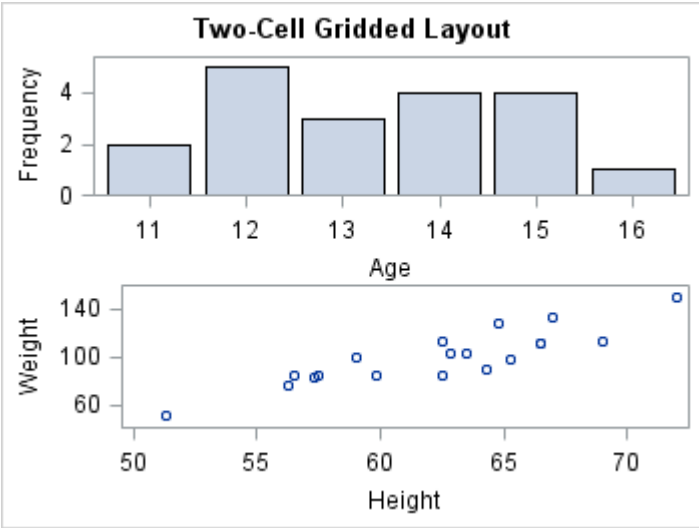
- a table of text, such as an inset (discussed in detail in Chapter 21, “Adding Insets to Your Graph,” on page 383)

- 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 standalone 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 category=age;
        scatterplot x=height y=weight;
      endlayout;
    endgraph;
  end;
run;

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

Here is the output.



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 standalone 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:

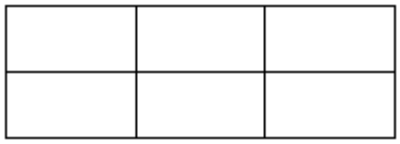
Grid Specification Code	Description

`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.

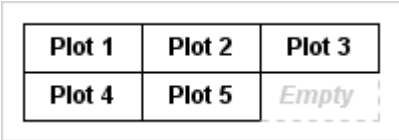
If you are willing to have an empty cell in the grid, you could use a 2x3 or a 3x2 grid:

```
layout gridded / columns=3;
endlayout;
```



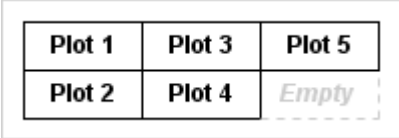
By default, the layout uses the ORDER=ROWMAJOR setting to populate grid cells. This specification essentially means "fill in all cells in the top row (starting at the top left) and then continue to the next row below." COLUMNS=1 by default when ORDER=ROWMAJOR, so you must specify an alternative setting 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 gridded / rows=2 order=columnmajor;
  /* plot1 definition */
  /* plot2 definition */
  /* plot3 definition */
  /* plot4 definition */
  /* plot5 definition */
endlayout;
```

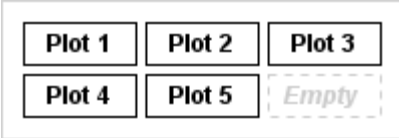


Setting Gutters

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;
```



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 215.

Defining Cells

Two valid techniques are available for indicating the contents of a cell:

Technique	Example	Advantages	Disadvantages
standalone plot statement or text statement	`scatterplot x= y=;`	simplicity	cannot have overlays cannot adjust axes, borders, or backgrounds (these are layout options)

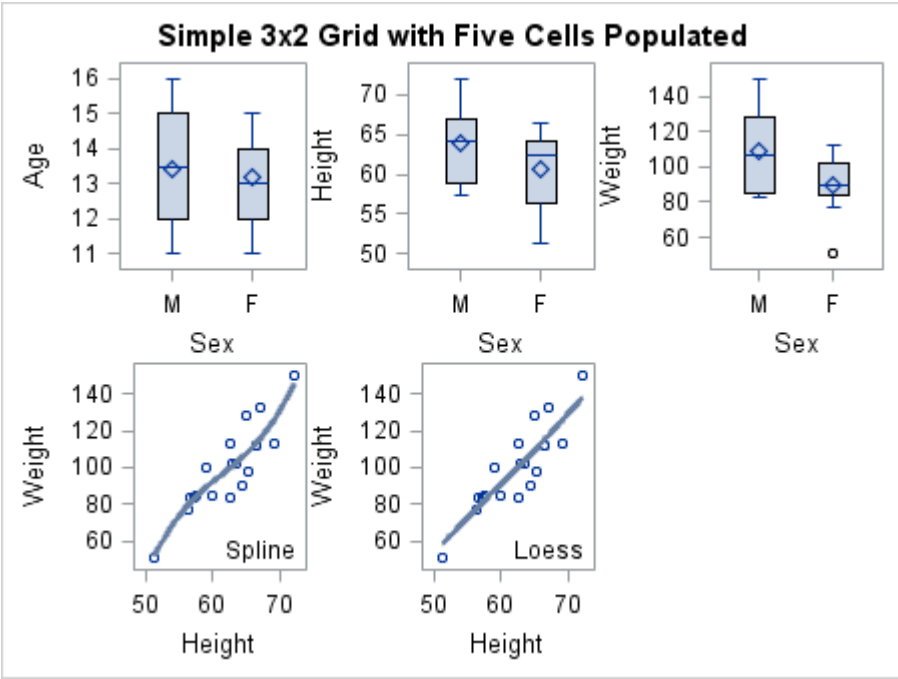
layout block	`layout overlay;`
	`  scatterplot x= y=;`
	`  seriesplot x= y=;`
	`endlayout;`
		cell can contain a complex plot
		axes can be adjusted
		other layout types can be used
			more complexity

Here is a simple example.

```
proc template;
  define statgraph celldefine;
    begingraph;
      entrytitle "Simple 3x2 Grid with Five Cells Populated";
      layout gridded / columns=3 rows=2;
        /* 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;
            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;
      endgraph;
    end;
  run;

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

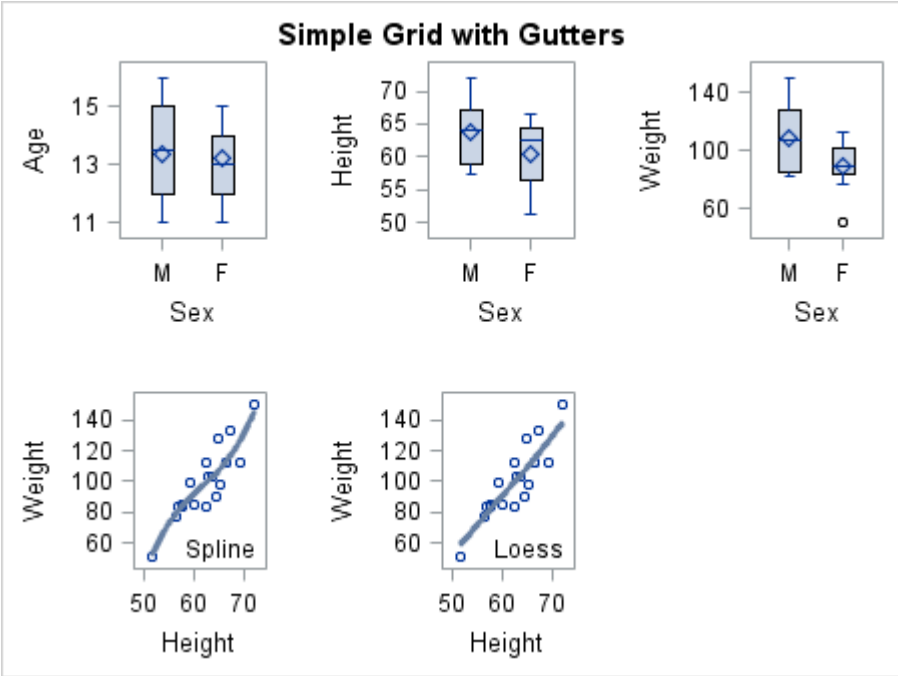
Here is the output.



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;
```

Here is example output.



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

One of the most common applications of the GRIDDED layout is to build a table of text or statistics similar to the following using nested ENTRY statements.

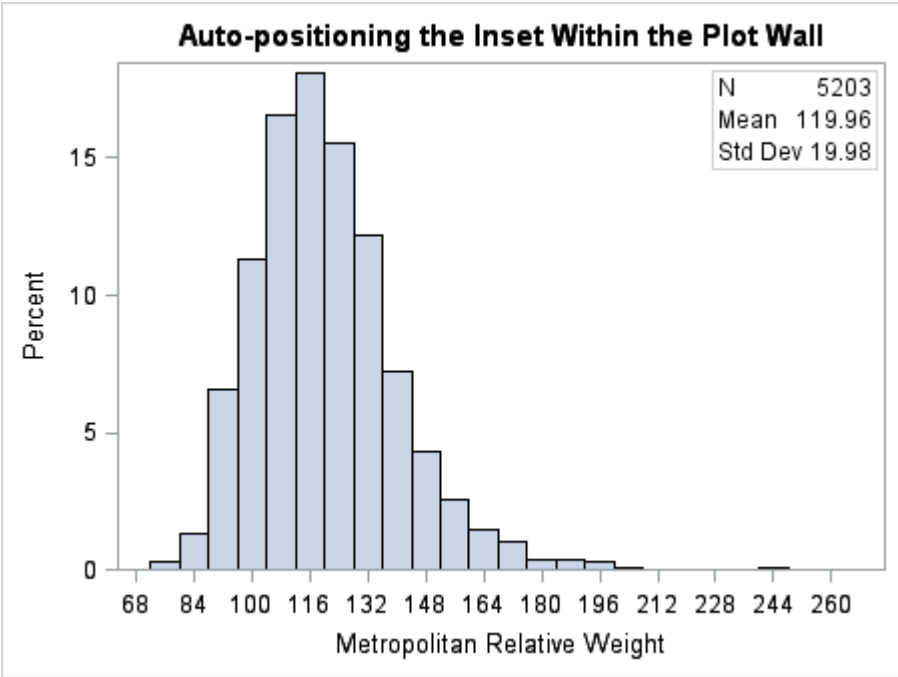
N	5203
Mean	119.96
Std Dev	19.98

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;
```

Here is the output.



Tables like this can be organized many different ways. For more information about these techniques, see Chapter 19, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 317 for details about ENTRY statements. In this example, the values for the statistics in the table are hardcoded. Obviously, you would prefer that the statistics values be calculated in the template. Chapter 21, “Adding Insets to Your Graph,” on page 383 shows how these values can be computed in the template or passed to the template using dynamic variables or macro variables.

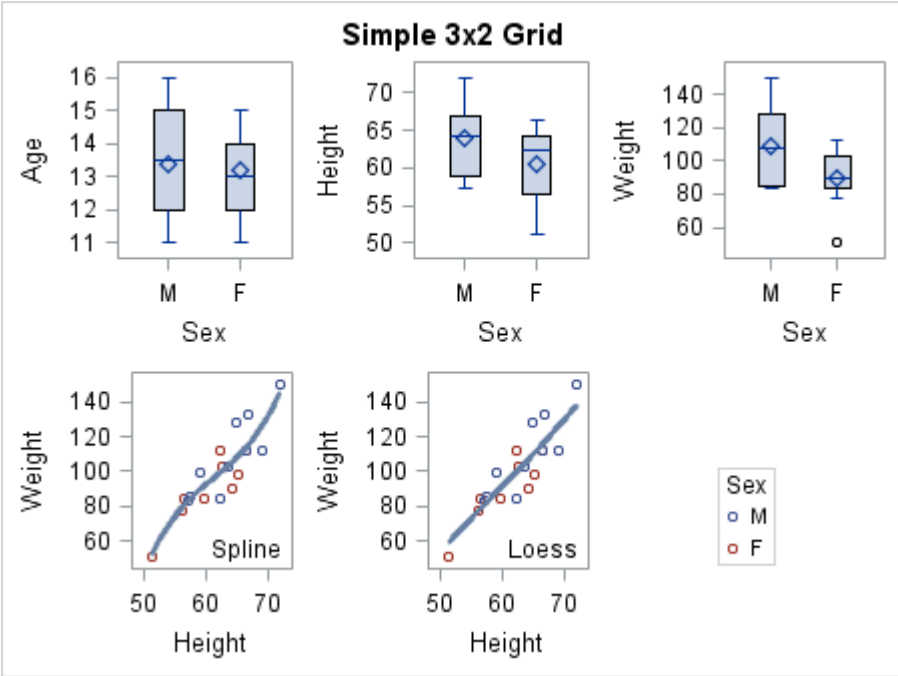
Sizing Issues

Row and Column Sizes

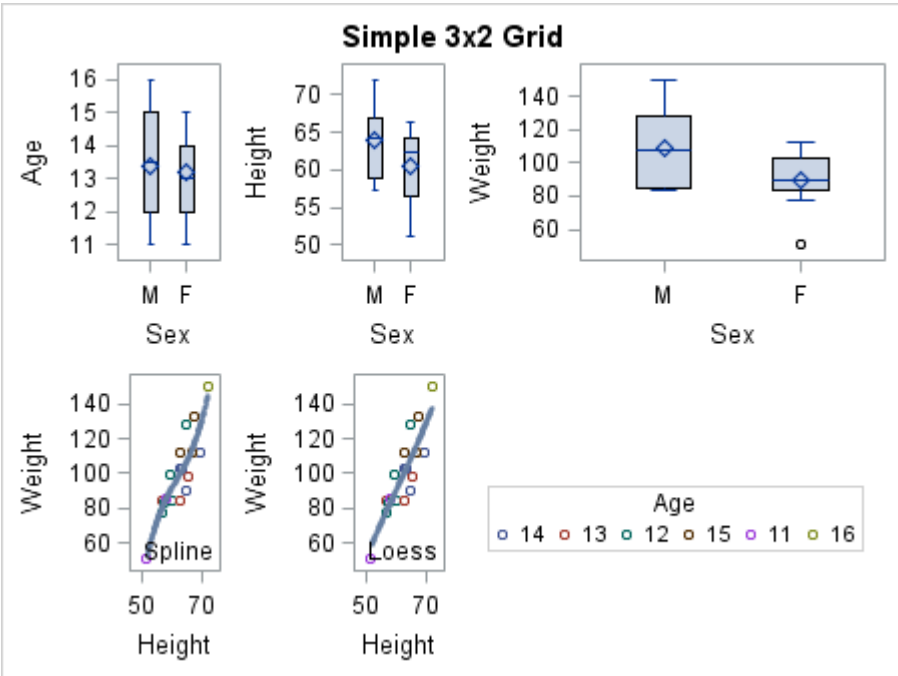
Unlike the LATTICE layout, the GRIDDED layout offers no way to control column sizes or row sizes. These sizes are determined by the contents of the cells. If only plots are used in the cells, the grid is partitioned equally based on the graph size. However, any individual cell in the grid might contain a legend or text. Consider the next two examples, in which the sixth cell of the grid is populated with a legend.

```
proc template;
  define statgraph celldefine;
    begingraph;
      entrytitle "Simple 3x2 Grid";
      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;
        discretelegend "scatter" / title="Sex";
      endlayout;
    endgraph;
  end;
run;

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

Here is the output.



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, the following 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.



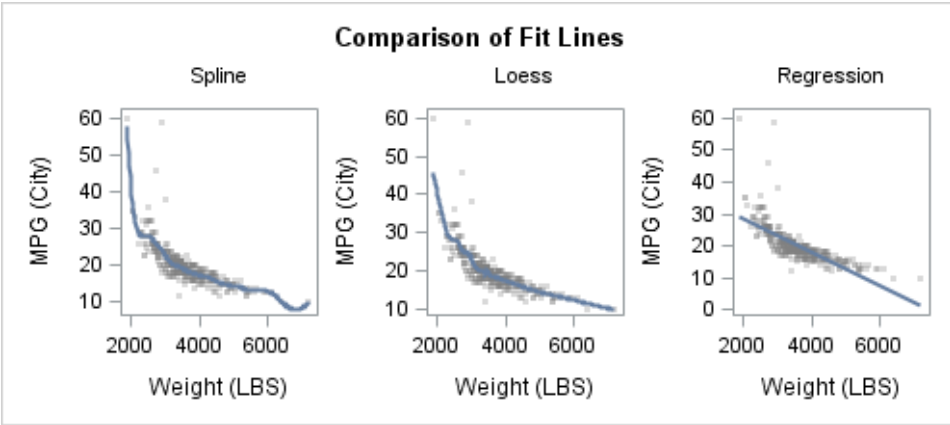
The same thing might happen when ENTRY statements with lengthy strings are used in cells. Because of this behavior, you should consider using a LATTICE layout whenever you want to enforce uniform or user-defined column widths and row heights for the grid, regardless of cell contents. If this layout were 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 the following three-cell GRIDDED layout example.

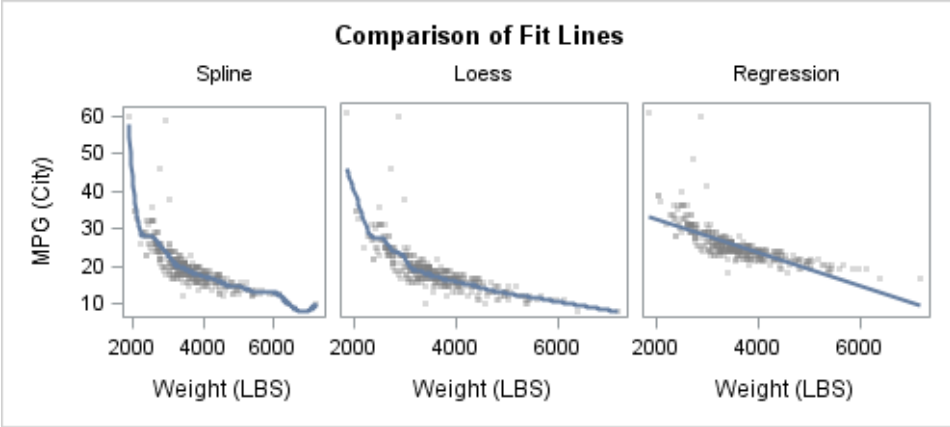
Example Code 14.1 *Three-Cell Gridded Layout Example*

```
proc template;
  define statgraph fitcompare;
    begingraph / designwidth=495 designheight=220;
      entrytitle "Comparison of Fit Lines";
      layout gridded / columngutter=5 columns=3 rows=1;
         /* Cell 1 */
         layout overlay;
           entry "Spline" / location=outside valign=top;
           scatterplot x=weight y=mpg_city /
             markerattrs=(size=3px symbol=circlefilled color=gray )
             datatransparency=.7;
           pbsplineplot x=weight y=mpg_city;
         endlayout;
         /* Cell 2 */
         layout overlay;
           entry "Loess" / location=outside valign=top;
           scatterplot x=weight y=mpg_city /
             markerattrs=(size=3px  symbol=circlefilled color=gray)
             datatransparency=.7;
           loessplot x=weight Y=mpg_city;
         endlayout;
         /* Cell 3 */
         layout overlay;
           entry "Regression" / location=outside valign=top;
           scatterplot x=weight y=mpg_city /
             markerattrs=(size=3px symbol=circlefilled color=gray)
             datatransparency=.7;
           regressionplot x=weight y=mpg_city;
         endlayout;
      endlayout;
    endgraph;
  end;
run;

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

Three OVERLAY layouts define the cell contents. Here is the output.



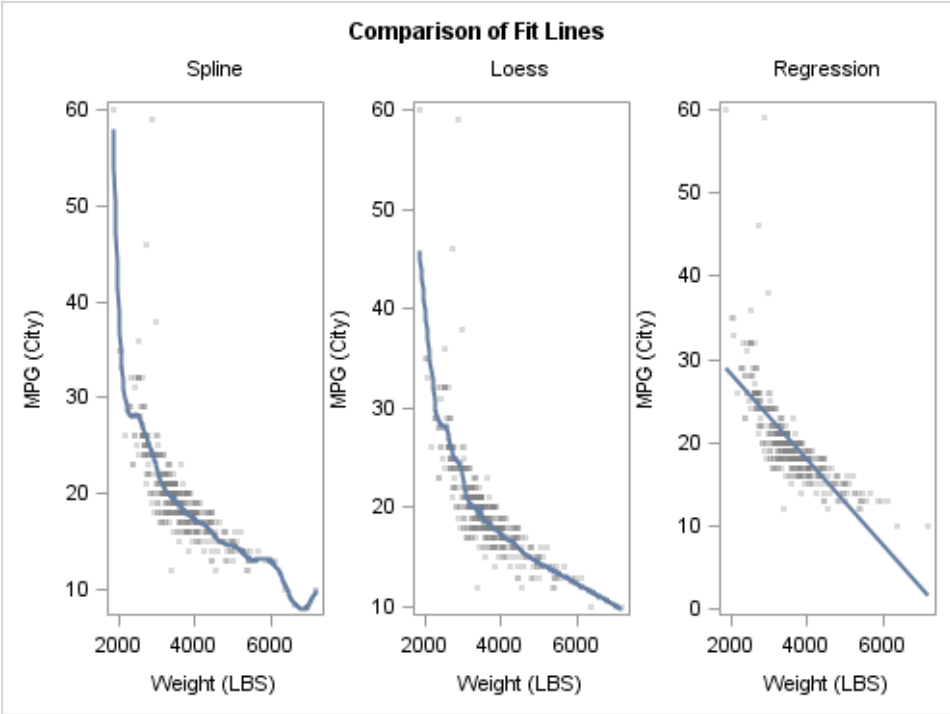
Because the Y axes are duplicated across cells, you might try to conserve space by turning off the Y axes for the second and third cells. You can do this by adding the YAXISOPTS=(DISPLAY=NONE) option to the LAYOUT OVERLAY statement for the second and third cells. Here is the result.



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. See Example Code 15.1 on page 239.

Adjusting Graph Size

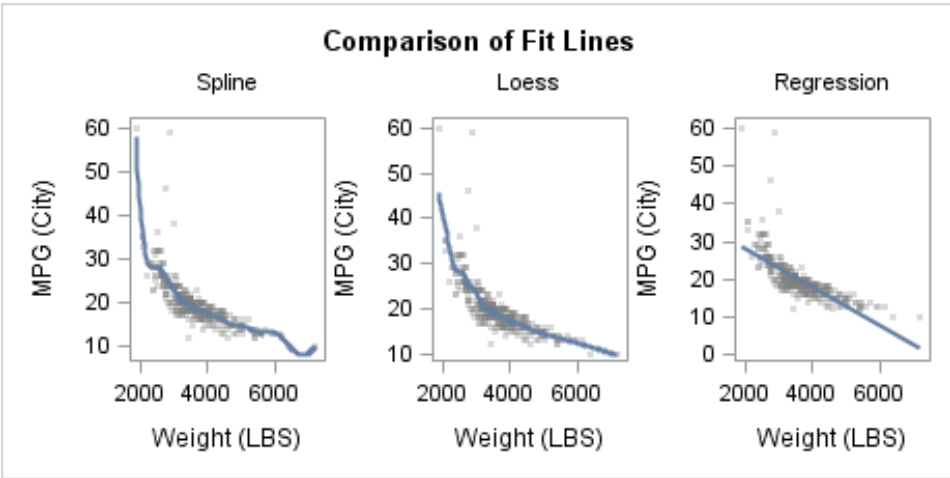
When defining the grid size, you will generally have some idea of a good overall aspect ratio for the graph. For example, if the graph in Example Code 14.1 on page 217 is rendered using the default size of 640 pixels by 480 pixels, the graph has an aspect ratio of 4:3, which looks as follows:



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 as shown in Example Code 14.1 on page 217.After some experimentation, you might decide that a 2:1 aspect ratio for this graph looks good. Here is the modified BEGINGRAPH statement.

```
begingraph / designwidth=460px designheight=230px;
```

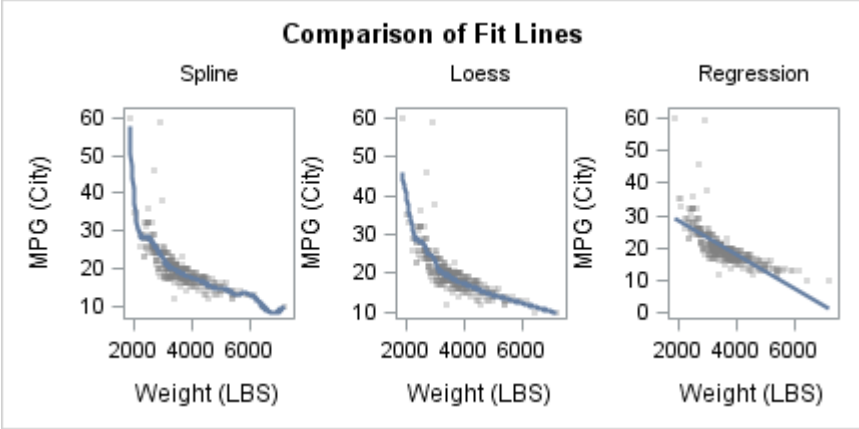
Here is the output.



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=430px;
proc sgrender data=sashelp.cars template=fitcompare;
run;
```

Here is the result.



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.

15

Creating Lattice Graphs Using the LATTICE 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

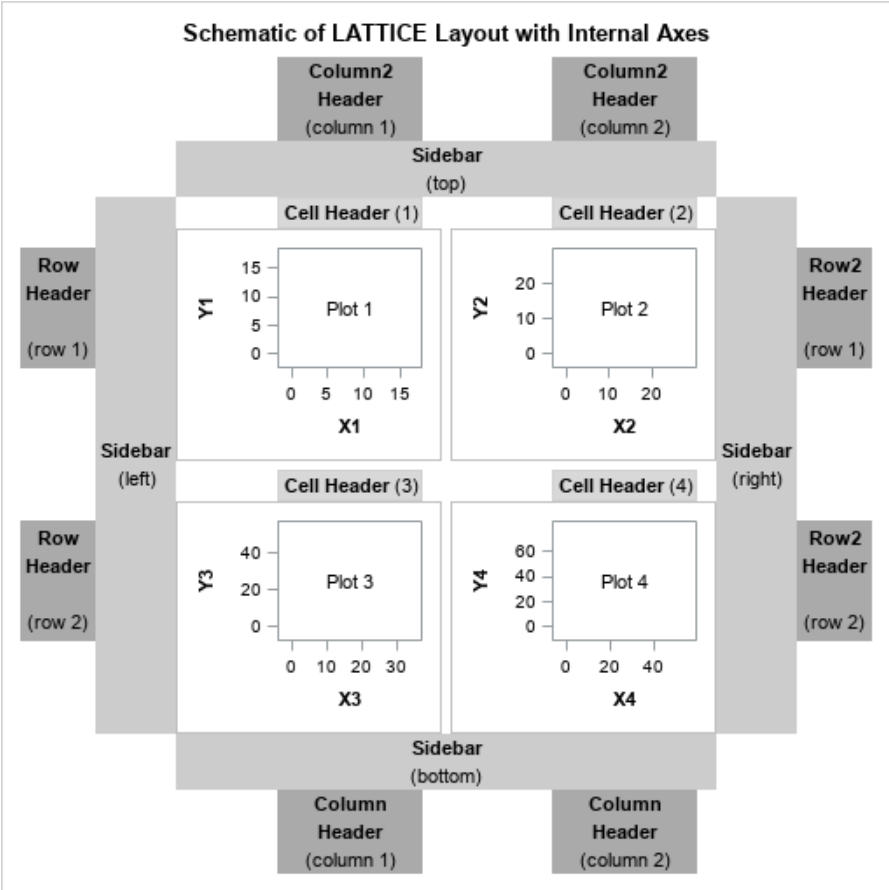
GRIDDED layout because it can create a grid of heterogeneous plots. However, the LATTICE has much more functionality and supports the following:

- 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 15.1 on page 222 shows a four-cell lattice (two rows and two columns). It was produced with a LAYOUT LATTICE statement 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 can 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 15.1 *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.

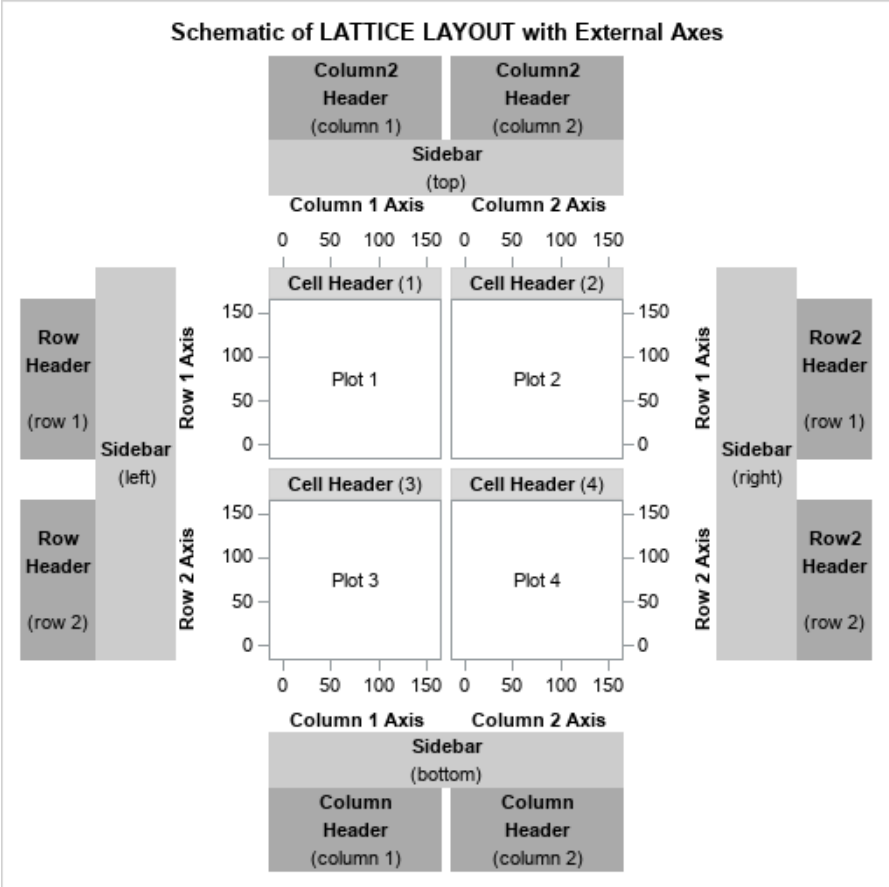
Figure 15.2 on page 224 shows how the lattice would look if you used additional options to externalize the internal axes by consolidating them into common row and column axes. The figure shows both row and column axes, but you could externalize the axes only for the rows, or only for the columns. When axes are external to the cells, the data ranges that are displayed for the plots are unified by taking the minimum of all data minima and the maximum of all data maxima from the affected 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 15.2 on page 224.)
- 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 15.2 on page 224.)

By default, row and column 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 row and column axes use less space and result in larger plot areas than internal axes. (Compare Figure 15.2 on page 224 with Figure 15.1 on page 222, which is the same size.)

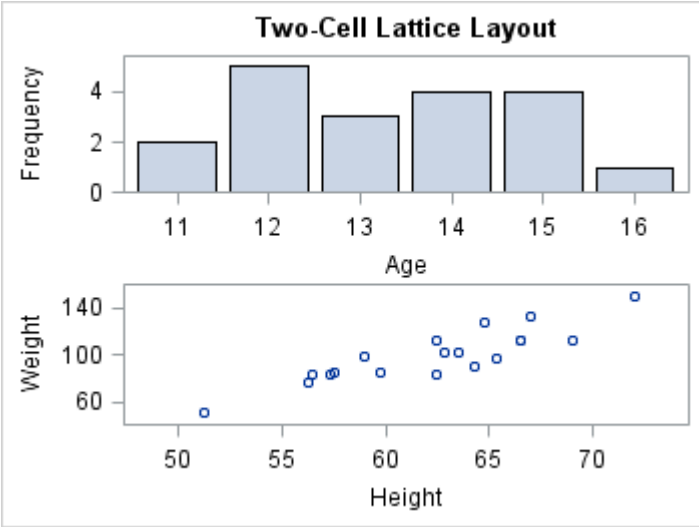
Figure 15.2 *LATTICE Layout with Row and Column Axes*



The following example shows a very simple LATTICE layout:

```
proc template;
  define statgraph intro;
    begingraph;
      entrytitle "Two-Cell Lattice Layout";
      layout lattice;
        barchart category=age;
        scatterplot x=height y=weight;
      endlayout;
    endgraph;
  end;
run;

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

Here is the output.



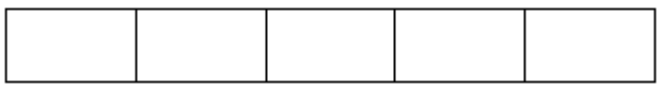
In a LATTICE layout, each plot statement is considered independent and is placed in a separate cell. When no grid size is provided, the default layout creates a graph with one column of cells, and it allots each cell the same amount of space. The number of rows in the grid is determined by the number of standalone plot statements in the LAYOUT LATTICE block.

Defining a Basic Lattice

Setting Grid Dimensions

Assume you want a grid of five plots. Your first step is to decide how many columns and rows you want and whether you want to permit empty cells in the grid. If you 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).

To specify grid size, 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:



Lattice Specification Code	Description

`layout lattice / columns=5 rows=1;`
`  /* plot definitions */`
`endlayout;`
	Specify the number of columns and the number of rows to explicitly set the grid size.

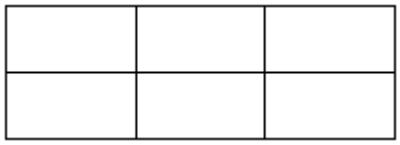
`layout lattice / order=columnmajor` `  rows=1;`	Specify only one grid row and ORDER=COLUMNMAJOR. In this case, there

Lattice Specification Code	Description
`/* plot definitions */` `endlayout;`	are 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;`
	Specify only the number of columns. This places 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:

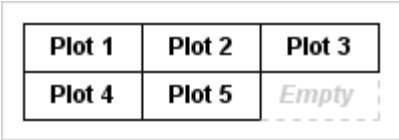
```
layout lattice / columns=3 rows=2;
endlayout;
```



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.

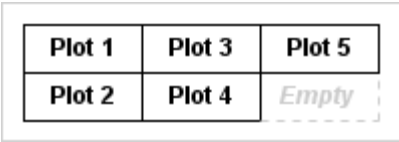
By default, the LATTICE layout uses the ORDER=ROWMAJOR setting, which populates the grid cells left to right starting at the top left cell and continuing to the next row below:

```
layout lattice / columns=3 rows=2;
  /* plot1 definition */
  /* plot2 definition */
  /* plot3 definition */
  /* plot4 definition */
  /* plot5 definition */
endlayout;
```



Alternatively, you can specify ORDER=COLUMNMAJOR, which populates the grid cells top to bottom starting at the top left cell and continuing to the top of the next column to the right:

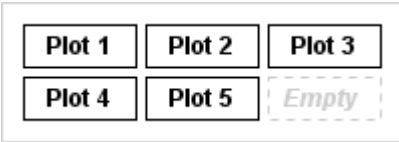
```
layout lattice / columns=3 rows=2 order=columnmajor;
  /* plot1 definition */
  /* plot2 definition */
  /* plot3 definition */
  /* plot4 definition */
  /* plot5 definition */
endlayout;
```



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;
```



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 239.

Populating Cells

The following table lists several techniques for populating cells.

Technique	Example	Advantages	Disadvantages
standalone plot statement	`scatterplot x= y=;`	simplicity	cannot adjust axes cannot have overlays cannot have cell headers

layout block	`layout overlay;`
	`  scatterplot x= y=;`
	`  seriesplot x= y=;`
		cell can contain a complex plot	cannot have cell headers

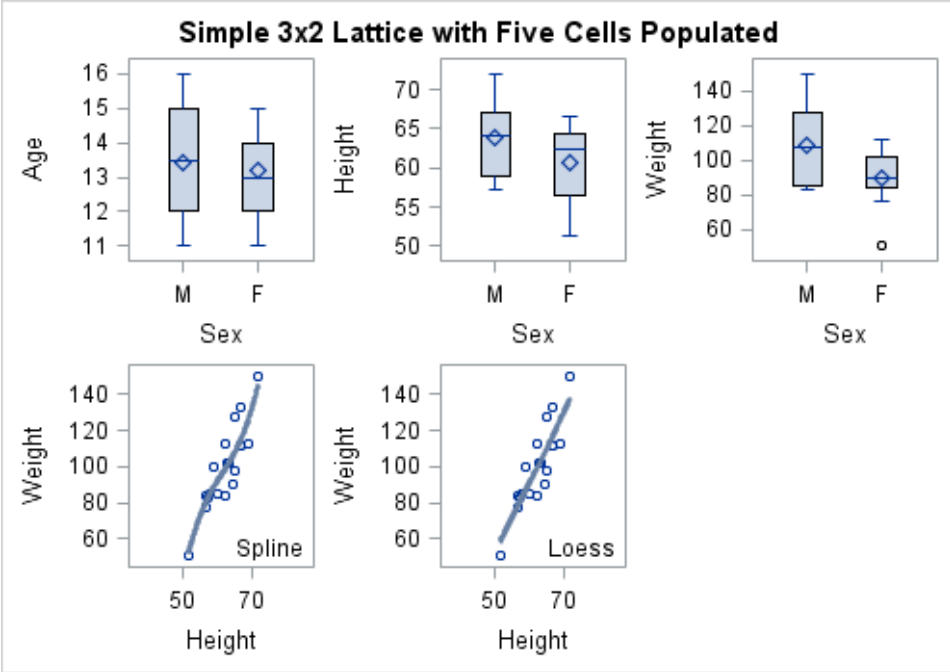
Technique	Example	Advantages	Disadvantages
	`endlayout;`	axes can be adjusted other layout types can be used	

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

Here is a simple example:

```
proc template;
  define statgraph lattice;
    begingraph;
      entrytitle "Simple 3x2 Lattice with Five Cells Populated";
      layout lattice / 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;
          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;
    endgraph;
  end;
run;

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

Here is the output.



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.

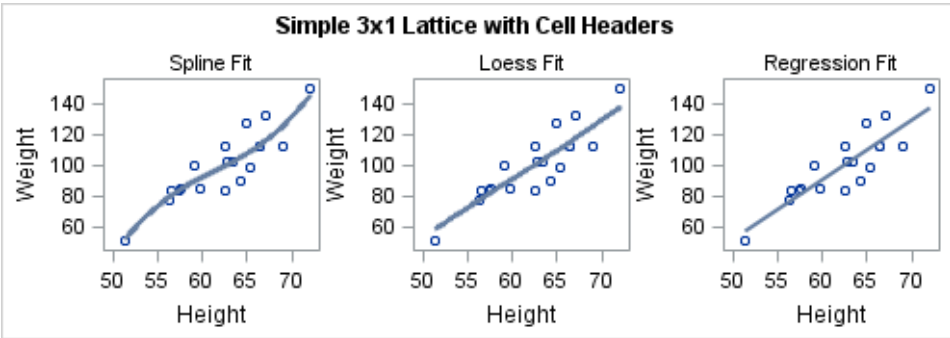
Adding Cell Headers

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 such as DISCRETELEGEND. Here is an example.

```
proc template;
  define statgraph lattice2;
    begingraph / designwidth=495px designheight=230px;
      entrytitle "Simple 3x1 Lattice with Cell Headers";
      layout lattice / columns=3 rows=1 columngutter=5;
        /* 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;
    endgraph;
  end;
run;

proc sgrender data=sashelp.class template=lattice2;
run;
```

Here is the output.



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.

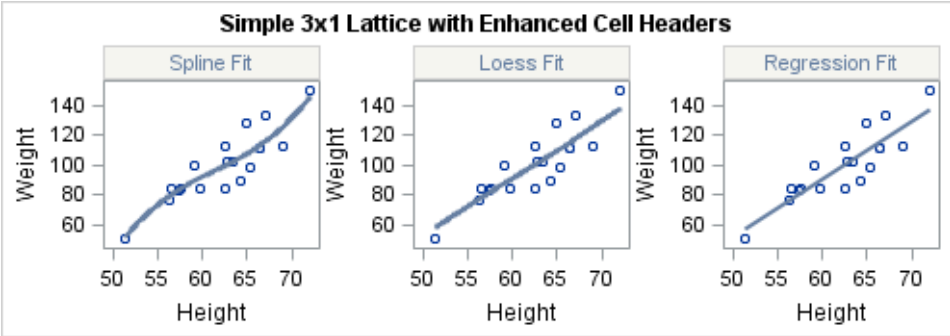
You can further enhance the cell header by making the GRIDDED layout's background opaque and setting a background color for it. To ensure that the color remains coordinated with the current style, you could choose any of several style elements that define light background colors, such as GraphHeaderBackground, GraphBlock, or GraphAltBlock. Note that several style elements set the GraphHeaderBackground color to be the same as the GraphBackground color. For styles such as 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 about ENTRY statements, see Chapter 19, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 317.

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;
```

Here is the result when this code is applied to all three cells.



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.

Managing Axes in LATTICE Layouts

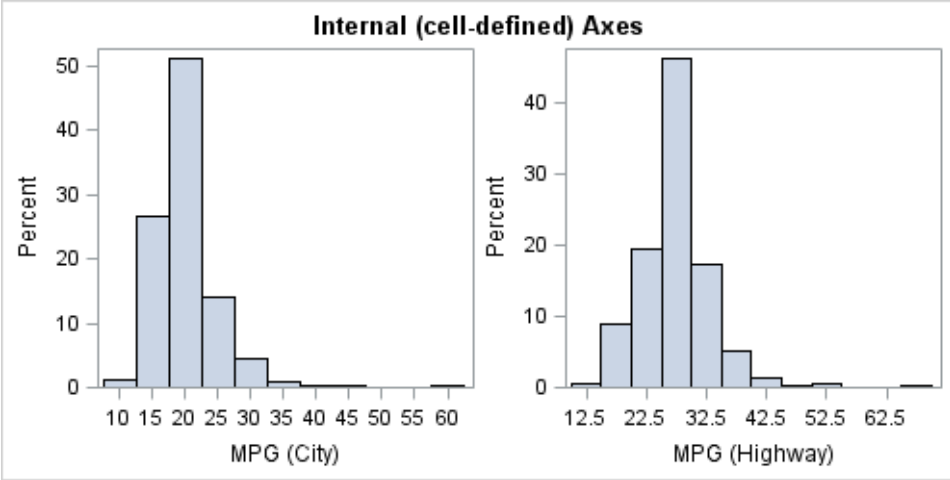
Internal Axes

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;

ods graphics / reset width=495px height=250px;
proc sgrender data=sashelp.cars template=internalaxes;
```

```
run;
```

Here is the output.



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 axis in each row of a lattice, use the COLUMNDATARANGE= option on the LAYOUT LATTICE statement. Likewise, to set a uniform scale on the Y axis in each row of the lattice, use the ROWDATARANGE= option. These options accept one of the following values:

DATA
: scales the axes independently for each cell. This value is the default.

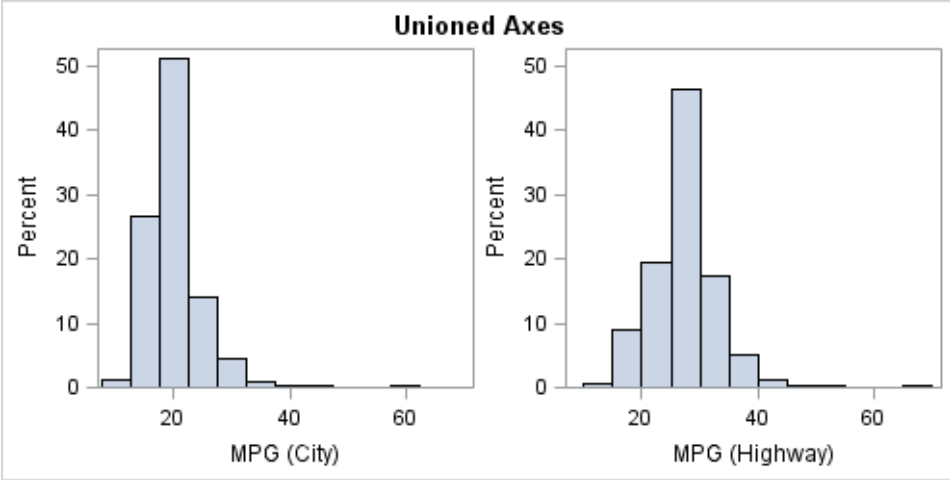
UNION
: finds the minimum of the data minima and the maximum of the data maxima, on a per-row or per-column basis, and sets this range on the appropriate axis for each cell in a row or column.

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.

The following LAYOUT LATTICE statement specifies UNIONALL for the X axis and UNION for the Y axis.

```
layout lattice / columns=2 columngutter=5px
  columndatarange=unionall
  rowdatarange=union;
  histogram mpg_city;
  histogram mpg_highway;
endlayout;
```

Here is example output.



Note: The default X axis for a histogram shows ticks at bin midpoints or at bin start and end points. If the histograms 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 axis so that the axis tick values can be uniform, even though they might not be at the bin midpoints or boundaries for all histograms.

The following 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.
- If a cell contains a LAYOUT OVERLAY3D or LAYOUT OVERLAYEQUATED statement, uniform axis ranges and external axes are not supported for that row or column.
- If you create a multipanel lattice and specify ROWDATARANGE=UNION, the axis range for each row might differ from panel to panel. The ROWDATARANGE=UNION option does not extend across panels, which means the axis range is computed on a per-row basis for each panel rather than across all of the panels.

Row and Column Axes

Specifying Row and Column Axes

Whenever axis scales have been unified for a row or a column, you can replace the internal cell axes in that row or column with a single common axis that is external to the cells.

To create a column X axis, use the following syntax:

COLUMNAXES;
    COLUMNAXIS / *options*;
    <COLUMNAXIS / *options*;>

ENDCOLUMNAXES;

To create a row Y axis, use the following syntax:

ROWAXES;
    ROWAXIS / *options*;
    <ROWAXIS / *options*;>
ENDROWAXES;

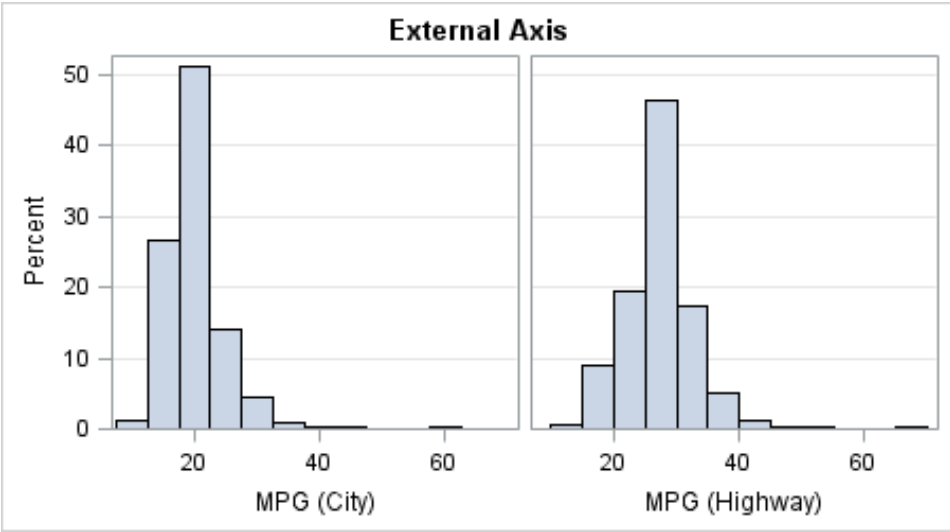
Within the axes blocks, specify as many COLUMNAXIS or ROWAXIS statements as there are columns or rows in the grid. The *options* that are available to each statement are similar to those that are available for the XAXISOPTS= and YAXISOPTS= options of a LAYOUT OVERLAY statement. The *options* that you specify can differ from statement to statement.

Note: When a row or column axis is used, all axis options in the layout statement for each cell in that row or column are ignored.

The following layout block externalizes the Y axes and displays Y-axis grid lines:

```
layout lattice / columns=2 columngutter=5px
  columndatarange=unionall
  rowdatarange=union;
  histogram mpg_city;
  histogram mpg_highway;
  rowaxes;
    rowaxis / griddisplay=on;
  endrowaxes;
endlayout;
```

Here is example output.

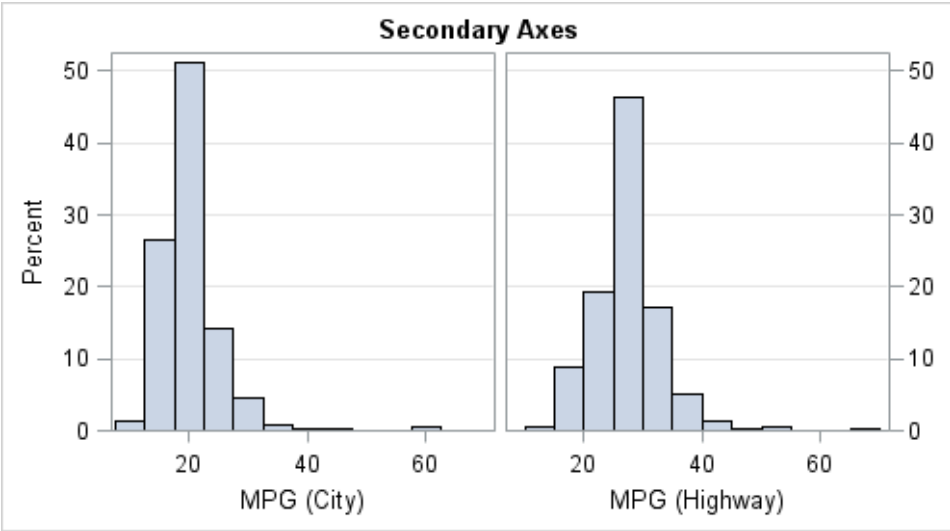


Displaying Row and Column Secondary Axes

The DISPLAYSECONDARY= option can be used in a ROWAXIS statement to display a row axis at the right of the lattice. It can be used in a COLUMNAXIS statement to display a column axis at the top of the lattice. These secondary axes are duplicates of the primary axis and are not truly independent axes. However, you can change the features that are displayed on the secondary axis. In the following LAYOUT LATTICE statement block, 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 in 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;
```

Here is example output.



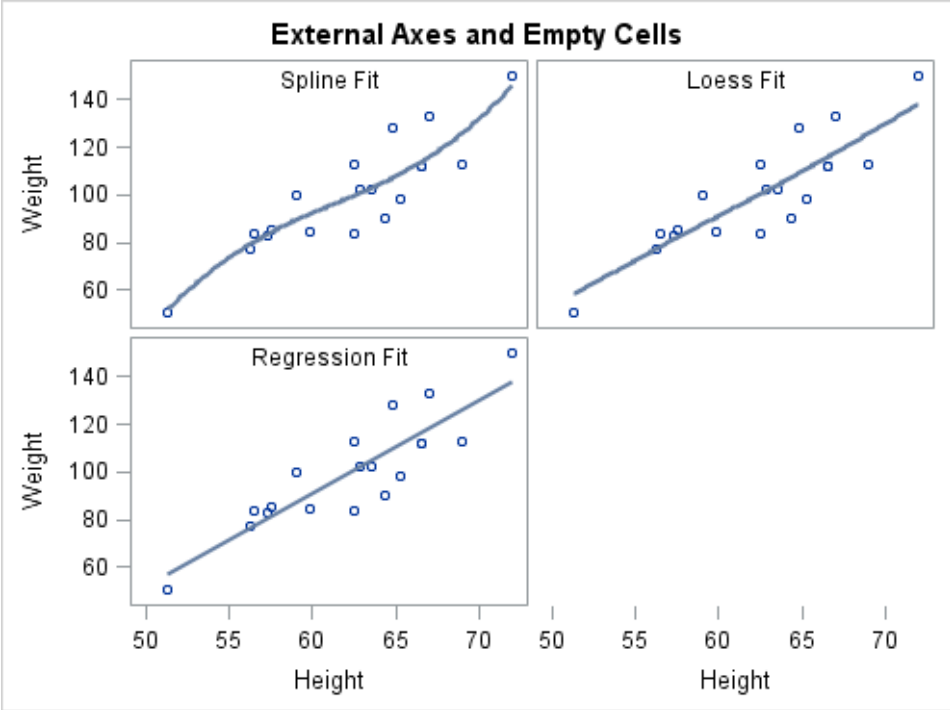
External Axes and Empty Cells

If a LATTICE layout generates empty cells and there are row and column axes, a row or column axis might be displayed near one or more of those empty cells. The following example shows the default case:

```
proc template;
  define statgraph skipemptycells1;
```

```
    begingraph;
      entrytitle "External Axes and Empty Cells";
      layout lattice / columns=2 rows=2
        rowgutter=5px columngutter=5px
        rowdatarange=unionall columndatarange=unionall;
        /* cell blocks 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;
    endgraph;
  end;
run;

proc sgrender data=sashelp.class template=skipemptycells1;
run;
```

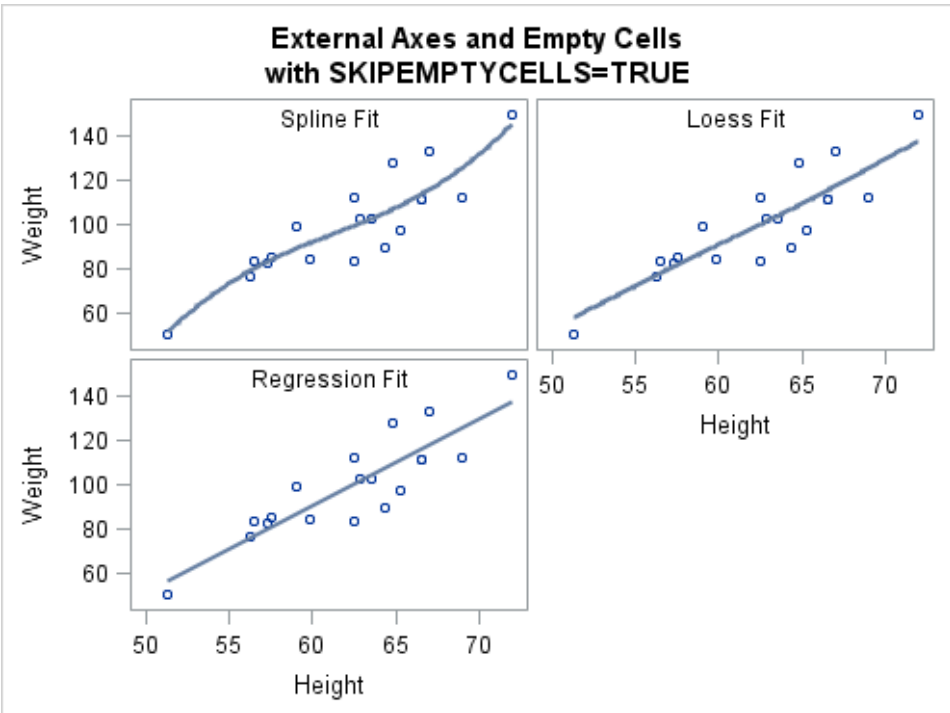
Here is the output.



Adding the SKIPEMPTYCELLS=TRUE option to the LAYOUT LATTICE statement as shown in the following example eliminates the space that is normally reserved for the empty cells.

```
layout lattice / columns=2 rows=2
  rowgutter=5px columngutter=5px
  rowdatarange=unionall columndatarange=unionall
  skipemptycells=true;
```

In this case, a column X axis that might be displayed near an empty cell is removed. Here is example output.



Adjusting the Sizes of Rows and Columns

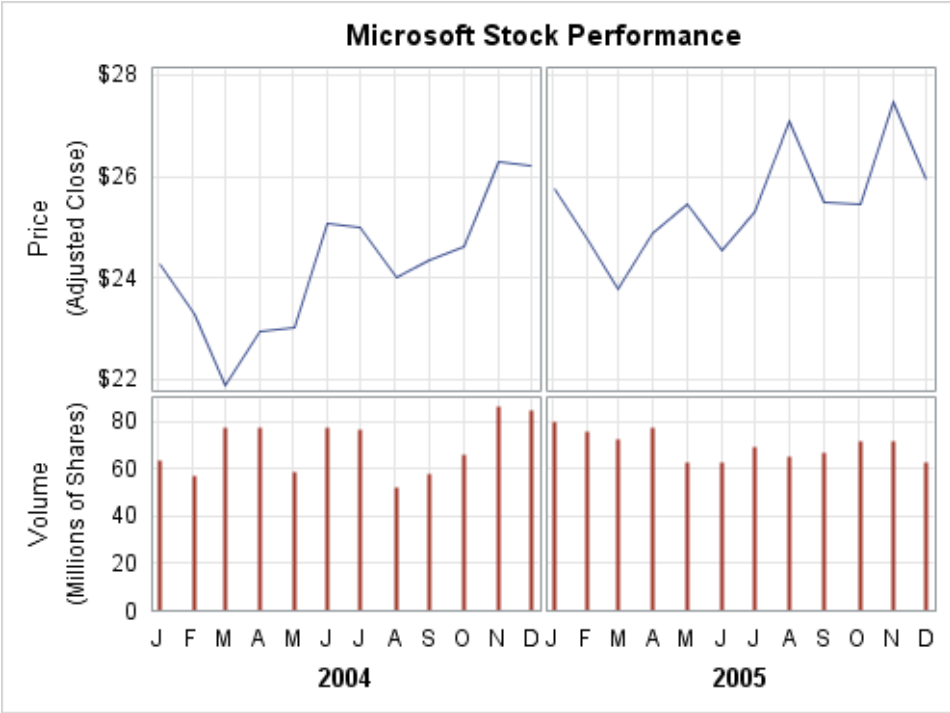
By default, the rows and columns of the lattice are of the same height and width. You can use the ROWWEIGHTS= and COLUMNWEIGHTS= options on the LAYOUT LATTICE statement to designate different row heights or column widths or both. Consider the following settings:

```
LAYOUT LATTICE /  ROW=2 COLUMNS=2
  ROWWEIGHTS=(.6 .4) COLUMNWEIGHTS=(.45 .65);
```

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. For example, in a traditional stock plot, the area devoted to price information is larger than the area devoted to the volume information. Here is an adjustment made to the row heights for the graph in Figure 15.7 on page 251:

```
layout lattice / columns=2 rows=2 rowweights=(.6 .4)
         rowdatarange=union columndatarange=union
         rowgutter=3px columngutter=3px;
```

Figure 15.3 *LATTICE with Adjusted Row Heights*



For another example, see "Example 5: Lattice with Custom Row Sizing" on page 251.

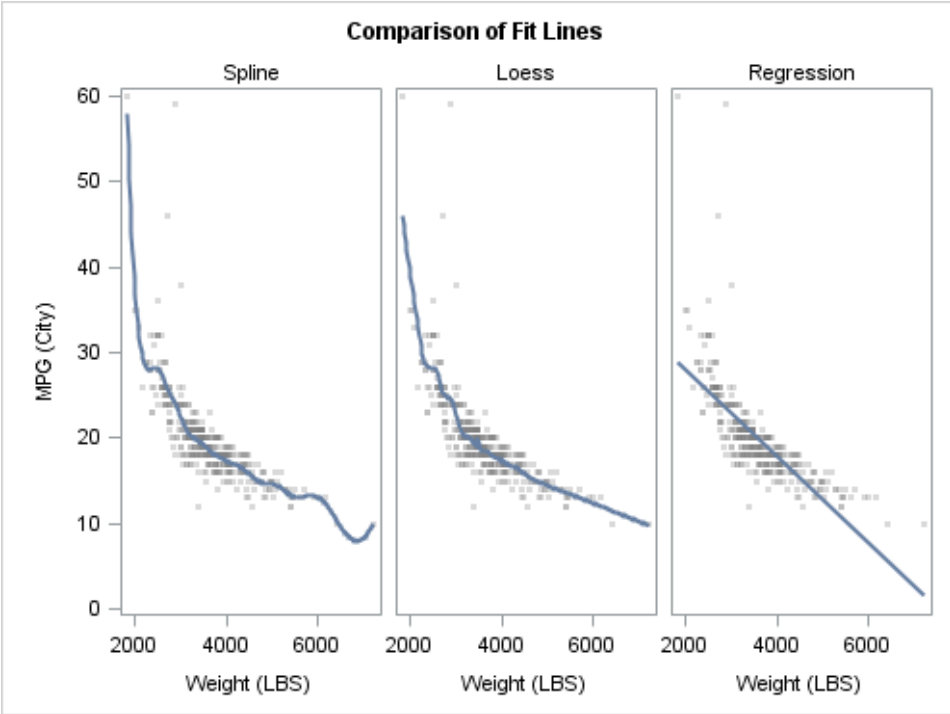
Adjusting the Graph Size

When defining the lattice grid size, you generally have some idea of a good overall aspect ratio for the graph. Consider the following example of a one row by three column lattice.

Example Code 15.1 *Three-Cell Lattice Layout Example*

```
proc template;
  define statgraph graphsize2;
    begingraph;
      entrytitle "Comparison of Fit Lines";
      layout lattice / columns=3 rows=1 rowdatarange=union;
         /* Cell 1 */
         layout overlay;
           entry "Spline" / location=outside valign=top;
           scatterplot x=weight y=mpg_city /
             markerattrs=(size=3px symbol=circlefilled color=gray )
             datatransparency=.7;
           pbsplineplot x=weight y=mpg_city;
         endlayout;
         /* Cell 2 */
         layout overlay;
           entry "Loess" / location=outside valign=top;
           scatterplot x=weight y=mpg_city /
             markerattrs=(size=3px  symbol=circlefilled color=gray)
             datatransparency=.7;
           loessplot x=weight Y=mpg_city;
         endlayout;
         /* Cell 3 */
         layout overlay;
           entry "Regression" / location=outside valign=top;
           scatterplot x=weight y=mpg_city /
             markerattrs=(size=3px symbol=circlefilled color=gray)
             datatransparency=.7;
           regressionplot x=weight y=mpg_city;
         endlayout;
         /* Externalize the row axis */
         rowaxes;
           rowaxis;
         endrowaxes;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.cars template=graphsize2;
run;
```

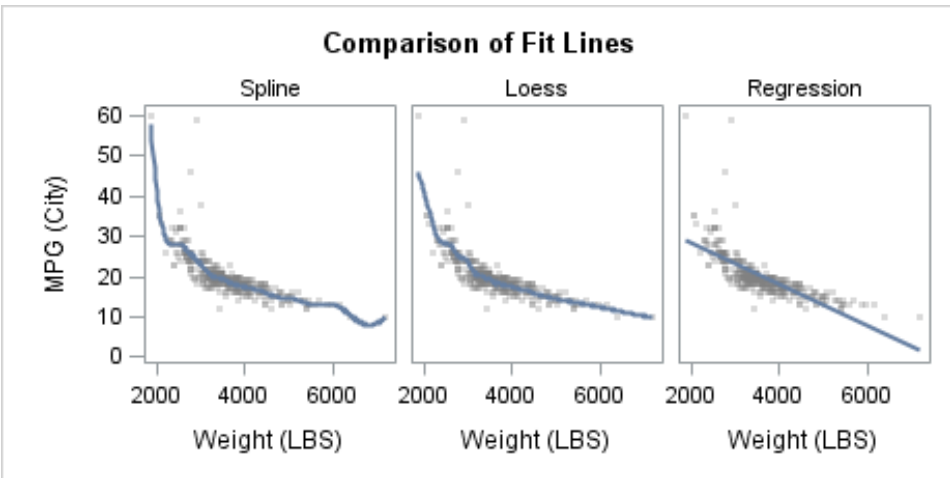
The graph has a default aspect ratio of 4:3 as shown in the following figure.



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=460px designheight=230px;
```

Here is the result.

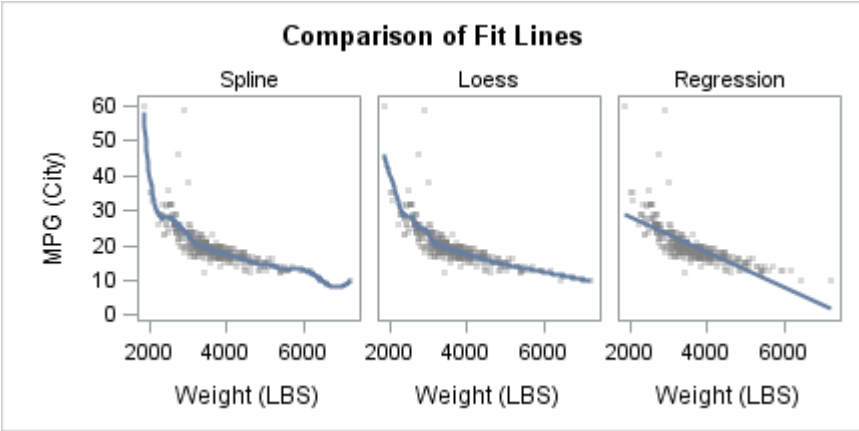


The DESIGNWIDTH= and DESIGNHEIGHT= options set the graph size as part of the template definition so that if you later want a larger or smaller version of this graph, you can use the ODS GRAPHICS statement rather than resetting the design size and recompiling the template. You need only specify either 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=430px;
```

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

Here is the result.



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.

Examples: Lattice Layout

Example 1: Basic Lattice with Internal Axes

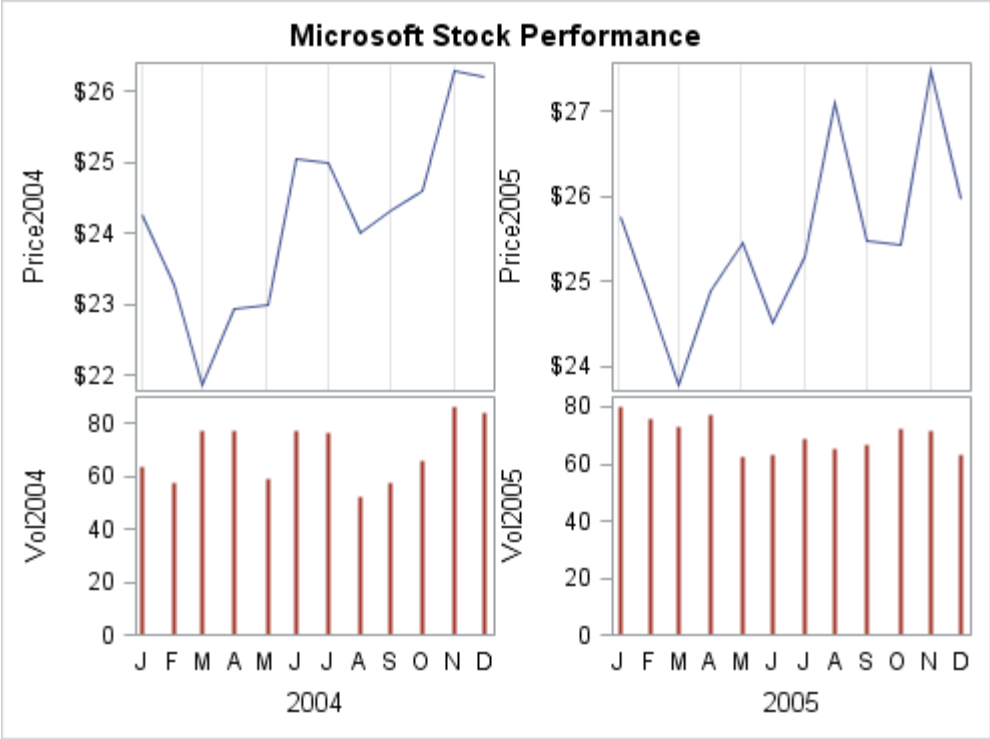
About This Example

This example demonstrates how to generate the basic lattice shown in Figure 15.4 on page 242. To generate the basic lattice, complete the following steps:

1. Transform the input data.
2. Create the basic lattice.
3. Customize the cell axes.

Here is the output.

Figure 15.4 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 frequently have to transform the input data to make it suitable for the graph that you are trying to create. You might need to perform any or all of the following tasks:

- Summarizing the data.
- Transposing the data.
- Scale the data values.
- Create new variables that represent subsets of the data.

The graph that is shown in Figure 15.4 on page 242 is based on data from Sashelp.Stocks, which contains several years of monthly stock information for three companies. The data set contains columns labeled Stock, Date, Volume, and AdjClose (Adjusted Closing Price). However, it does not contain 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 for the Volume and Stock Price within each year. Because the volumes are large, they are scaled to millions. This scaling is 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	.

Every plot in every cell must use variables that contain just the information that is appropriate for that cell. You cannot use WHERE clauses within the template definition to form subsets of the data.

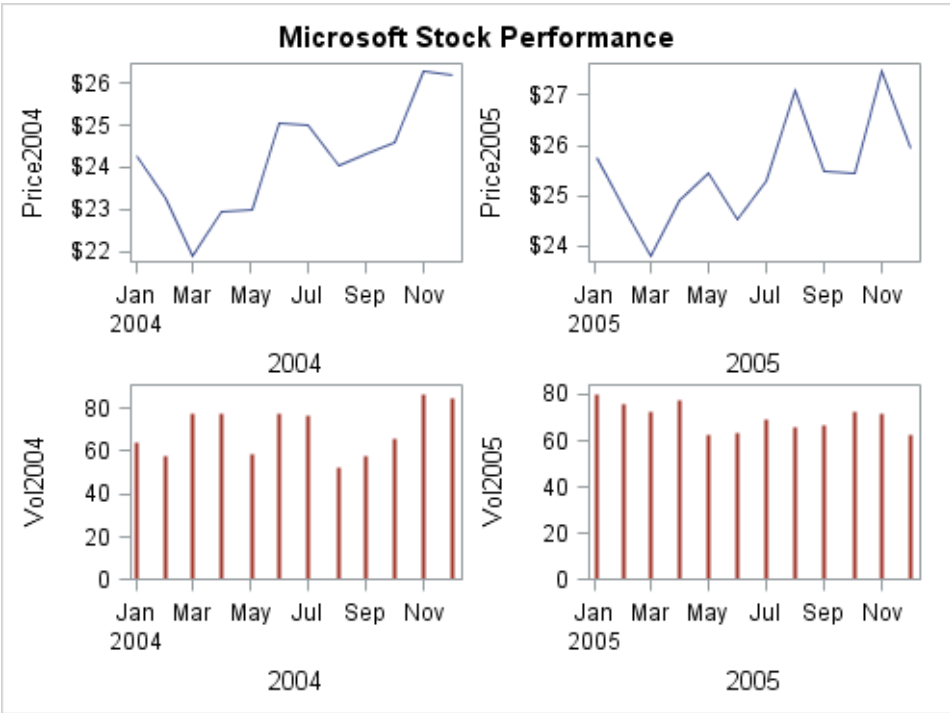
Creating the Basic Lattice

Here is the template code for the basic lattice:

```
proc template;
  define statgraph latticebasic;
    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=latticebasic;
run;
```

Here is the output.

Figure 15.5 *Initial Lattice for the Graph*



Because Date2004 and Date2005 have an associated SAS date format, a TIME axis is used and the variable labels are used for X-axis labels.

Customizing the Cell Axes

In most cases, externalizing axes improves graph appearance and streamlines coding. However, if some axis options do not apply uniformly to all axes in a column or row, you need to use the standard axis options on a per-cell basis instead of using common row and column axes.

For example, suppose you want X-axis grid lines to appear on the top row of plots but not on the second row of plots. You cannot use column axes. Instead, you must 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:

```
proc template;
  define statgraph latticebasic;
    begingraph;
      entrytitle "Microsoft Stock Performance";
      layout lattice / columns=2 rows=2;
        /* define row 1 */
        /* 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;

        /* define row 2 */
        /* 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;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=stock template=latticebasic;
run;
```

Figure 15.4 on page 242 shows the output for the final basic lattice.

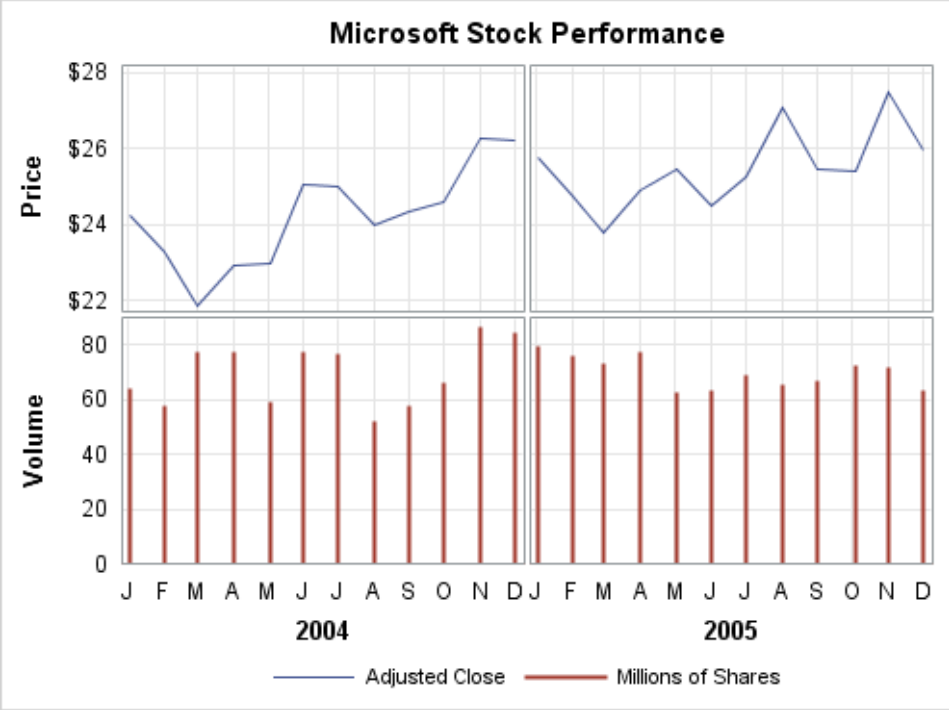
Example 2: Lattice with Side Bars

This example demonstrates how to use side bars to enhance the graph in Figure 15.6 on page 249. In the original graph, the ENTRYTITLE is centered on the entire graph. It would look better if it was centered on the grid area. This can be accomplished by removing the ENTRYTITLE statement and adding a side bar along the top that contains the title text in an ENTRY statement. Note that the prices in the first row represent an adjusted close value and that the axis scaling for the second row is in millions of shares. Two strategies are available for providing this information. One strategy is to create an external legend. For this strategy, you define legend text on two of the plot statements, and then add a DISCRETELEGEND statement to a side bar along the bottom. Here is the modified template code.

```
proc template;
  define statgraph latticesidebar;
    begingraph;
      layout lattice / columns=2 rows=2
        rowdatarange=union columndatarange=union
        rowgutter=3px columngutter=3px;
        /* define row 1 */
        seriesplot x=date2004  y=price2004 / lineattrs=GraphData1
          name="series" legendlabel="Adjusted Close";
        seriesplot x=date2005  y=price2005 / lineattrs=GraphData1;
        /* define row 2 */
        needleplot x=date2004  y=vol2004 /
          lineattrs=GraphData2(thickness=2px pattern=solid)
          name="needle" legendlabel="Millions of Shares";
        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;
        /* Add top and bottom sidebars */
        sidebar / align=top;
          entry "Microsoft Stock Performance" /
                textattrs=GraphTitleText pad=(bottom=5px);
        endsidebar;
        sidebar / align=bottom;
          discretelegend "series" "needle" / border=off;
        endsidebar;
      endlayout;
```

```
      endgraph;
    end;
  run;

  proc sgrender data=stock template=latticesidebar;
  run;
```

The following graph shows the modifications.



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 as shown in the following ROWAXES block.

```
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 you might want more text, not alternate text, and there is no way to wrap the axis label to two lines. The solution is to use row headers instead of specifying axis labels.

Example 3: Lattice with Row and Column Axes

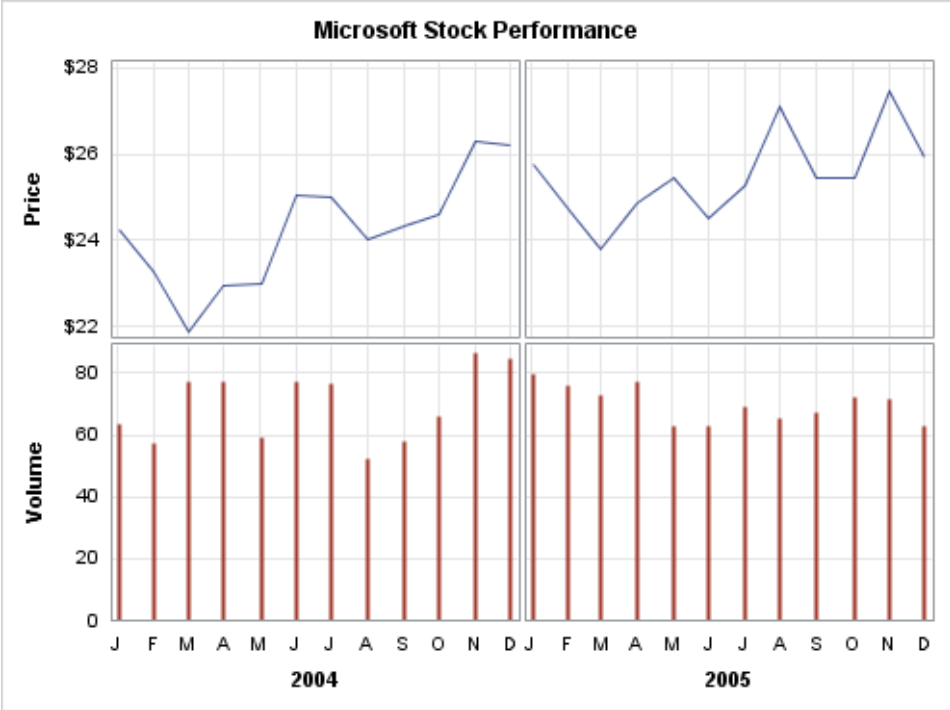
Figure 15.5 on page 244 would benefit from externalizing the X and Y axes because row and column axes reduce the redundant X-axis information and unify the data ranges in the Y axes. You might also want to add grid lines to all axes. To conserve

space along the X axes, the automatic formatting of each TIME axis is turned off in the following template code. The TICKVALUEFORMAT=MONNAME1. option demonstrates how to format the time axis tick values.

```
proc template;
  define statgraph latticeexternalaxes;
    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=latticeexternalaxes;
run;
```

Here is the output.

Figure 15.6 *Lattice with Row and Column Axes*

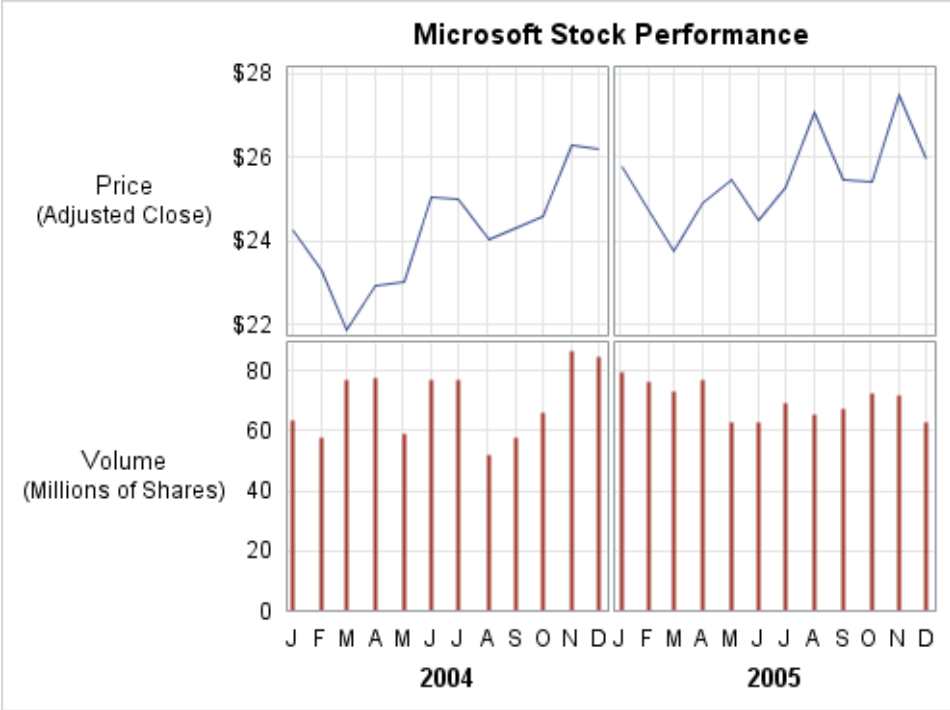


Example 4: Lattice with Row Headers

In "Example 2: Lattice with Side Bars" on page 246, a legend in a sidebar was used to indicate that the first row values are adjusted close values and that the second row values are millions of shares. Another way to display that information is to remove the label information from the row axes and introduce a ROWHEADERS block, as shown in the following code fragment.

```
rowaxes;
  rowaxis / griddisplay=on display=(tickvalues);
  rowaxis / griddisplay=on display=(tickvalues);
endrowaxes;

rowheaders;
  layout gridded / columns=1;
    entry "Price" / textattrs=GraphLabelText;
    entry "(Adjusted Close)" / textattrs=GraphValueText;
  endlayout;
  layout gridded / columns=1;
    entry "Volume" / textattrs=GraphLabelText;
    entry "(Millions of Shares)" / textattrs=GraphValueText;
  endlayout;
endrowheaders;
```

By nesting the ENTRY statements in the LAYOUT GRIDDED statement block, you can have multiple lines of text split exactly where you want and in any text style that you desire. Without the GRIDDED layouts, only one ENTRY statement could be used per row. Here is the result.

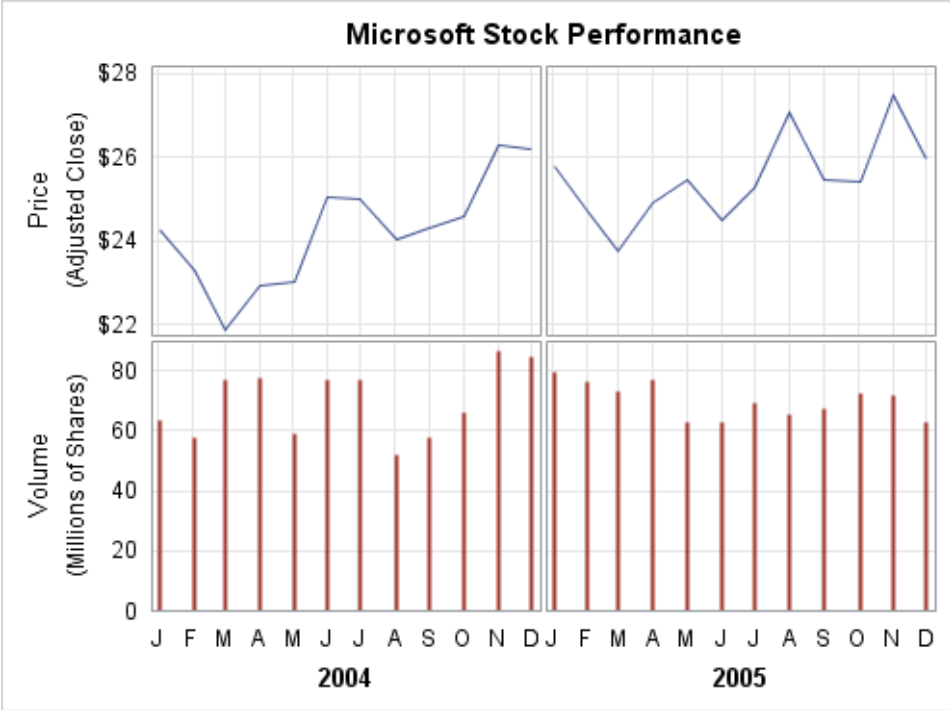


To allow more space for the plots, you can rotate the row header text to make it appear to be a row axis label as shown in the following modified ROWHEADERS block:

```
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;
```

Notice that you must specify COLUMNS=2 in the LAYOUT GRIDDED statements. Here is the final graph.

Figure 15.7 *Final Lattice*

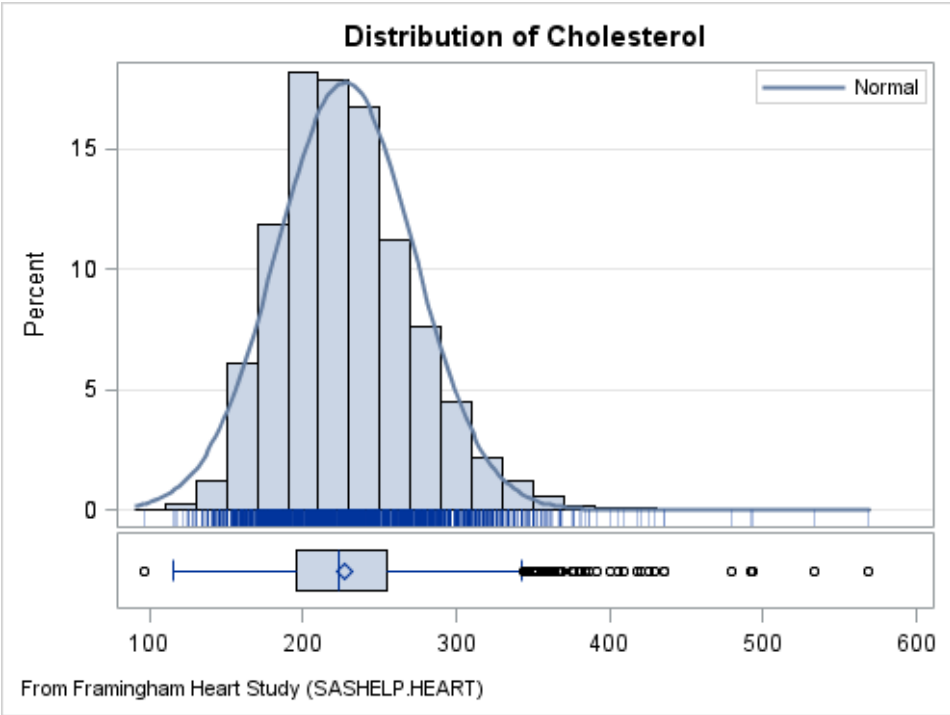


The clean look of the graph is achieved by removing redundant cell axis information and moving it to external column and row locations. In this example, the use of row headers provided the desired flexibility over row axis labels.

Example 5: Lattice with Custom Row Sizing

This example shows you how to use the ROWWEIGHTS= and COLUMNWEIGHTS= options to size the rows. Figure 15.8 on page 252 shows a two-row by one-column lattice that contains two-dimensional plots and a one-dimensional plot. 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 one-dimensional box plot. The X axes have a uniform scale to ensure that the box plot aligns correctly with the histogram. By default, each row is apportioned an equal share of the total height. However, 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= is used to reapportion the row space. This example uses the ROWHEIGHTS=PREFERRED option to set the rows to an appropriate height automatically, as shown in the following figure.

Figure 15.8 Graph with ROWEIGHTS=PREFERRED



Here is the SAS code for this graph.

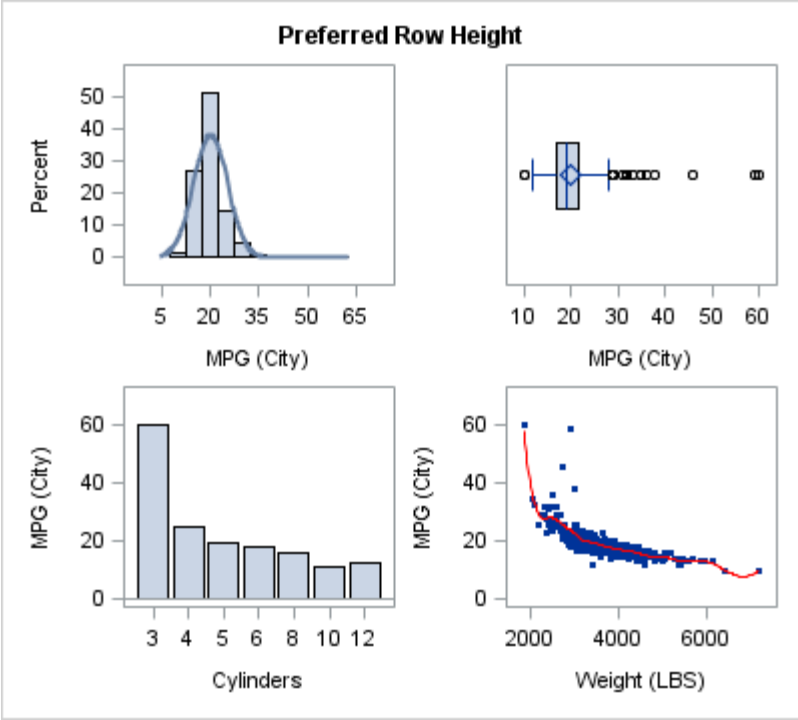
```
proc template;
  define statgraph distribution;
    begingraph;
      entrytitle "Distribution of Cholesterol";
      entryfootnote halign=left
        "From Framingham Heart Study (SASHELP.HEART)";
      layout lattice / rowweights=PREFERRED
        columndatarange=union rowgutter=2px;
        columnaxes;
          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;
```

Note: For a generic version of this template that you can use to show the distribution for any continuous variable without redefining the template, see Chapter

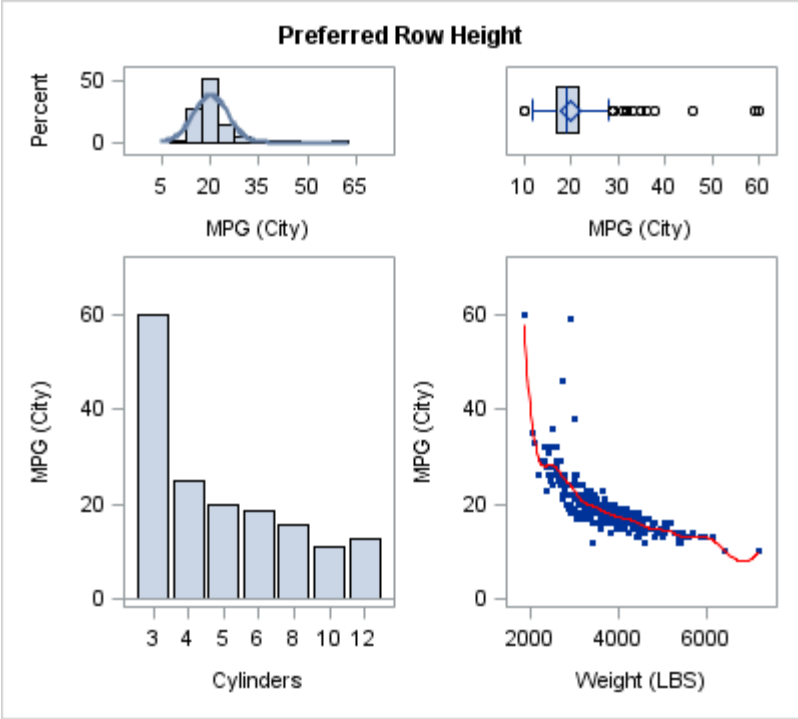
30, “Using Dynamic Variables and Macro Variables in Your Templates,” on page 605.

The PREFERRED option is typically used to size rows and columns automatically when a lattice contains a mix of two-dimensional and one-dimensional plots. In this example, the system automatically sets the height of the second row to what is necessary to properly display the box plot. It then sets the height of the first row to the balance of the available height. You can also use numeric weights to manually set the row heights. Using the ROWHEIGHTS=(0.9 0.1) option instead of the ROWHEIGHTS=PREFERRED option in the previous example produces a similar result. It apportions 90% of the available height to the first row and 10% to the second row.

If a row contains a mix of one-dimensional and two-dimensional plots, the PREFERRED option sets the row height to the maximum preferred height of the one-dimensional plots in the row. For columns, it does the same for the column width. This might produce undesirable results. For example, the following figure shows a simple two-column by two-row lattice that uses the default row and column sizing. The first column contains two-dimensional plots. The second column contains a one-dimensional plot in the first row and two-dimensional plots in the second row.



By default, the row height is set to the maximum available height as shown, which easily accommodates all of the plots. If the row weight is set to PREFERRED, the row height for the first row is set to the preferred height of the box plot in the second column, which compresses the plots in the first column, first row, as shown in the following figure.



Use the PREFERRED option for rows and columns that contain only one-dimensional plots. For a row or column that contains a mix of one-dimensional and two-dimensional plots, use UNIFORM or a list of numeric weights instead.

For more information about the ROWWEIGHTS=PREFERRED and COLUMNWEIGHTS=PREFERRED options, see "LAYOUT LATTICE" in *SAS Graph Template Language: Reference*.

16

Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts

About Classification Panels

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:

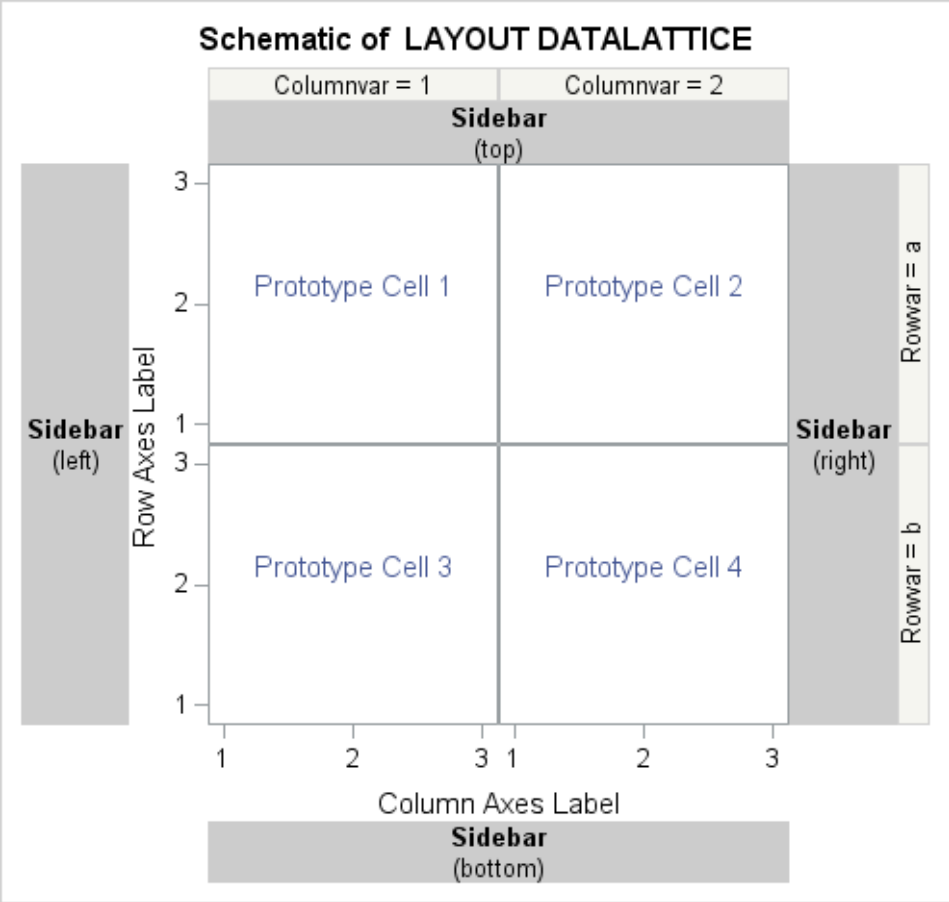
LAYOUT DATAPANEL
: supports a list of class variables. The number of rows and columns are controlled by statement options. Each cell is labeled with the class variable values in the cell header.

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 DATALATTICE Statement

Figure 16.1 on page 257 shows the general organization of a graph that is produced with a DATALATTICE layout. If the template code does not use the sidebar areas that are shown in the schematic, that space is reclaimed in the graph. Notice that the sidebar area is between the cells and the row and column headers.

Figure 16.1 Schematic of a DATALATTICE Layout

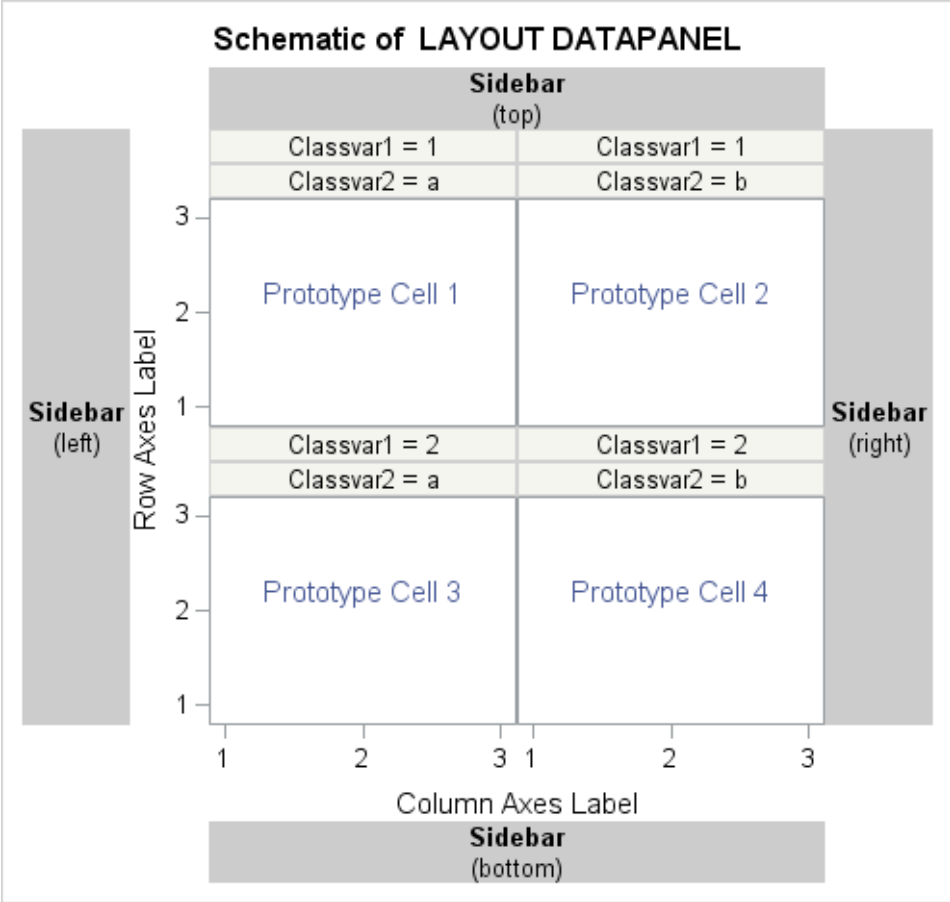


Use a LAYOUT DATALATTICE statement to open the layout block for your DATALATTICE layout. In the LAYOUT DATALATTICE statement, specify the column variable in the COLUMNVAR= option, and the row variable in the ROWVAR= option. Several other options are available in the LAYOUT DATAPANEL statement that you can use to control various aspects of the panel. Use a PROTOTYPE block to define the contents for each cell. To add one or more side bars, use a SIDEBAR block to define each side bar. For information about the LAYOUT DATALATTICE statement, see "LAYOUT DATALATTICE" in *SAS Graph Template Language: Reference*.

The LAYOUT DATAPANEL Statement

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)`.

Figure 16.2 Schematic of a DATAPANEL Layout



Use a LAYOUT DATAPANEL statement to open the layout block for your DATAPANEL layout. In the LAYOUT DATAPANEL statement, specify the classification variables in the CLASSVARS= option. Several other options are available in the LAYOUT DATAPANEL statement that you can use to control various aspects of the panel. Use a PROTOTYPE block to define the contents for each cell. To add one or more side bars, use a SIDEBAR block to define each side bar. For information about the LAYOUT DATAPANEL statement, see "LAYOUT DATAPANEL" in *SAS Graph Template Language: Reference*.

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 290 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 about adding a legend or other information outside of the cells, see "Example 3: A Data Panel with Sidebars" on page 298.
- Axis options for classification panels are specified on the LAYOUT DATALATTICE or LAYOUT DATAPANEL statement, not on the LAYOUT PROTOTYPE statement. For information about setting axis options for the layout, see "Managing Axes in DATALATTICE and DATAPANEL Layouts" on page 268.

Distinction between DATAPANEL and DATALATTICE

The DATAPANEL and DATALATTICE layouts differ in how their classification variables are declared and in how they populate their cells. For the DATAPANEL layout, the classification variables are declared as a list of variables in parentheses in the CLASSVAR= option as shown in the following example.

```
layout datapanel classvars=(product division) / ...;
```

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.

For the DATALATTICE layout, one classification variable is assigned for a row dimension, and one classification variable is assigned for a column dimension. The row variable is assigned in the ROWVAR= option, and the column variable is assigned in the COLUMNVAR= option as shown in the following example.

```
layout datalattice rowvar=product columnvar=division / ...;
```

Another key difference between the DATAPANEL and the DATALATTICE layouts is how they create their cells. The DATAPANEL layout creates a cell for each classification-variable crossing that produces data. Any crossing that does not produce data is not included in the panel. Conversely, the DATALATTICE layout creates a cell for each crossing of its classification variables regardless of whether it produces data. A crossing that produces no data appears as an empty cell in the lattice. This difference is significant when analyzing data that is sparse. In that case, the DATAPANEL layout produces a panel that contains only cells that display data while the DATALATTICE layout produces a lattice that might contain a large number of empty cells.

Organizing Panel Contents

Overview of What to Consider When Planning a Classification Panel

When planning a classification panel, the following factors 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;
```

Here is the resulting grid layout.

var = value 1	var = value 2	var = value 3
Plot 1	**Plot 2**	**Plot 3**
var = value 4	var = value 5	
Plot 4	**Plot 5**	

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;
```

Here is the resulting grid layout.

var = value 1	var = value 3	var = value 5
Plot 1	**Plot 3**	**Plot 5**
var = value 2	var = value 4	
Plot 2	**Plot 4**	

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;
```

Here is the resulting grid layout.

var = value 4	var = value 5	
Plot 4	**Plot 5**	
var = value 1	var = value 2	var = value 3
Plot 1	**Plot 2**	**Plot 3**

```
layout datapanel classvars=(var) / columns=3 rows=2
    order=columnmajor start=bottomleft;
  layout prototype;
    ... plot statements ...
  endlayout;
endlayout;
```

Here is the resulting grid layout.

var = value 2	var = value 4	
Plot 2	**Plot 4**	
var = value 1	var = value 3	var = value 5
Plot 1	**Plot 3**	**Plot 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 that 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 277.

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;
```

Here is the resulting grid layout.

A = a1	A = a1	A = a1
B = b1	B = b2	B = b3
Plot 1	**Plot 2**	**Plot 3**
A = a2	A = a2	A = a2
B = b1	B = b2	B = b3
Plot 4	**Plot 5**	**Plot 6**

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;
```

Here is the resulting grid layout.

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

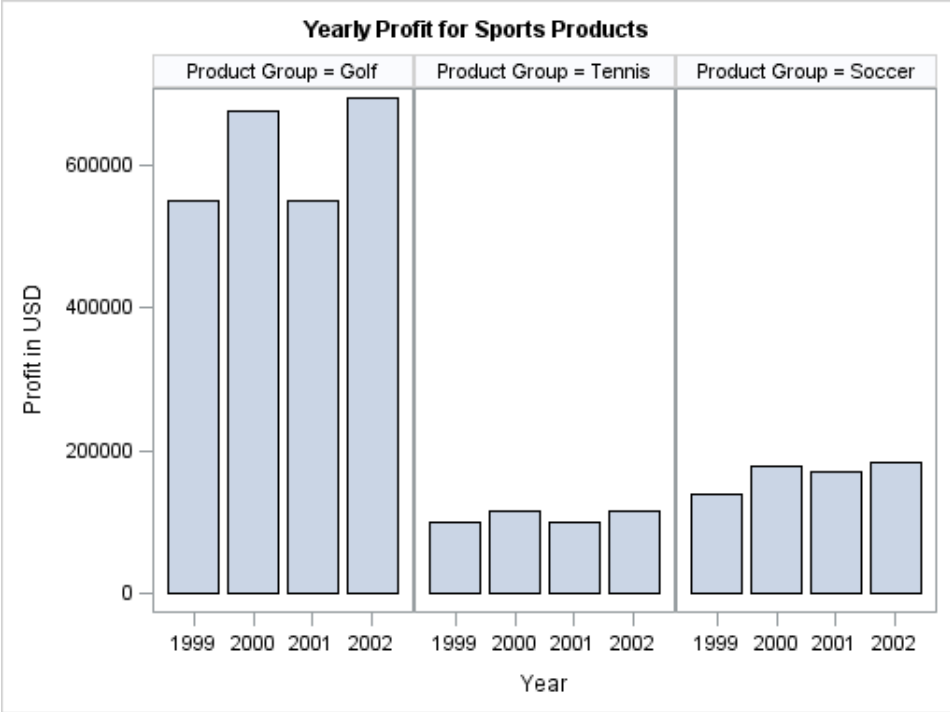
The default graph size is 640 pixels in width and 480 pixels in height, which sets a default aspect ratio of 4:3 (640:480). Depending on your grid size, you might want to adjust the aspect ratio to improve the appearance of the panel. For example, Example Code 16.1 on page 263 defines a three column by one row grid.

Example Code 16.1 *3x1 Panel Template*

```
proc template;
 define statgraph onerow;
  begingraph;
    entrytitle "Yearly Profit for Sports Products";
    layout datapanel   classvars=(product_group) / rows=1;
      layout prototype;
        barchart category=year response=profit / stat=sum;
      endlayout;
    endlayout;
  endgraph;
 end;
run;
```

Here is the output when this template is rendered with the default aspect ratio.

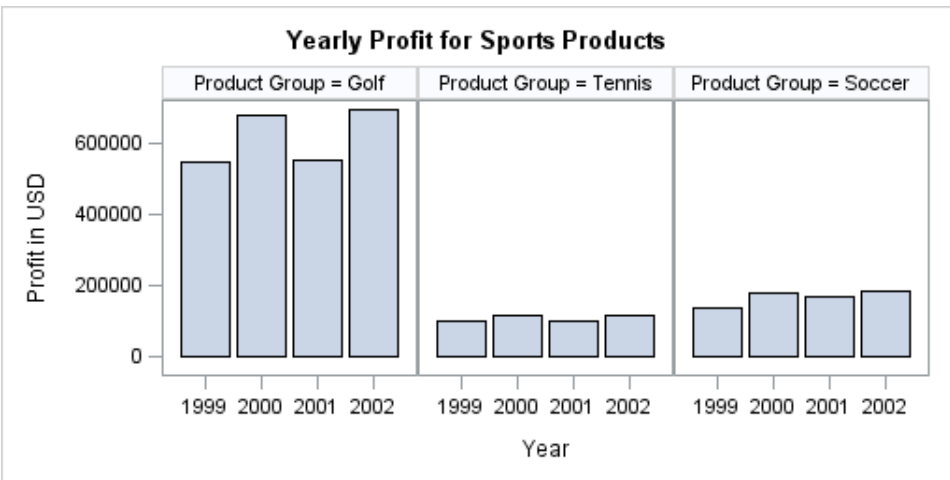
```
proc sgrender data=sashelp.orsales template=onerow;
  where product_group in ("Golf" "Tennis" "Soccer");
run;
```



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. For example, to render the panel in Example Code 16.1 on page 263 with a 2:1 aspect ratio, modify the BEGINGRAPH statement as follows:

```
begingraph / designwidth=640px designheight=320px;
   . . .
endgraph;
```

Here is the result.



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, to render the panel in Example Code 16.1 on page 263 as 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;
```

To render it as 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

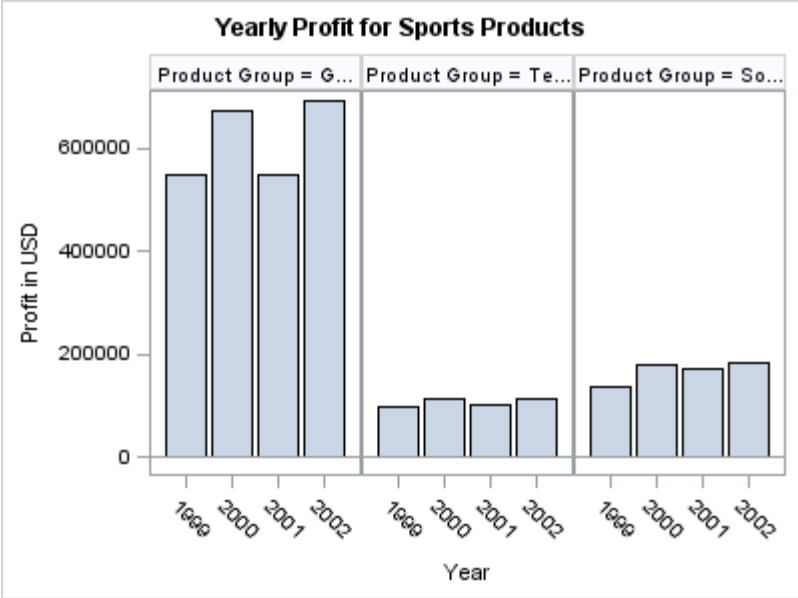
You might think that the panel size can be varied to be as big or small as desired. However, problems arise as the graph size shrinks. The following adjustments in the graph enable small images to be produced:

- The font sizes are reduced.
- The axis tick values are thinned, rotated, or truncated.
- The 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 272.)

For example, to render the panel in Example Code 16.1 on page 263 as a 400-pixel-wide graph:

```
ods graphics / reset width=400px;
proc sgrender data=sashelp.orsales template=onerow;
  where product_group in ("Golf" "Tennis" "Soccer");
run;
```

Here is the result.



This panel is approaching the limits of how small it can be. Reducing the size even more eventually produces log messages similar to the following:

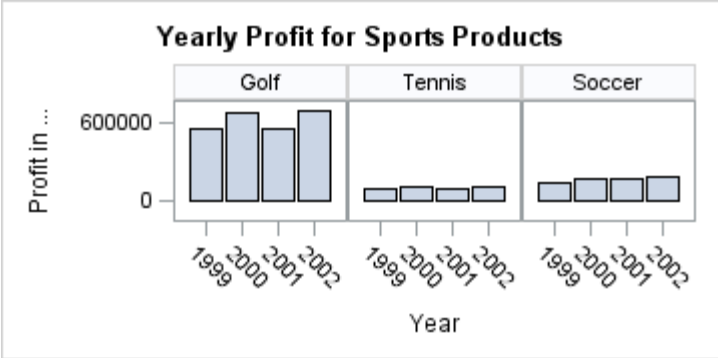
```
Cell width 72 is smaller than the minimum cell width 100. All contents are
removed from the layout.
```

Although an image is produced, it is empty. 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 the panel design included additional titles, 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 cell sizes of less than 100 pixels as shown in the following example.

```
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 category=year response=profit / stat=sum;
        endlayout;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.orsales template=onerow;
  where product_group in ("Golf" "Tennis" "Soccer");
run;
```

Here is the output.



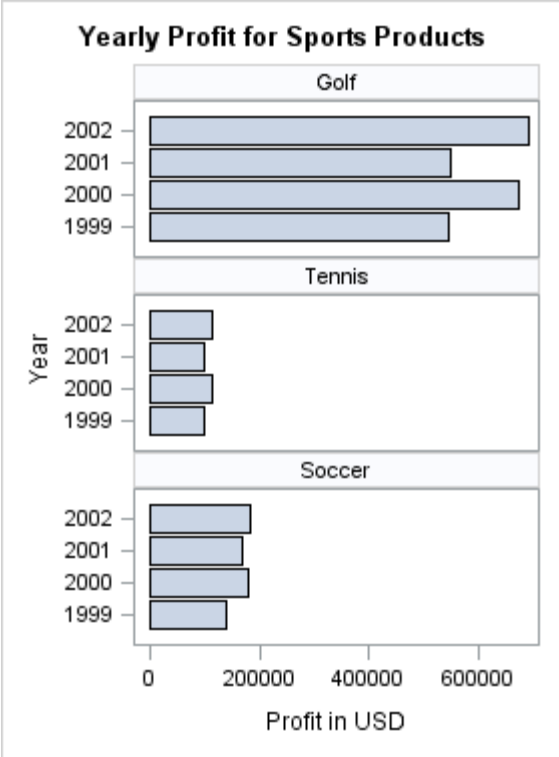
For graph templates that are intended for repeated use (such as the ones that are part of other SAS products), the effort has been made to set the CELLWIDTHMIN= and CELLHEIGHTMIN= option to the smallest values that produce a reasonable panel. Other strategies produce smaller cells without truncating 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 example 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 category=year response=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;
```

Here is the output.



Managing Axes in DATALATTICE and DATAPANEL Layouts

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. Consider the following template:

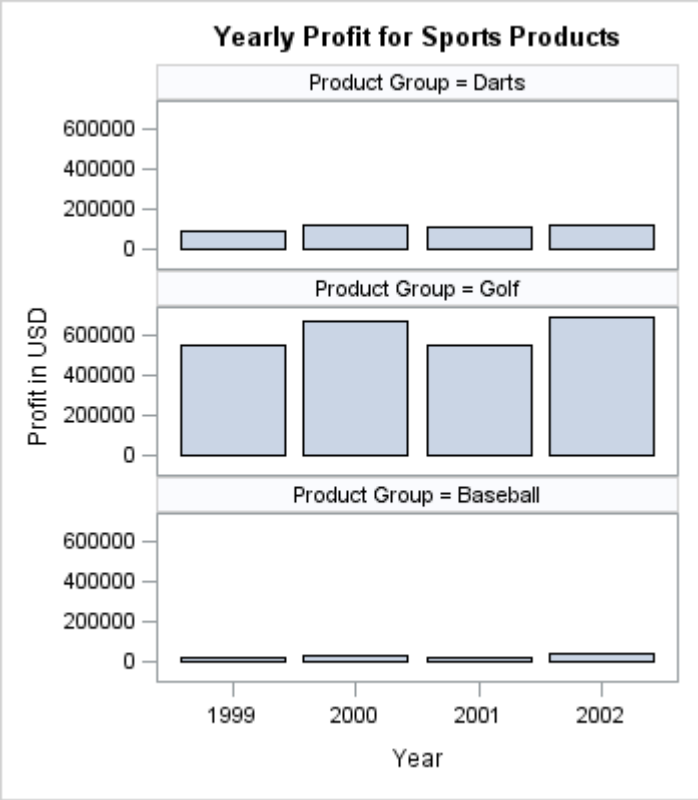
Example Code 16.2 *Classification Panel Template*

```
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 category=year response=profit /
          stat=sum;
      endlayout;
```

```
      endlayout;
    endgraph;
  end;
run;
```

This template generates a classification panel that compares the profits for Darts, Golf, and Baseball. Here is the output.

```
proc sgrender data=sashelp.orsales template=unionall;
  where product_group in
    ("Golf" "Darts" "Baseball");
run;
```



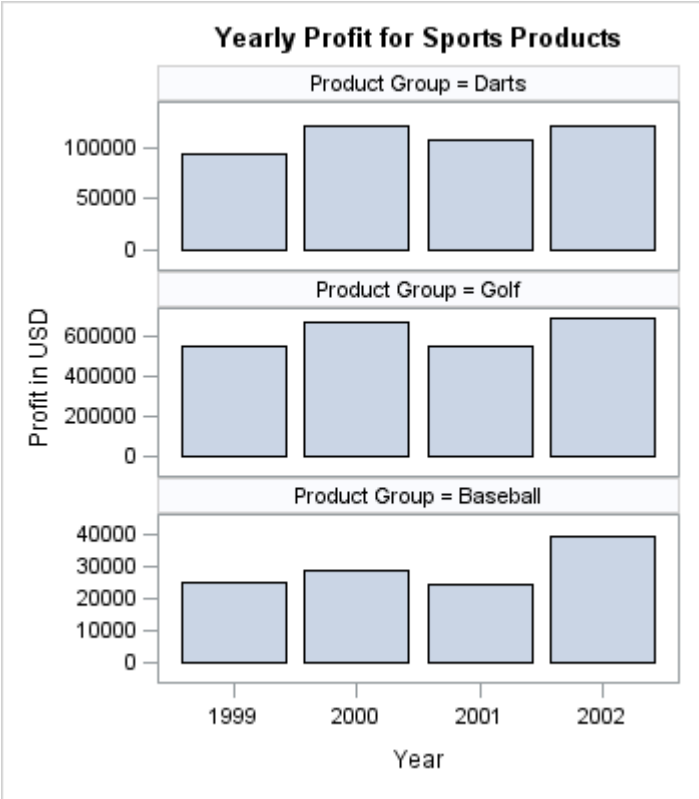
By default, the minimum and maximum data ranges over all rows in all panels are used to establish identical data ranges across for axes that appear in the rows. The same is true for columns. The options that set these defaults are ROWDATARANGE=UNIONALL and COLUMNDATARANGE=UNIONALL. In most cases, these 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 LAYOUT DATAPANEL statement 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;
```

Here is example output.



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

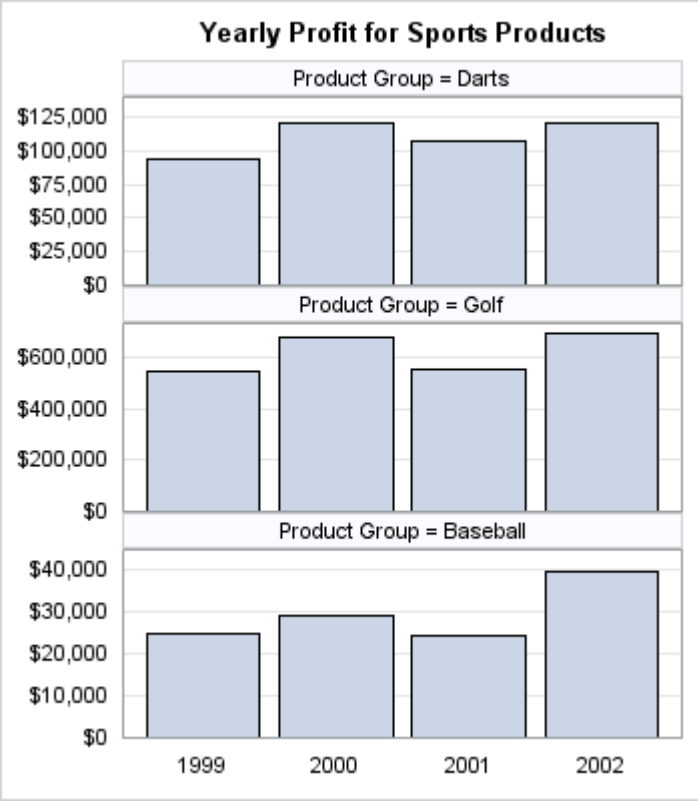
Classification panels use the ROWAXISOPTS=(*axis-opts*) and COLUMNAXISOPTS= (*axis-opts*) options to set axis features. Options are available for all four axis types (LINEAR, DISCRETE, LOG, and TIME), and most of the available axis options are a slightly restricted set of the axis options that are available 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. The following LAYOUT DATAPANEL statement 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);
```

Here is example output.



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. In 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);
```

Here is example output.

Controlling the Classification Headers

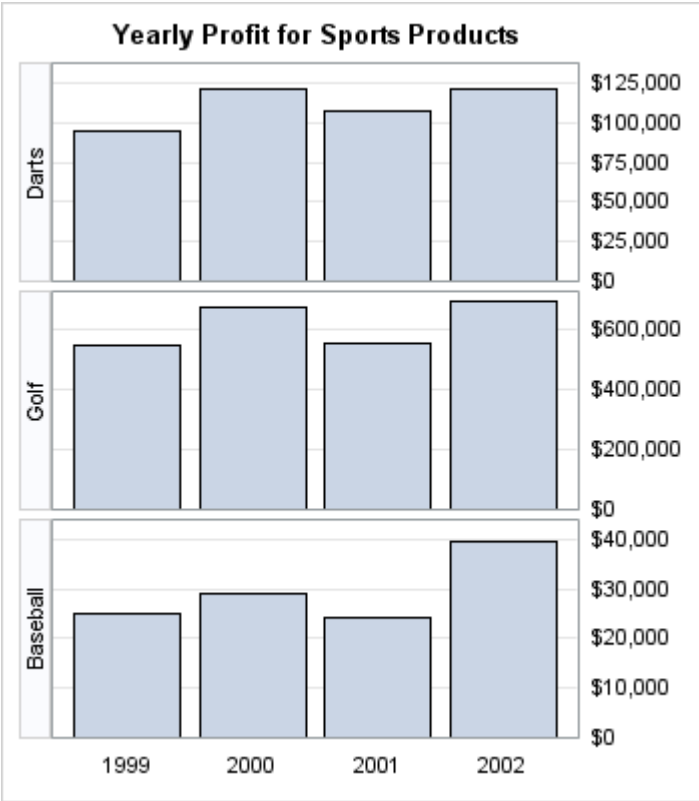
In many cases, it is not necessary to display the classification-variable name in the classification headers. Often, just the classification value is sufficient. Both the DATALATTICE and DATAPANEL layouts support the HEADERLABELDISPLAY= option. By default, HEADERLABELDISPLAY=NAMEVALUE, which displays both the variable name and the value. You can set HEADERLABELDISPLAY=VALUE to display only the value, or you can set HEADERLABELDISPLAY=NONE to suppress the headers.

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. You can change the location of the row or column axis information by using the DISPLAYSECONDARY= axis option. In the following example, the row headers are relocated to the left, and the axis information is relocated to the right. Note that DISPLAY=NONE is also included in order to remove the default row axis information from the left side.

```
proc template;
  define statgraph unionall;
    begingraph / designwidth=350px designheight=400px;
      entrytitle "Yearly Profit for Sports Products";
      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 );
        layout prototype;
          barchart category=year response=profit / stat=sum;
        endlayout;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.orsales template=unionall;
  where product_group in
    ("Golf" "Darts" "Baseball");
run;
```

Here is example output.



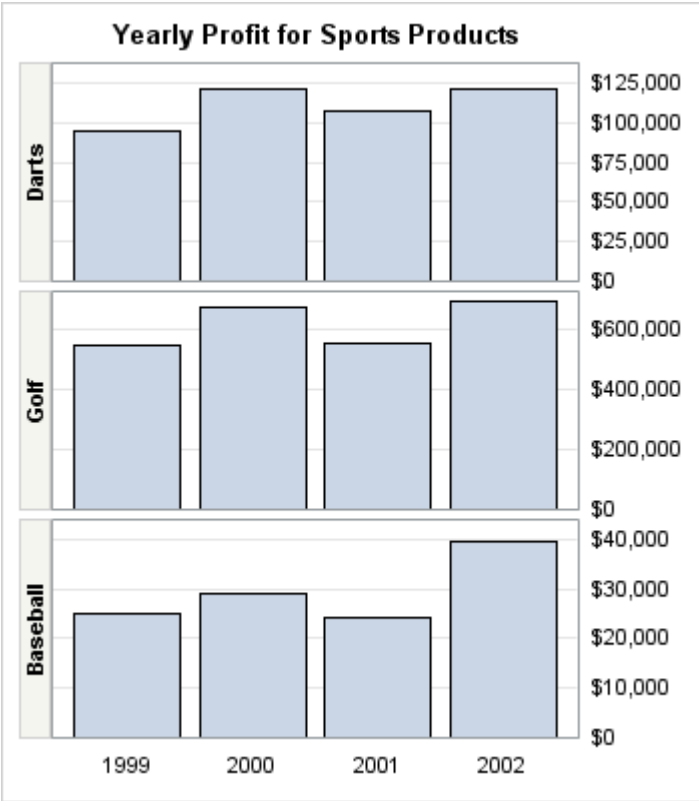
Both the DATAPANEL and DATALATTICE layouts support options that control the background and text properties of the classification headers. By default, the background of the cell headers is transparent (HEADEROPAQUE=FALSE).

Use the HEADERBACKGROUNDCOLOR= option to set the background fill color. In the following example, the background color specification is a style reference. You must also set HEADEROPAQUE=TRUE. 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 LAYOUT DATALATTICE statement, 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);
```

Here is example output.

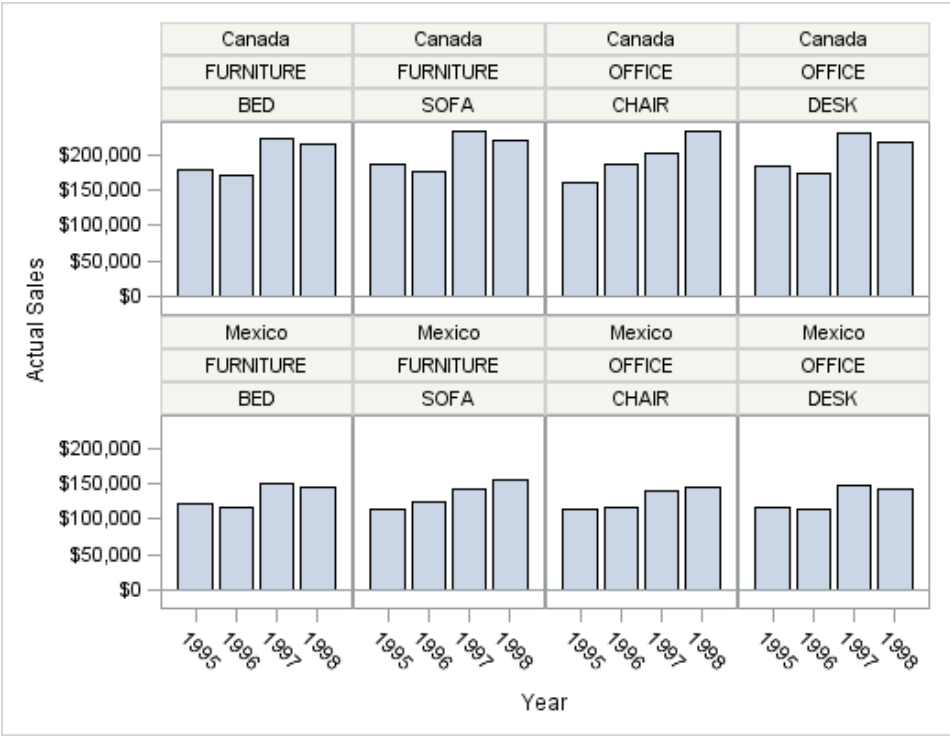


When HEADERLABELLOCATION=INSIDE and multiple classification variables are displayed in the cell headers in a DATAPANEL or DATALATTICE layout, each classification value occupies a separate cell in the header. Here is an example of a DATAPANEL layout with three classification variables.

```
proc template;
  define statgraph layoutdatapanel;
    begingraph / drawspace=layoutpercent;
      layout datapanel classvars=(country prodtype product) /
        columns=4 rowdatarange=unionall
        headerlabeldisplay=value
        headerbackgroundcolor=GraphAltBlock:color;
        layout prototype;
          barchart x=year y=TotalActual;
```

```
        endlayout;
      endlayout;
    endgraph;
  end;
run;

/* Summarize the data in SASHELP.PRDSAL2 */
proc summary data=sashelp.prdsal2 nway;
  class country year product prodtype;
  var actual predict;
  output out=prdsal2 sum=TotalActual TotalPredict;
run;

/* Generate the panel using the summarized data */
proc sgrender data=prdsal2 template=layoutdatapanel;
  where country in ("Canada" "Mexico");
run;
```

Here is the output.

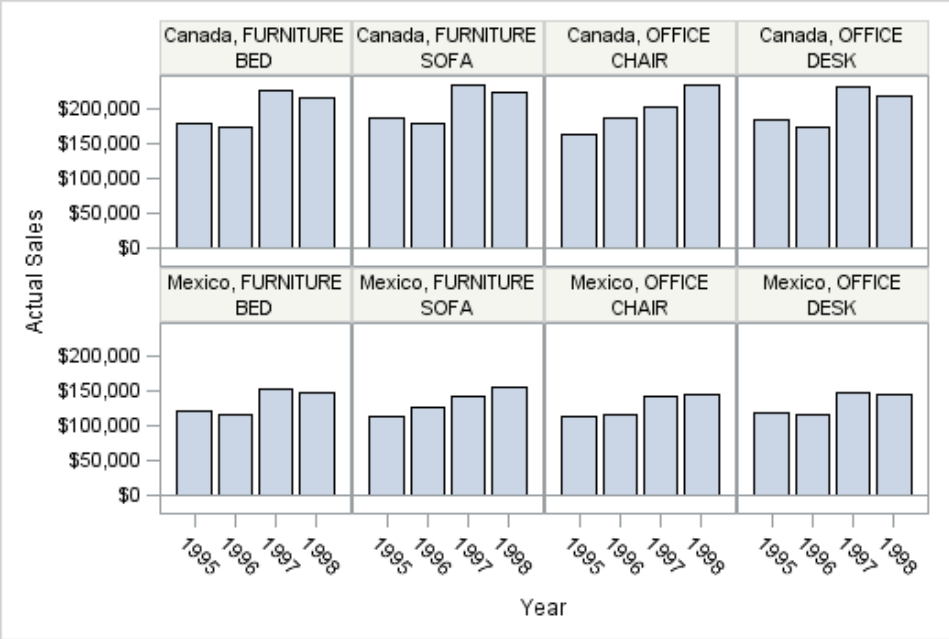


Starting with SAS 9.4M1, you can use the HEADERPACK=TRUE option to consolidate the header values into a comma-separated list in order to save space. If the width of the consolidated header exceeds the available width, the header is truncated. In that case, use the HEADERSPLITCOUNT= option to specify the number of values that are to be consolidated before the list is split into a separate line. Here is the template from the previous example, modified to consolidate the values in the column headers into two lines.

```
proc template;
  define statgraph layoutdatapanel;
    begingraph / drawspace=layoutpercent;
      layout datapanel classvars=(country prodtype product) /
        columns=4 rowdatarange=unionall
```

```
          headerlabeldisplay=value
          headerpack=true headersplitcount=2
          headerbackgroundcolor=GraphAltBlock:color;
          layout prototype;
            barchart x=year y=TotalActual;
          endlayout;
        endlayout;
      endgraph;
    end;
  run;

  proc sgrender data=prdsal2 template=layoutdatapanel;
    where country in ("Canada" "Mexico");
  run;
```

Here is the output.



By default, the items in the list are separated by a comma and a space. Use the HEADERSEPARATOR= option to specify a different separator. For more information about the HEADERSPLIT=, HEADERSPLITCOUNT=, and HEADERSEPARATOR= options, see "LAYOUT DATALATTICE" in *SAS Graph Template Language: Reference* or "LAYOUT DATAPANEL" in *SAS Graph Template Language: Reference*.

Using Sidebars

A side bar is a reserved area along the left, right, top, or bottom of a DATAPANEL or DATALATTICE layout in which you can put additional information. A side bar is useful for adding information that applies to that row or column such as a legend or explanatory text. You can add up to four side bars to a DATAPANEL or DATALATTICE layout as shown in Figure 16.1 on page 257 and Figure 16.2 on

page 258. As shown, a left or right side bar spans all of the rows while a top or bottom side bar spans all of the columns. You define the contents of a side bar in SIDEBAR block in the LAYOUT DATAPANEL or LAYOUT DATALATTICE statement. The position of the sidebar is controlled by the ALIGN= option in the SIDEBAR statement. For information about using side bars with the DATAPANEL layout, see "Sidebar Blocks" in *SAS Graph Template Language: Reference*. For information about using side bars with the DATALATTICE layout, see "SIDEBAR Blocks" in *SAS Graph Template Language: Reference*.

For an example, see "Example 3: A Data Panel with Sidebars" on page 298.

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

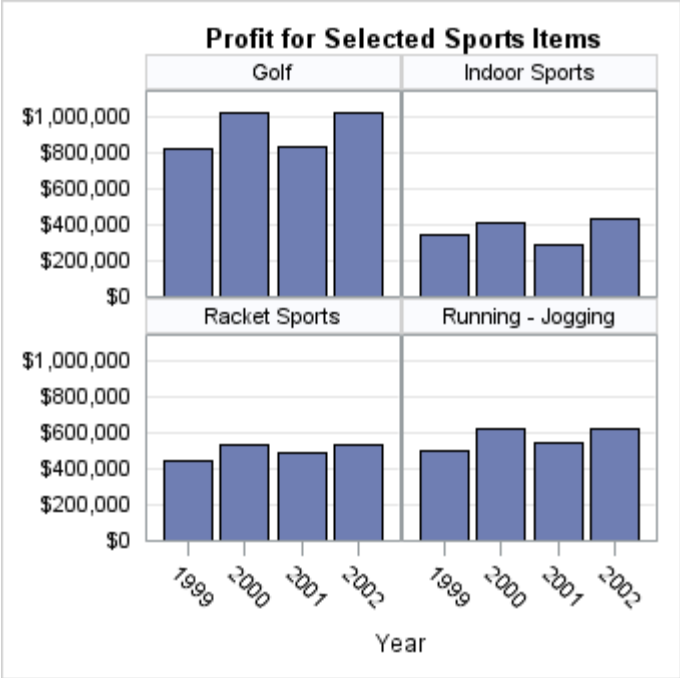
Appearance of the Last Panel

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 as shown in the following example.

```
proc template;
  define statgraph multipanel;
    begingraph / designwidth=340px designheight=340px;
      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;
    endgraph;
end;
run;

proc sgrender data=sashelp.orsales template=multipanel;
```

```
    where product_line in ("Sports") and
      product_category ne "Assorted Sports Articles";
  run;
```

Here are the panels that are generated.



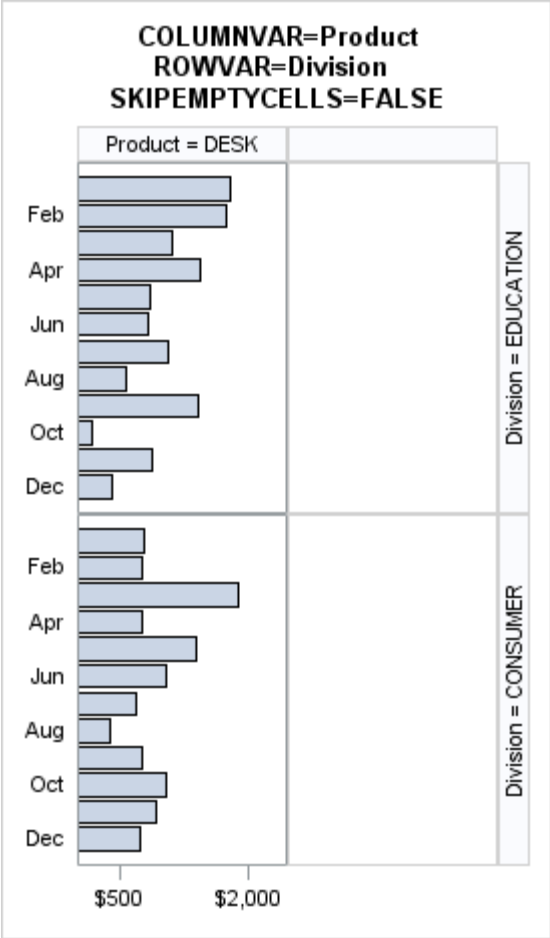
To eliminate empty cells on the last panel, you can specify SKIPEMPTYCELLS=TRUE as shown in the following LAYOUT DATAPANEL statement.

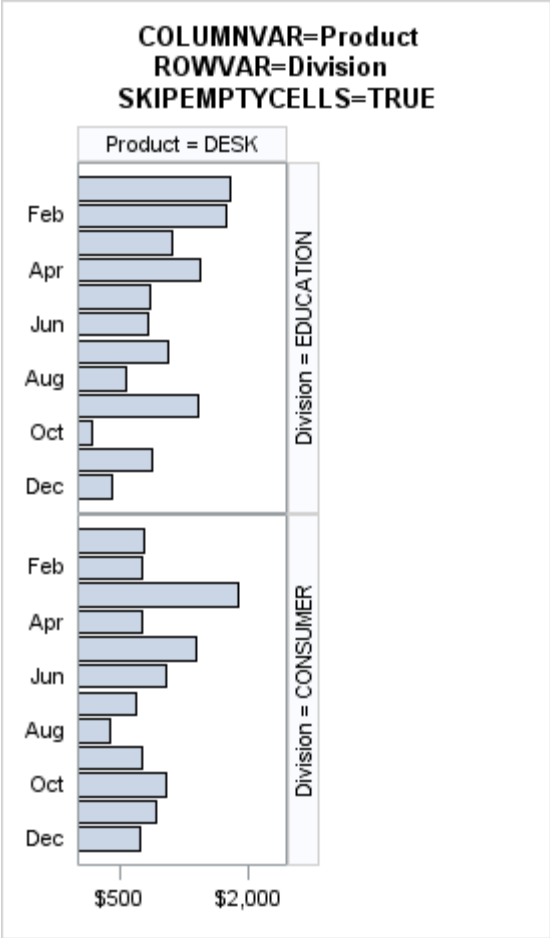
```
layout datapanel classvars=(product_category) /
    rows=2 columns=2
    skipemptycells=true
```

```
headerlabeldisplay=value
rowaxisopts=(griddisplay=on offsetmin=0
display=(tickvalues) linearopts=(tickvalueformat=dollar12.));
```

Here is the result on the last panel.

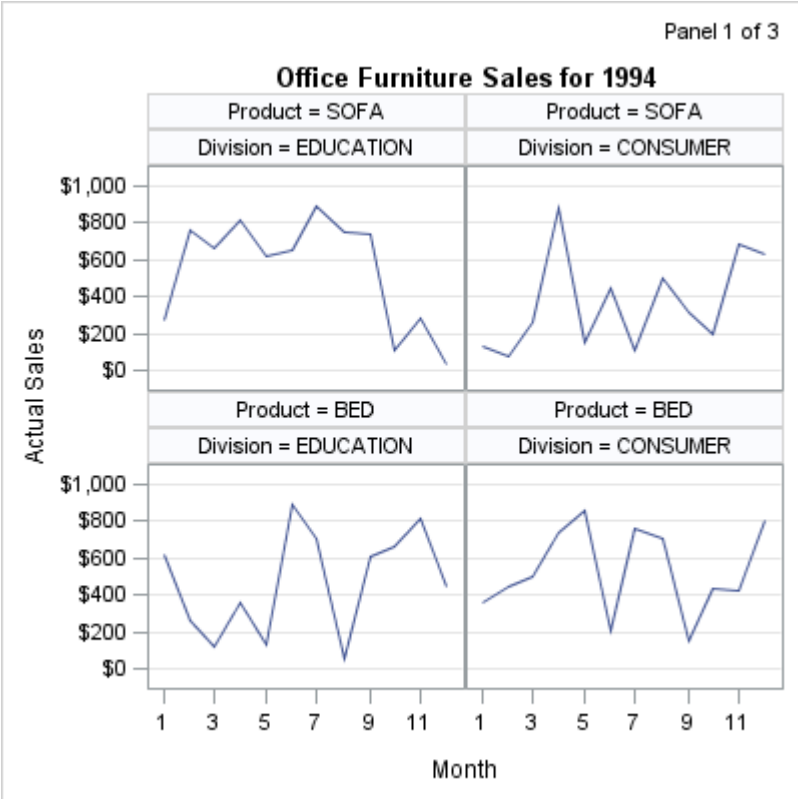
The SKIPEMPTYCELLS= option also applies to a DATALATTICE layout. The following output shows the last panel when Division has two levels and Product has three levels, while ROWS=2 and COLUMNS=2. When SKIPEMPTYCELLS=FALSE, the last panel will have a column of empty cells. Entire rows or columns of empty cells can be removed by setting SKIPEMPTYCELLS=TRUE.

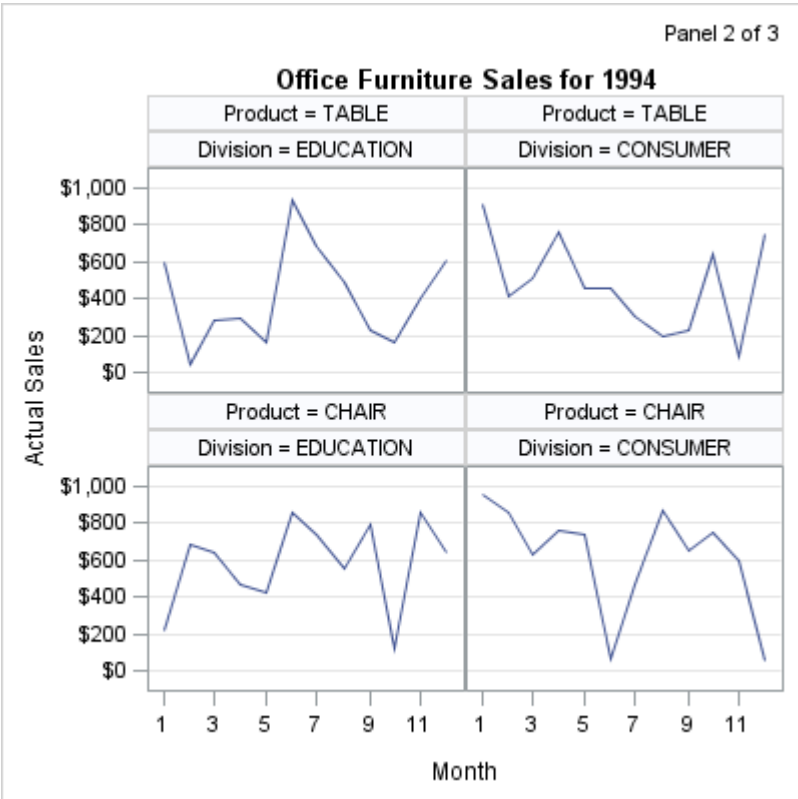

COLUMNVAR=Product
ROWVAR=Division
SKIPEMPTYCELLS=FALSE
Product = DESK
Division = EDUCATION
Division = CONSUMER
Feb
Apr
Jun
Aug
Oct
Dec
$500
$2,000

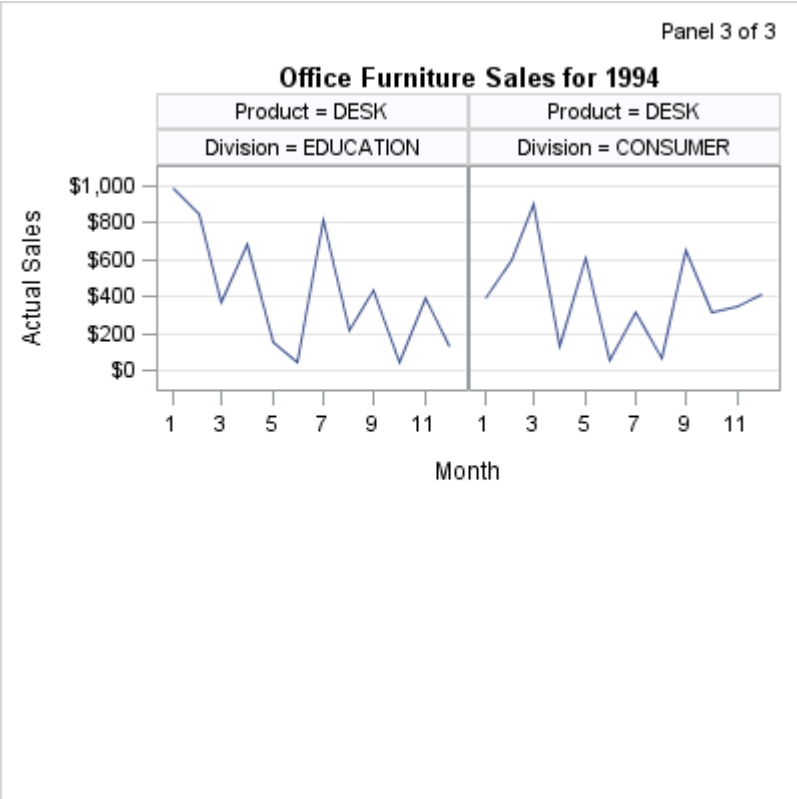


User Control of Panel Generation

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
$800
$600
$400
$200
$0
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
$800
$600
$400
$200
$0
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.

```
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;
```

The PANELNUMBER=PANELNUM setting is a directive indicating which panel to produce. The ENTRYTITLE statement changes as the panel number changes. For more information about how to pass information to a template at run time, see Chapter 30, “Using Dynamic Variables and Macro Variables in Your Templates,” on page 605.

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. Here is the macro definition.

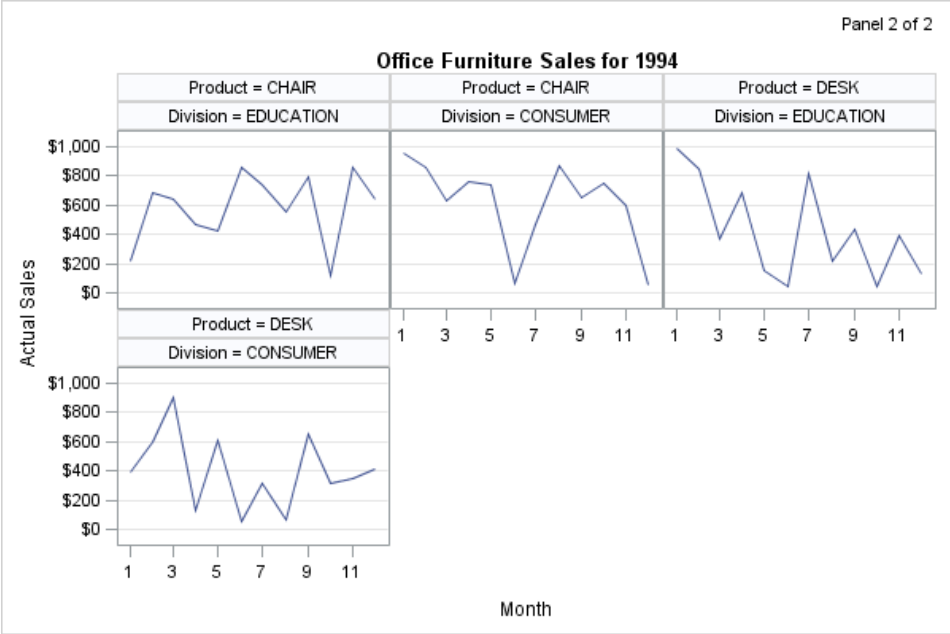
```
%macro panels(rows=1,cols=1,colwidth=200,year=1994,style=htmlblue);
  %local div_vals prod_vals panels totpanels panelnumber;

  /* 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
*/

  /* generate the panels */
  ods graphics / reset;
  ods _all_ close;
  ods listing style=&style gpath="path-to-image-folder";
  %do panelnum=1 %to &totpanels;
    ods graphics / imagename="Panel&panelnum"
      width=%sysevalf(&colwidth*&cols)px
      height=%sysevalf(&colwidth*&rows)px;
    proc sgrender data=sashelp.prdsale template=panelgen;
      where country="U.S.A." and region="EAST" and year=&year;
    run;
  %end;

  ods listing close;
  ods html; /* Not necessary in SAS Studio */
%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:

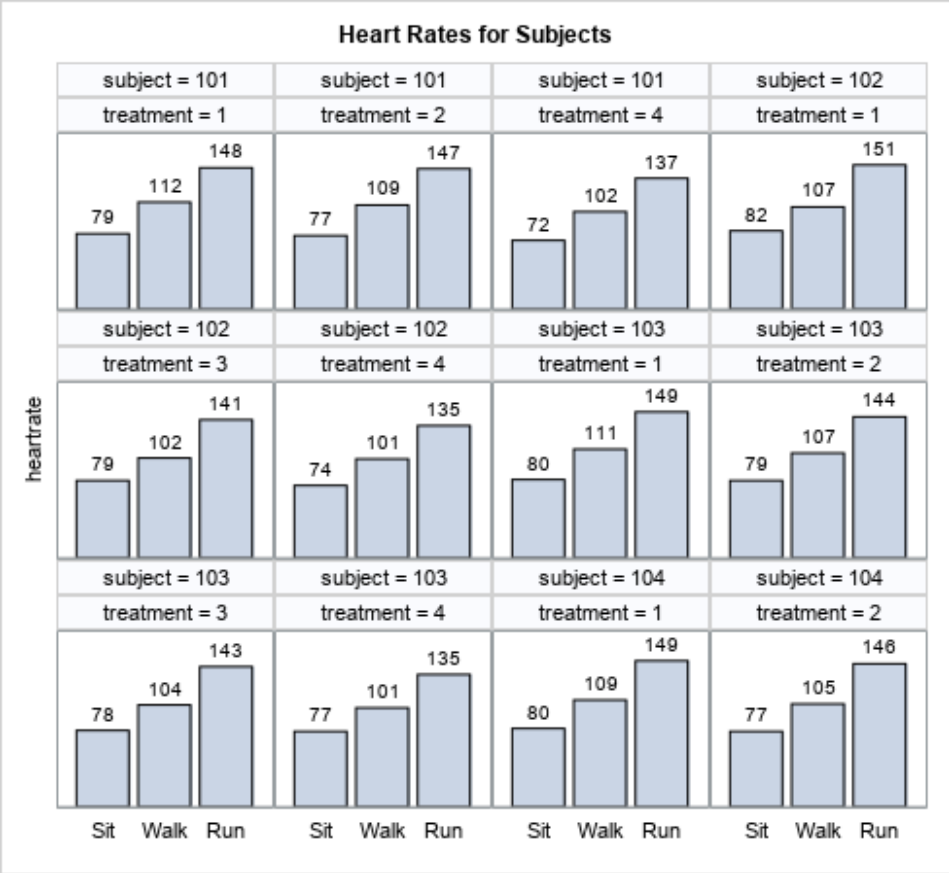
```
layout datapanel classvars=(state city) / rows=4 columns=5;
```

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.

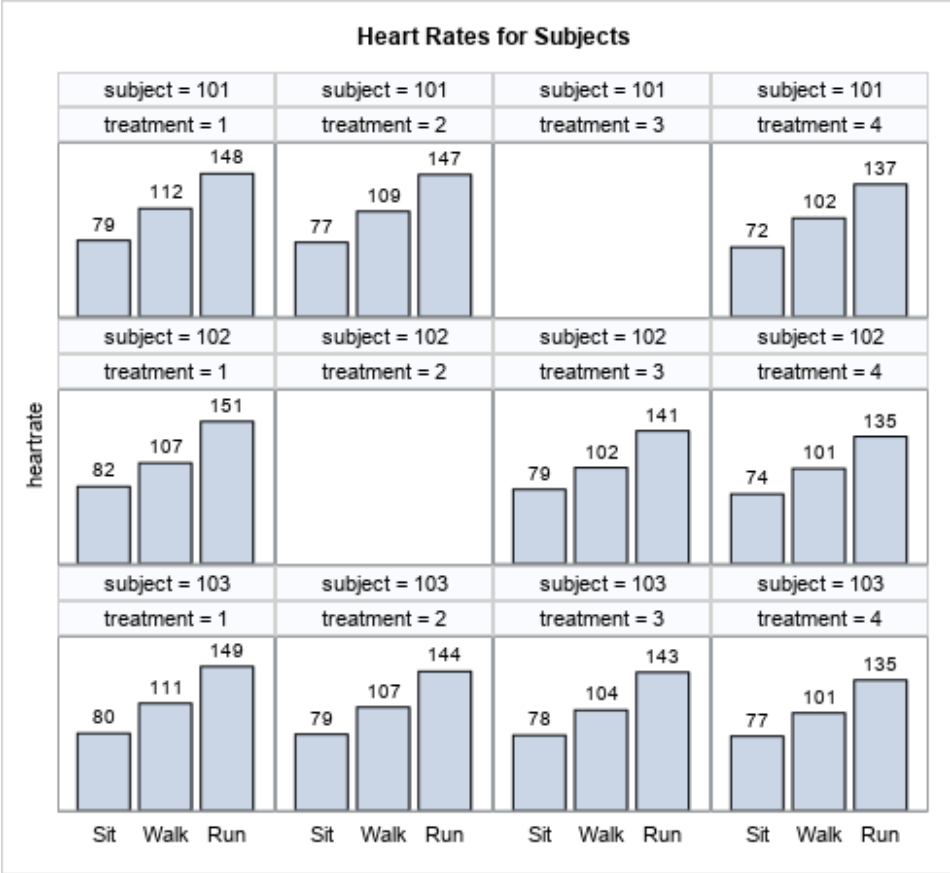
Sometimes there are unexpected gaps in the data when classification variables are crossed. For example, suppose you are conducting a study where a number of subjects each receives over time four treatments that might lower the subject's heart rate after various amounts of physical activity. However, assume that Subject 101 did not get Treatment 3, and Subject 102 did not 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 as shown in the following figure.



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 as shown in the following template:

```
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 category=task response=heartrate / barlabel=true;
      endlayout;
    endlayout;
  endgraph;
 end;
run;
```

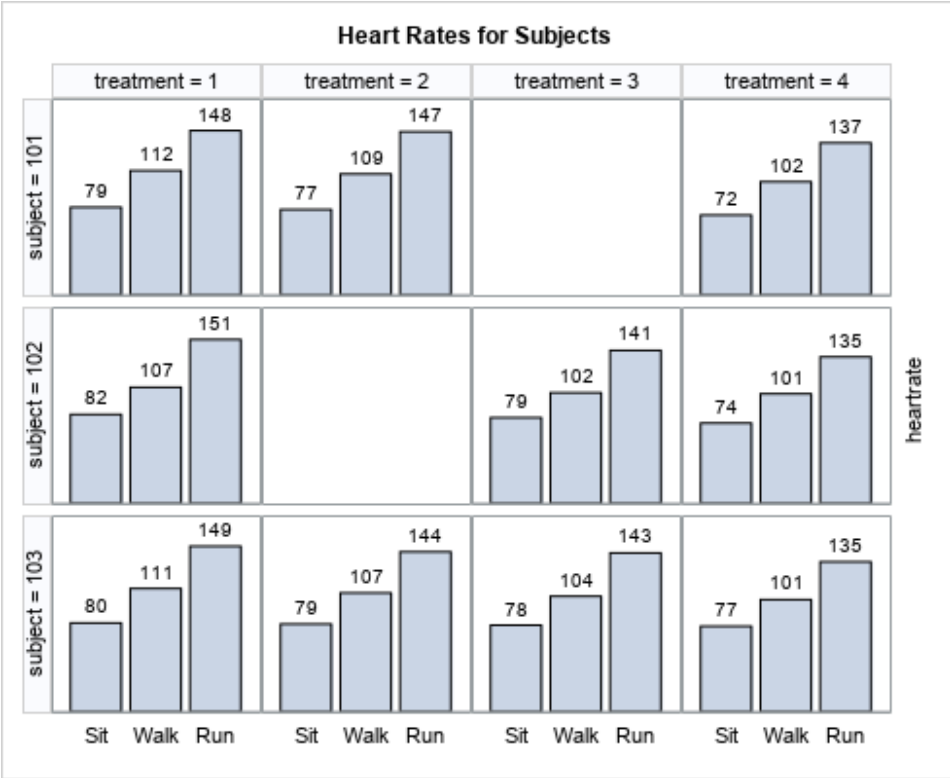
Here is the result.



For the DATALATTICE layout, the SPARSE= option does not apply because they are inherently sparse. When you specify two classifiers, the DATALATTICE layout manages this situation automatically. Here is a template that uses the DATALATTICE layout for this example.

```
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 category=task response=heartrate / barlabel=true;
      endlayout;
    endlayout;
  endgraph;
 end;
run;
```

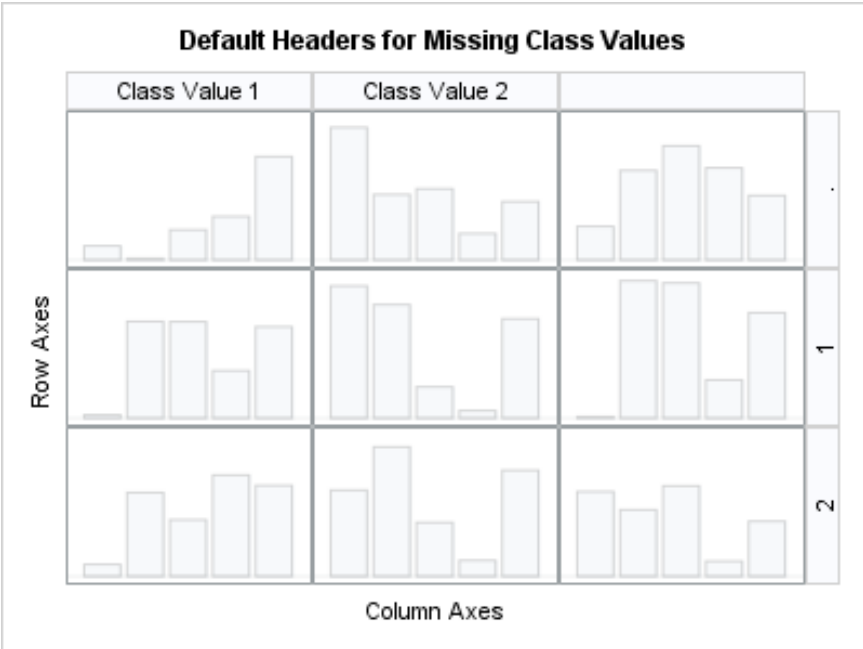
Here is the result.



Missing Class Values

By default, missing class values are included in the classification levels for the panel. When the data contains missing classification values, cells are created in the panel for the missing classes. The classification headers for the missing values are either blank for missing string values or a dot for missing numeric values. The following figure shows a data lattice in which both the row and column classification variables contain missing values.

Figure 16.3 *Data Lattice with Missing Class Values*



In the top right cell, the header for the missing column classification values is blank and the header for the missing row column classification values is a dot. If you want to remove the missing values from the lattice, specify the INCLUDEMISSINGCLASS=FALSE option in the LAYOUT DATALATTICE statement. If you prefer to keep the missing values, you can create a format that specifies a more meaningful header for the missing classes. The following code defines a format for the missing class values in the previous example.

```
proc format;
    value missingrowclass
      . = "Missing Row";
    value $missingcolclass
      " " = "Missing Column";
run;
```

A single space enclosed in quotation marks specifies a missing character value, and a dot specifies a missing numeric value. Using empty quotation marks (" or "") to specify a missing character value produces unexpected results. To specify a missing character value, enclose a single space in single or double quotation marks (' ' or " ").

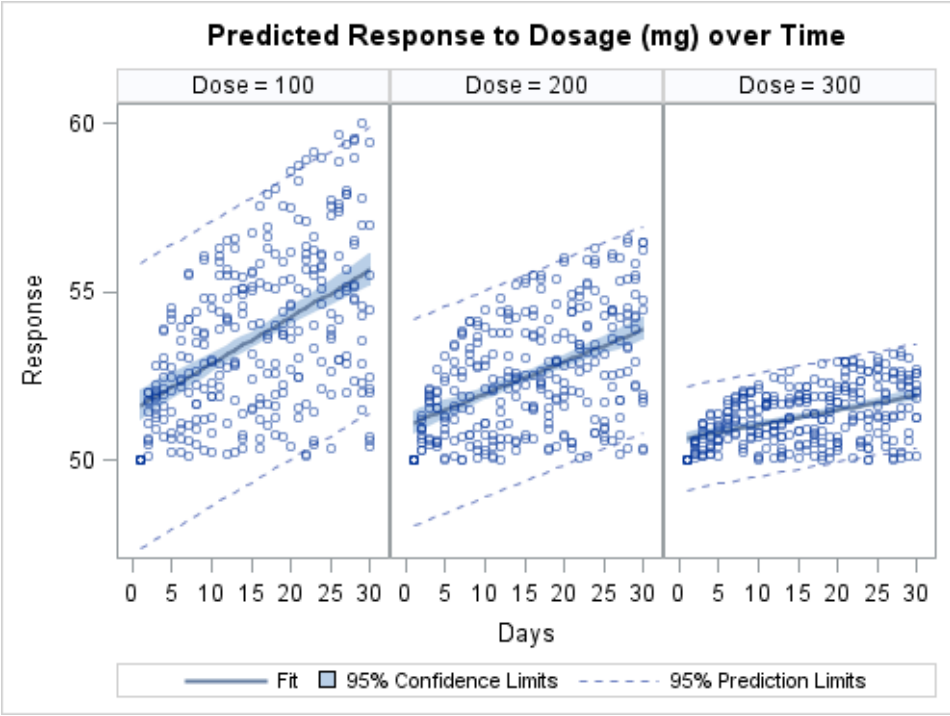
The format is applied to the classification variables in the PROC SGRENDER statement as shown in the following example. For the missing data, the labels specified in the format statement are used as the headers for the missing classes. The following clip shows the top right cell in Figure 16.3 on page 289 when these formats are applied to the classification variables.

Note: In SAS 9.4M2 and in earlier releases, ODS Graphics does not support Unicode values in user-defined formats. Starting with SAS 9.4M2, ODS Graphics supports Unicode values in user-defined formats only if they are preceded by the (*ESC*) escape sequence. For an example of how to use Unicode values in a user-defined format, see "Formatting the Tick Values on a Discrete Axis" on page 141.

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;

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

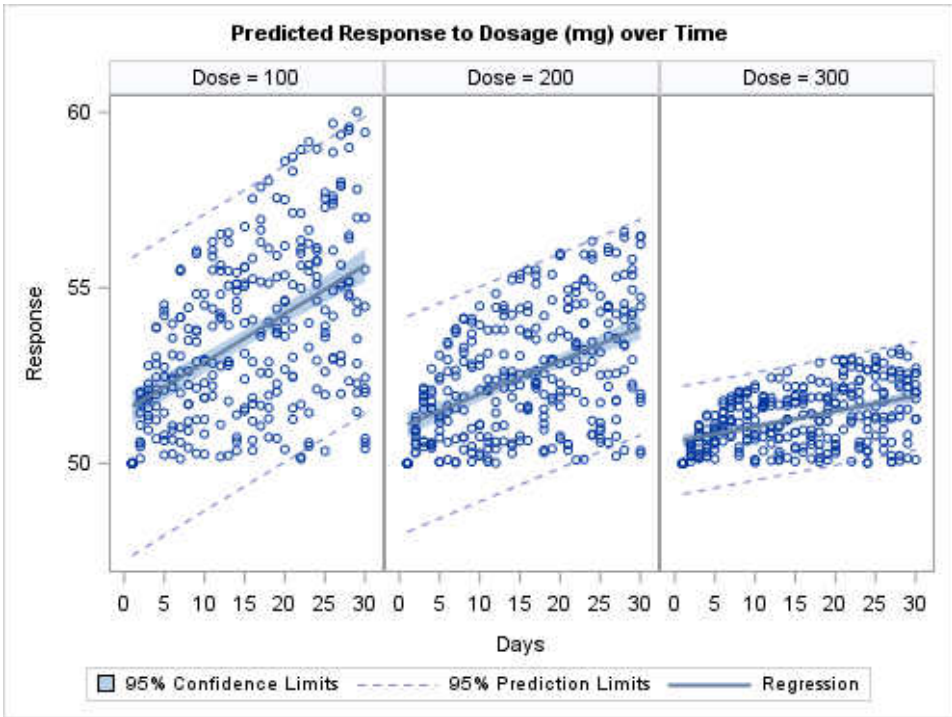
For both a DATAPANEL and DATALATTICE layout, you can use the INSET= option in the layout statement to display information about one or more variables in an inset in each cell. For each variable specified in the INSET= option, the inset displays the variable label and value. You can use the INSETOPTS= option to specify the alignment, background color, and border of the inset. Starting with SAS 9.4M1, you can also use the CONTENTDISPLAY= option to control the information that is displayed in the inset. For more information, see *SAS Graph Template Language: Reference*. For an example, see "Example 4: A Data Panel with an Inset in Each Cell " on page 301.

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 290, you might find it easier to create the panel by using PROC SGPANEL in SAS 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.

```
title "Predicted Response to Dosage (mg) over Time";
proc sgpanel data=trial;
  panelby dose / rows=1;
  reg x=days y=response / cli clm;
run;
```

Here is the output.



Most, but not all, features of the DATALATTICE and DATAPANEL layouts are provided in the SGPANEL procedure.

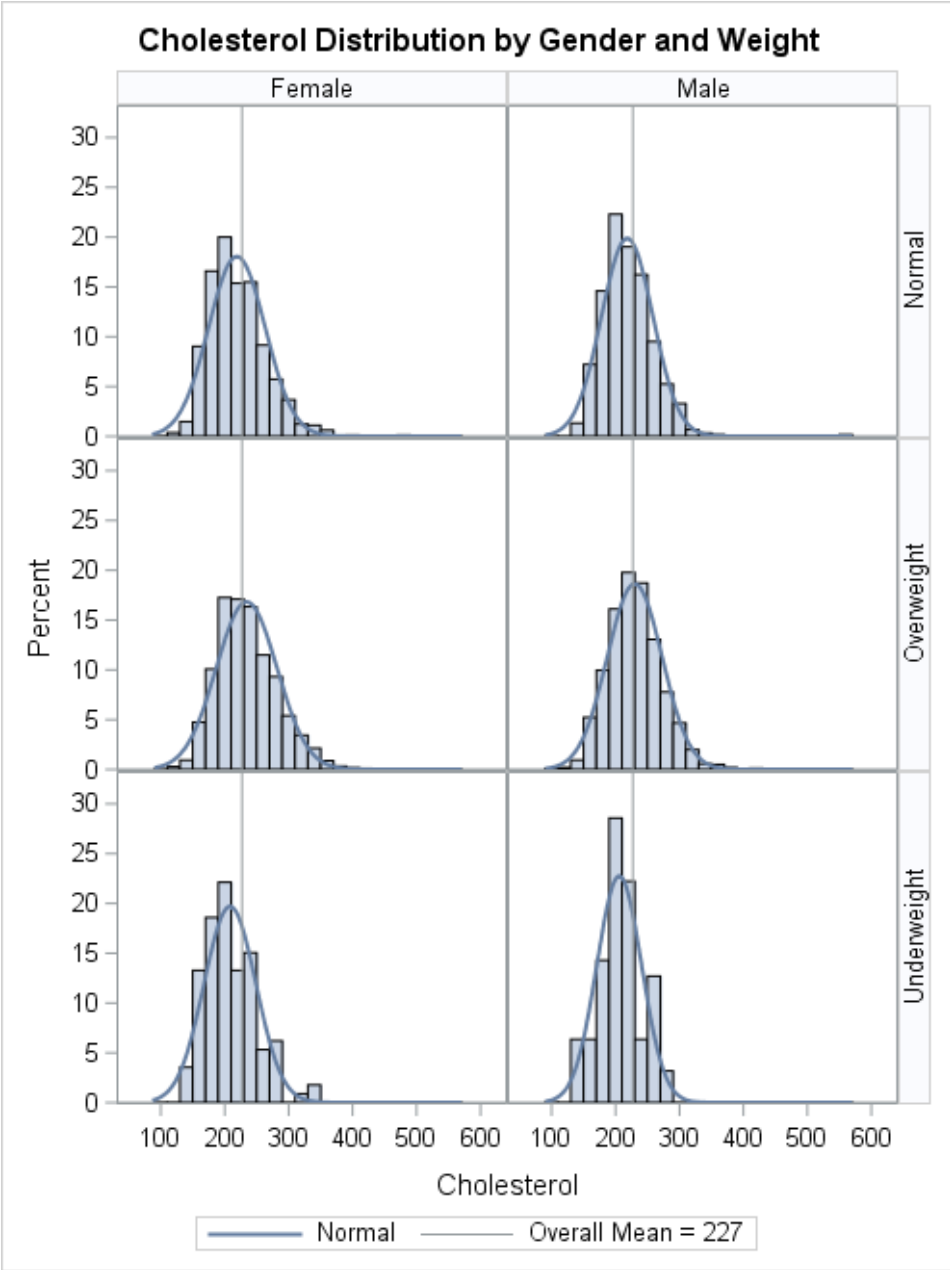
The SGPANEL procedure supports computed plot statements such as HISTOGRAM, DENSITY, DOT, VBOX, and HBOX (vertical and horizontal box plots). The PANELBY statement controls the layout, determining whether a DATAPANEL, DATALATTICE, or other layout is used to produce the graph. ROWAXIS and COLAXIS statements 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 *SAS ODS Graphics: Procedures Guide* for details.

The following example shows additional features of SGPANEL:

```
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;
```

Here is the output.


Cholesterol Distribution by Gender and Weight
Female
Male
Normal
Overweight
Underweight
Percent
0
5
10
15
20
25
30
100
200
300
400
500
600
Cholesterol
Normal
Overall Mean = 227

Examples: Data Lattice Layout and Data Panel Layout

Example 1: A Basic Data Lattice

This example uses the LAYOUT DATALATTICE statement to specify two classification variables: DIVISION and PRODUCT. The following criteria apply to the LAYOUT DATALATTICE statement:

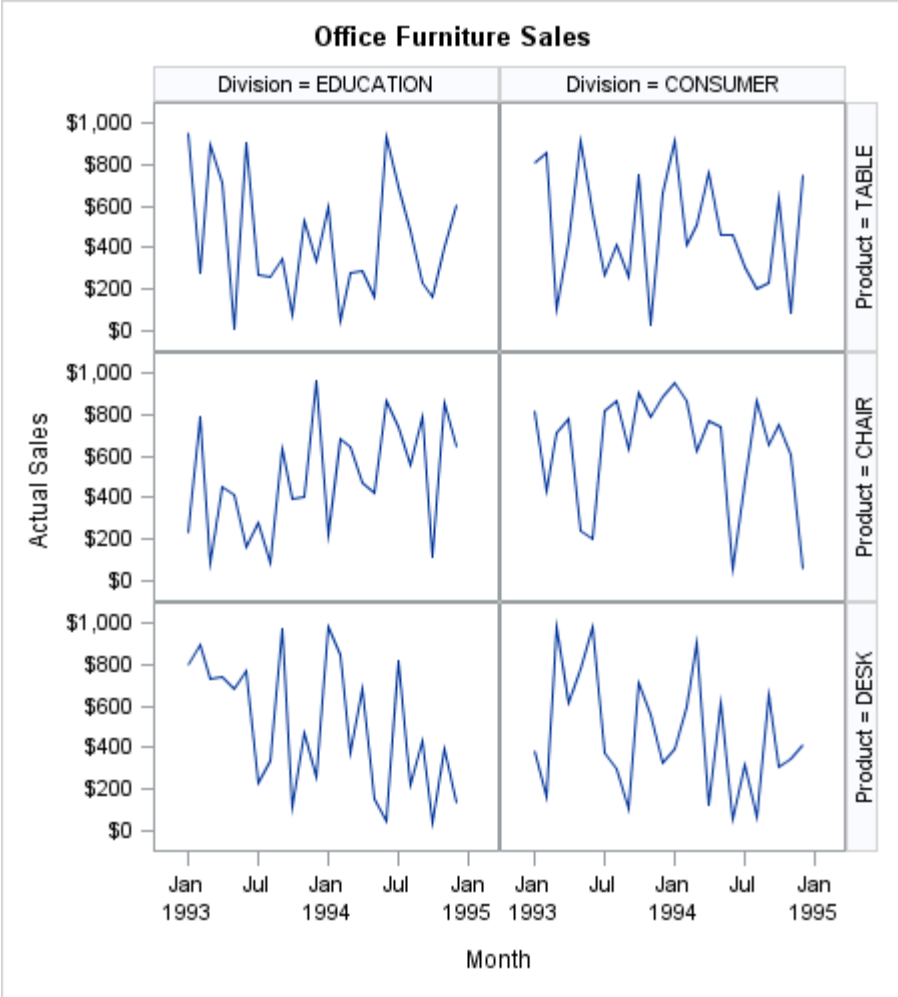
- One of the ROWVAR= or COLUMNVAR= arguments is required. You can specify both. Each argument specifies a single classification variable, enabling you to specify either one or two classifiers for the graph.
- In the resulting graph, the data crossings are identified by row or column headers.
- The default 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.

Here is the code for this example.

```
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;
```

Here is the output.

Figure 16.4 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.

Example 2: A Basic Data Panel

This example 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.

The following criteria apply to 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.

Here is the code for this example.

```
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;
```

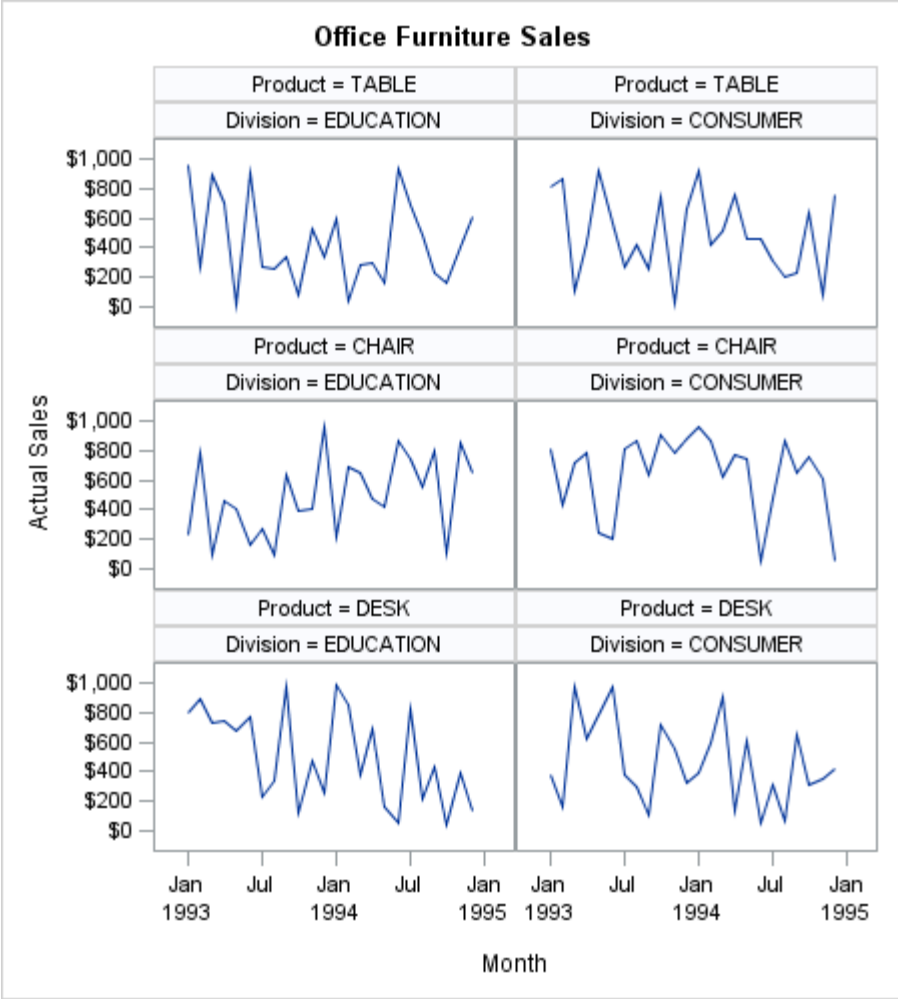
In the template code, notice the LAYOUT PROTOTYPE block, which is inside the LAYOUT DATAPANEL block. This nested block, which is a required part of the DATAPANEL layout, defines the graphical content of all of the cells. The COLUMNS=2 setting forces a DATAPANEL layout to display the cells in a two-column 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. When you render the graph for a classification panel, you can 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;
```

Here is the output.

Figure 16.5 Classification Panel Created with LAYOUT DATAPANEL



Example 3: A Data Panel with Sidebars

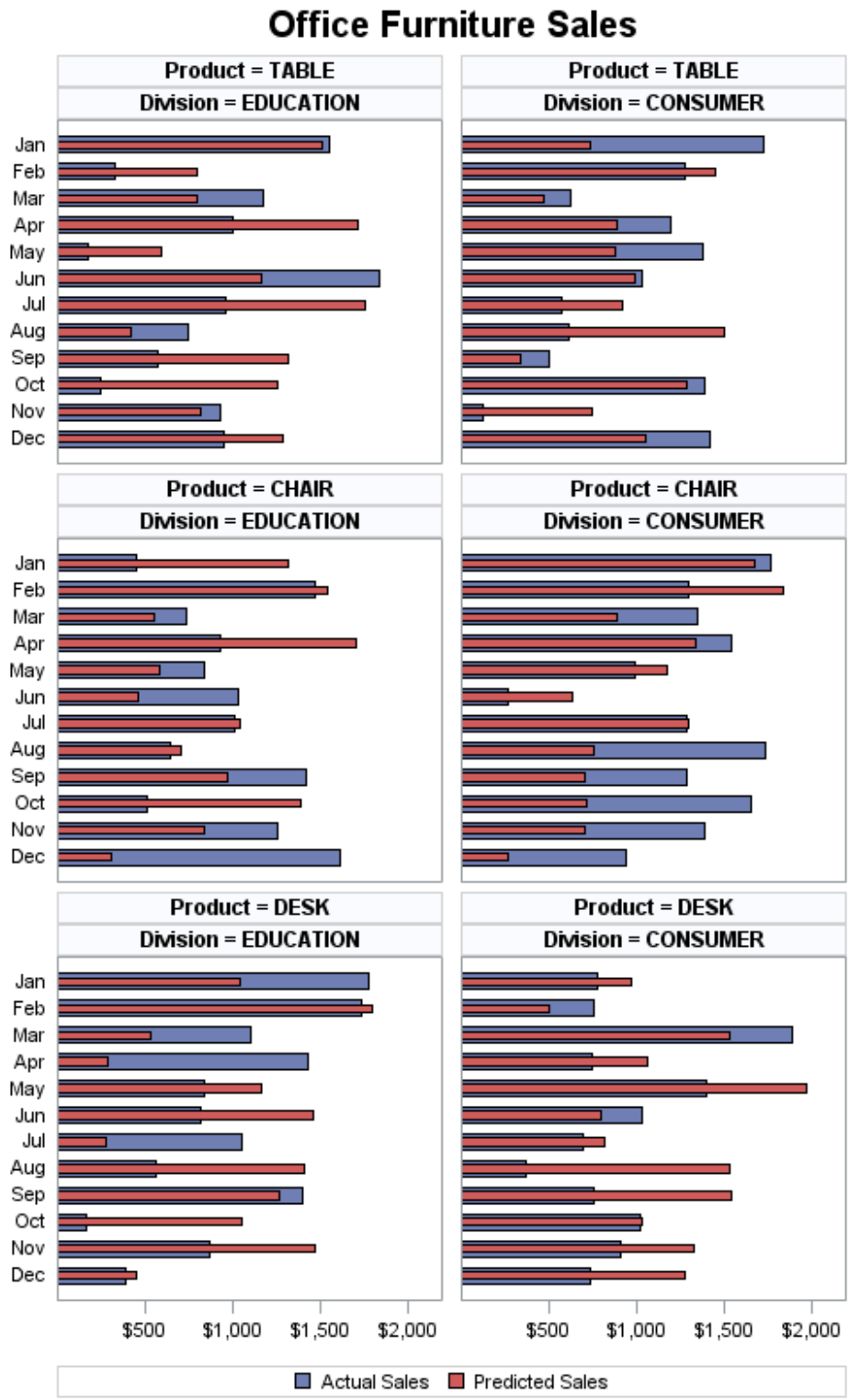
Sidebars are useful for aligning information outside of the grid. In the following example, a sidebar is used to display a graph title, rather than using an ENTRYTITLE statement. The advantage of using sidebars for title and footnote information is that a sidebar is always horizontally aligned on the grid itself, not on the complete graph width. Of course, you have to specify the title text in 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;
```

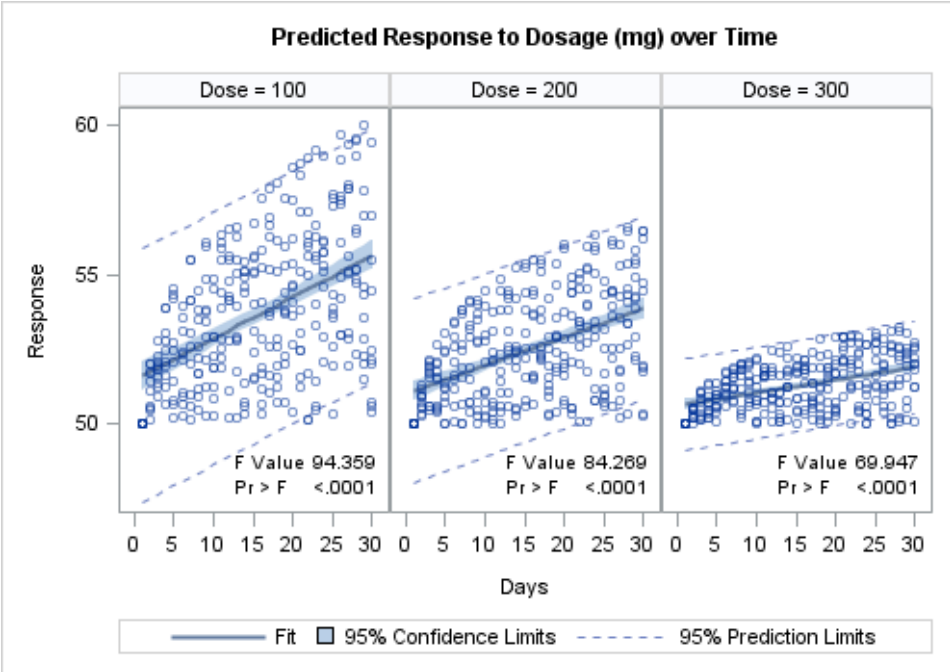
```
      layout datapanel classvars=(product division) / columns=2
        columngutter=10  rowgutter=5
        headerlabelattrs=GraphLabelText(weight=bold)
        rowaxisopts=(display=(tickvalues))
        columnaxisopts=(display=(ticks tickvalues)
          offsetmin=0
          linearopts=(tickvalueformat=dollar6. 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 category=month response=actual  /
            orient=horizontal fillattrs=GraphData1
              barwidth=.6 name="actual";
          barchart category=month response=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;
```

Here is the output.


Office Furniture Sales
Product = TABLE
Division = EDUCATION
Product = TABLE
Division = CONSUMER
Product = CHAIR
Division = EDUCATION
Product = CHAIR
Division = CONSUMER
Product = DESK
Division = EDUCATION
Product = DESK
Division = CONSUMER
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
$500
$1,000
$1,500
$2,000
Actual Sales
Predicted Sales

Example 4: A Data Panel with an Inset in 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:



Here is the template code.

```
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, note the following:

- 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 Inset DATA step selects the appropriate three observations from the Outstat data set. The F and PROB variables are assigned labels and formats.
- The Stats2 DATA 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 401 discusses 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.

17

Creating Graphs with No Axis Using the REGION Layout

The LAYOUT REGION Statement

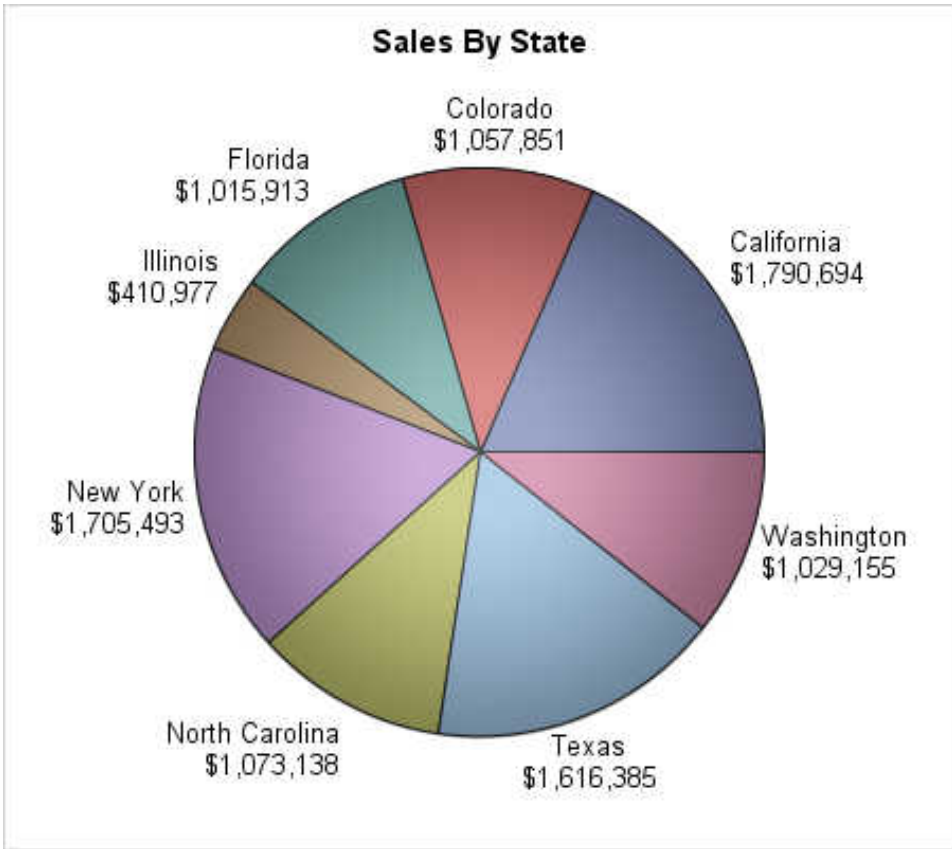
The LAYOUT REGION statement supports plots that have no axes, such as pie charts. It also supports annotations such as text, lines, arrows, images, and geometric shapes, that are generated with the draw statements. For information about adding annotations, see Chapter 22, “Adding Code-Driven Graphics Elements to Your Graph,” on page 425. The LAYOUT REGION statement allows only one plot statement. If you include more than one, the additional plot statements are ignored. You can include DISCRETELEGEND, CONTINUOUSLEGEND, and ENTRY statements with your region plot statement. You can use a LAYOUT REGION statement to define a top-level region container or to define a region container for a single cell in a LAYOUT GRIDDED or LAYOUT LATTICE statement. You can also nest a REGION layout in other layouts, such as GRIDDED, LATTICE, and overlay-type layouts. However, the REGION layout cannot be nested in DATAPANEL and DATALATTICE layouts.

Note: When you nest a REGION layout in an overlay-type layout, you must include the HEIGHT=, WIDTH=, VALIGN=, and HALIGN= options in your LAYOUT REGION statement to specify the size and alignment of the plot.

Examples: Region Layout

Example 1: Pie Chart

This example uses a LAYOUT REGION statement to create the pie chart that is shown in the following figure.



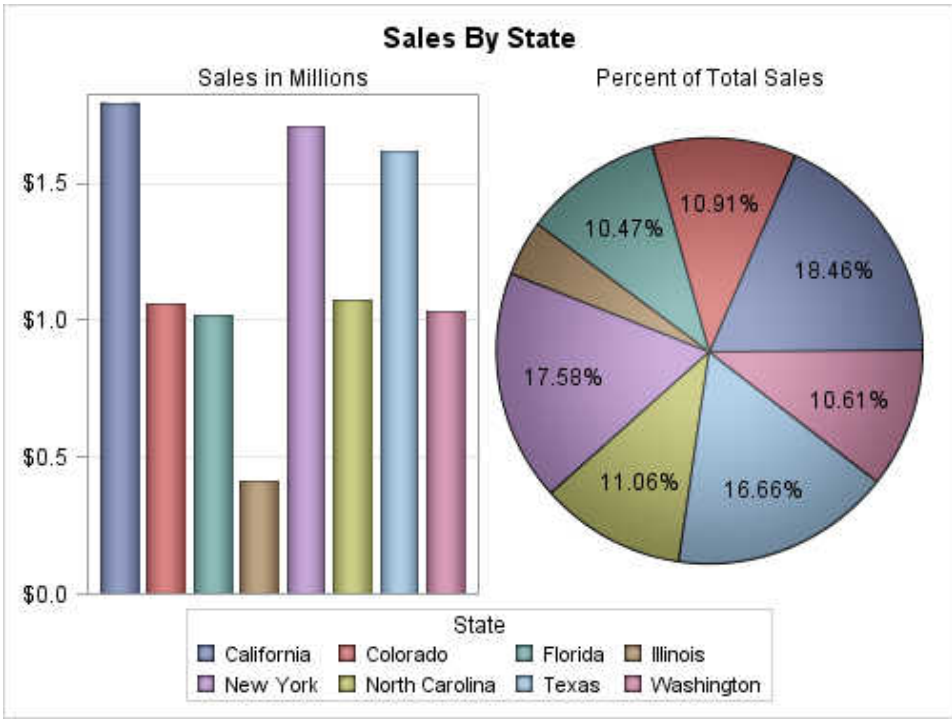
Here is the SAS code that generates this chart.

```
/* Define the template for the chart */
proc template;
define statgraph region;
  begingraph;
    entrytitle "Sales By State";
    layout region;
        piechart category=state response=actual /
          dataskin=pressed datalabellocation=outside;
    endlayout;
  endgraph;
end;
run;
```

```
quit;

/* Set the antialias maximum to 15000 */
ods graphics on / reset antialiasmax=15000;

/* Render the chart */
proc sgrender data=sashelp.prdsal2 template=region;
  where country eq "U.S.A.";
  format actual dollar12.0;
run;
quit;
```

Example 2: Bar Chart and Pie Chart

This example nests a LAYOUT OVERLAY and a LAYOUT REGION statement in a LAYOUT LATTICE block to create the charts shown in the following figure.



Here is the SAS code that generates these charts.

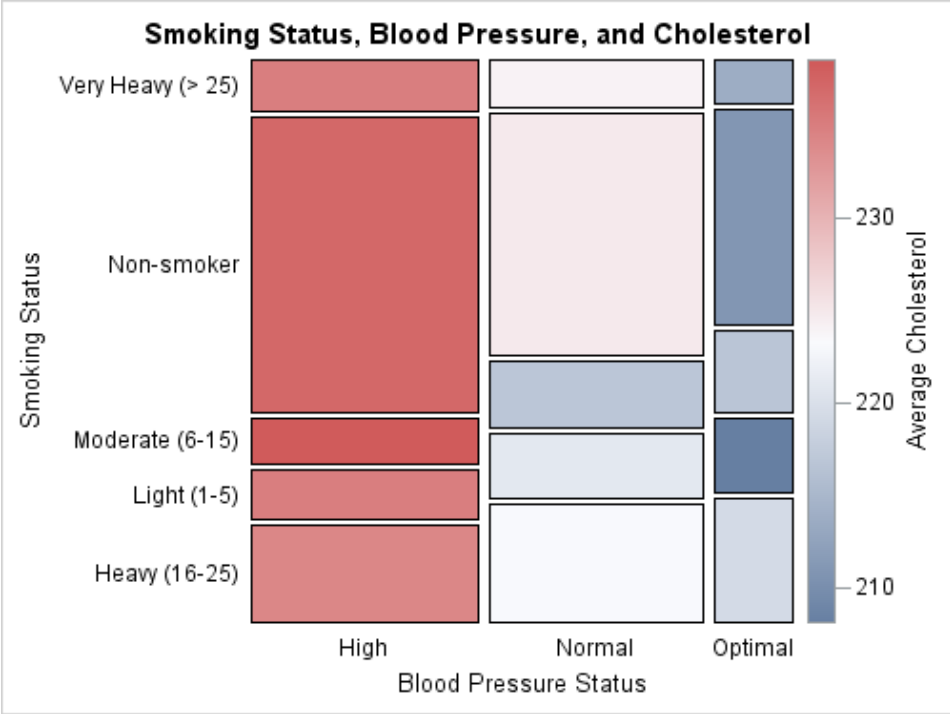
```
/* Get sales data in millions from SASHELP.PRDSAL2 */
data sales;
  set sashelp.prdsal2;
  actualmils=(actual / 1000000);
run;

/* Define the template for the graph */
proc template;
  define statgraph region;
    begingraph;
      entrytitle "Sales By State";
```

```
      layout gridded / columns=1 rows=2;
        layout lattice / columns=2 rows=1;
          cell;
            /* Generate a bar chart of sales in millions. */
            cellheader;
              entry "Sales in Millions";
            endcellheader;
            layout overlay / width=250px
              xaxisopts=(display=none)
              yaxisopts=(griddisplay=on display=(ticks tickvalues));
              barchart category=state response=actualmils /
                group=state
                orient=vertical
                dataskin=pressed barwidth=0.8;
            endlayout;
          endcell;

          cell;
            /* Generate a pie chart of percent of total sales. */
            cellheader;
              entry "Percent of Total Sales";
            endcellheader;
            layout region / pad=10;
             piechart category=state response=actualmils /
               name="salespct"
               dataskin=pressed
               datalabelcontent=(percent)
               datalabellocation=inside labelfitpolicy=drop;
            endlayout;
          endcell;
        endlayout;
        discretelegend "salespct" / title="State" across=4;
      endlayout;
    endgraph;
  end;
run;
quit;

/* Set the antialias maximum to 15000 */
ods graphics on / reset antialiasmax=15000;

/* Render the graph */
proc sgrender data=sales template=region;
  where country eq "U.S.A.";
  format actualmils dollar5.1;
run;
quit;
```

Notice that the LAYOUT OVERLAY statement defines the layout for the bar chart in the first cell in the lattice. Also, the LAYOUT REGION statement defines the layout for the pie chart in the second cell. You can use the LAYOUT REGION statement with other layout statements to combine region plots and other types of plots in a single LAYOUT GRIDDED or LAYOUT LATTICE statement.

Example 3: Mosaic Plot

This example uses a LAYOUT REGION statement to create the mosaic plot that is shown in the following figure.



Here is the SAS code that generates this plot.

```
/* Summarize the SASHELP.HEART data for BP_STATUS and SMOKING_STATUS */
proc summary data=sashelp.heart nway;
  class bp_status smoking_status;
  var cholesterol;
  output out=heart mean=avgCholesterol N=count / noinherit;
run;

/* Define the template for the plot */
proc template;
  define statgraph mosaicplot;
    begingraph;
      entrytitle "Smoking Status, Blood Pressure, and Cholesterol";
      layout region;
       mosaicPlotParm category=(bp_status smoking_status) count=count /
          name="mosaic" colorresponse=avgCholesterol;
       continuouslegend "mosaic" / title="Average Cholesterol"
          pad=(left=5);
      endlayout;
    endgraph;
  end;
run;
```

```
/* Render the plot */
proc sgrender data=heart template=mosaicplot;
run;
```

18

Plotting a SAS Cloud Analytic Services (CAS) In-Memory Table

Accessing In-Memory Tables

Starting with SAS 9.4M5, the SGRENDER procedure can access an in-memory table through a CAS engine libref. In that case, the data is downloaded from SAS Cloud Analytic Services to the SAS client through the CAS engine libref, and then it is processed on the SAS client by the SGRENDER procedure. The amount of data that can be transferred from the CAS server to the SAS client is limited by the following options:

- The ODS GRAPHICS statement option MAXOBS= limits the number of observations that can be read from any in-memory table by only ODS Graphics procedures.See "ODS GRAPHICS" in *SAS Graph Template Language: Reference*.
- System option CASDATAMAX= limits the amount of data in bytes that can read from any in-memory table in any caslib in the SAS client. See "CASDATALIMIT= System Option" in *SAS Cloud Analytic Services: User's Guide*.
- Option DATALIMIT= limits the amount of data, in bytes, that can be read from any in-memory table in a specific caslib or from a specific in-memory table. See the DATALIMIT= LIBNAME statement option and the DATALIMIT= data set option.

If a data limit is exceeded, the SGRENDER procedure step terminates, and an error message is written to the SAS log. Although you can use one of these options to increase the maximum, processing a large amount of data locally might degrade

SAS performance. For a very large in-memory table, use a procedure that is supported by the CAS server or use a CAS action to summarize or otherwise reduce the data, and then use GTL to plot the results.

For information about SAS Cloud Analytic Services programming in SAS, see *An Introduction to SAS Viya Programming*. For information about how to use a CAS engine libref to access in-memory tables, see "CAS LIBNAME Statement" in *SAS Cloud Analytic Services: User's Guide*. For an example of how to use a CAS engine libref with the SGRENDER procedure, see "Example: Preprocessing Data in CAS and Plotting the Results Using GTL" on page 311.

About In-Memory Tables and Data Order

For GTL graphs, data order determines many attributes of the graph such as group order, item order in discrete legends, and tick-value order on discrete axes. It also determines the order in which many of the graphical elements are drawn. In many cases, a change in data order can result in a change in the graph appearance. For example, the assignment of GraphData1–GraphDataN style elements can change, as can the order in which some graphical elements are overlaid. Data order for local SAS data is static. Graphs rendered with local data remain consistent over multiple renderings. In contrast, data order for in-memory tables can be unpredictable. The data order might vary over multiple graph renderings when an in-memory table is used.

To ensure consistent graphs when you are plotting an in-memory table, data accessed through a CAS engine libref is sorted automatically by default in ascending order before it is plotted. As a result, the default value for options that control order such as GROUPORDER= for grouped plots, SORTORDER= for columns and rows in a data lattice or data panel, and so on, might differ between local SAS data and in-memory tables. The default value for some other options might also differ between local SAS data and in-memory tables. For each of these options, the default values for both cases are listed in *SAS Graph Template Language: Reference*.

GTL Features That Do Not Support In-Memory Tables

Some features in GTL require a specific data order. Because data order can be unpredictable when you are using an in-memory table, you must store the data for these features in a SAS data set. These features include the following:

- the POLYGONPLOT statement. The polygon is dropped if the plot data is an in-memory table. See "POLYGONPLOT" in *SAS Graph Template Language: Reference*.

- the annotation facility. The SGRENDER procedure option SGANNO= is ignored if it specifies an in-memory table. See "About the GTL Annotation Facility" in *SAS Graph Template Language: Reference*.
- discrete attribute maps. The SGRENDER procedure option DATTRMAP= is ignored if it specifies an in-memory table. See "Key Concepts for Using Attribute Maps" in *SAS Graph Template Language: Reference*.

Example: Preprocessing Data in CAS and Plotting the Results Using GTL

Here is an example that shows how to summarize an in-memory table in CAS and how to plot the result using GTL. For very large in-memory tables, it is recommended that you preprocess the data in CAS to reduce the size of the data before you plot it using GTL. This example uses the data in SAS data set Sashelp.Cars as sample data. The MDSUMMARY procedure is used to summarize the data before it is plotted. It is supported by the CAS server and can process a very large in-memory table. For information about the MDSUMMARY procedure, see "PROC MDSUMMARY Statement" in *SAS Cloud Analytic Services: User's Guide*.

Here is the SAS code for this example.

```
*options cashost="cloud.example.com" casport=5570; /* 1 */
cas casauto sessopts=(caslib=casuser);             /* 2 */

libname mycas cas;                                 /* 3 */

proc casutil;                                      /* 4 */
  load data=sashelp.cars casout="cars";
quit;

proc mdsummary data=mycas.cars;                    /* 5 */
  var mpg_highway;
  groupby type origin;
  output out=mycas.cars_summary;
run;

proc print data=mycas.cars_summary;                /* 6 */
  var origin type _mean_;
run;

proc template;                                     /* 7 */
  define statgraph meanmpg;
    begingraph;
      entrytitle "Average Highway MPG and Vehicle Type";
      layout overlay /
        xaxisopts=(label="Vehicle Type")
        yaxisopts=(label="Average Highway MPG");
        barchartparm category=type response=_mean_ /
          name="barchart" group=origin;
```

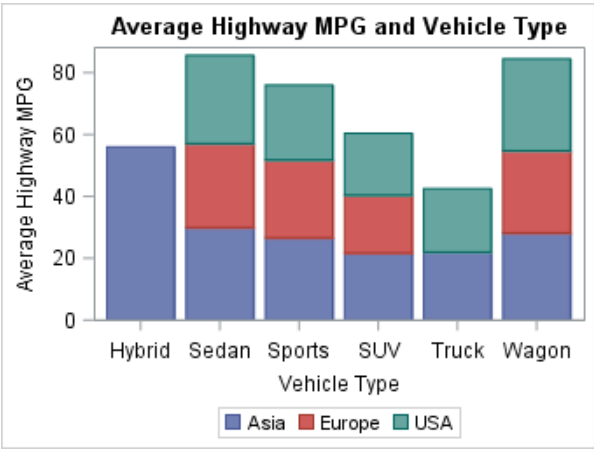
```
        discretelegend "barchart";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=mycas.cars_summary              /* 8 */
  template=meanmpg;
run;

cas casauto terminate;
```

1. If necessary, set system options CASHOST= and CASPORT= to the host name and port for your CAS server.
2. Connect to your CAS server. Specify caslib CASUSER as the active caslib.
3. Create a CAS engine libref to the active caslib, CASUSER.
4. Load the SAS data set Sashelp.Cars into the in-memory table Cars.
5. Use the MDSUMMARY procedure to summarize the variable Mpg_Highway, and group the results by the variables Type and Origin. Store the results in the in-memory table Cars_Summary in the caslib CASUSER.
6. Optional: Print columns Type, Origin, and _Mean_ in the table Cars_Summary.
7. Create the template for your graph. Use the parameterized plot statement BARCHARTPARM to create a bar chart of variables _Mean_ and Type, grouped by variable Origin in the summarized data.
8. Use the SGRENDER procedure to render the graph in the template MEANMPG using the data Mycas.Cars_Summary.

The summarized data generated by the MDSUMMARY procedure contains only 15 observations, as shown in the following image.

Obs	Origin	Type	_Mean_
1	Asia	Truck	22
2	Europe	Sedan	27.115384615
3	Asia	Sedan	29.968085106
4	USA	Sedan	28.544444444
5	USA	Wagon	29.714285714
6	Asia	Sports	26.647058824
7	Asia	Hybrid	56
8	Asia	Wagon	28.181818182
9	Europe	Sports	25.130434783
10	Europe	SUV	18.7
11	USA	Sports	24.222222222
12	Asia	SUV	21.68
13	USA	SUV	20.04
14	USA	Truck	20.5
15	Europe	Wagon	26.583333333

Here is the resulting bar chart.



Notice that the group values and the category axis tick values are arranged in ascending order. As described in "About In-Memory Tables and Data Order" on page 310, group values and discrete-axis tick values are automatically sorted in ascending order by default when an in-memory table is used.

For more information about SAS Cloud Analytic Services programming in SAS, see *An Introduction to SAS Viya Programming*.

PART 5

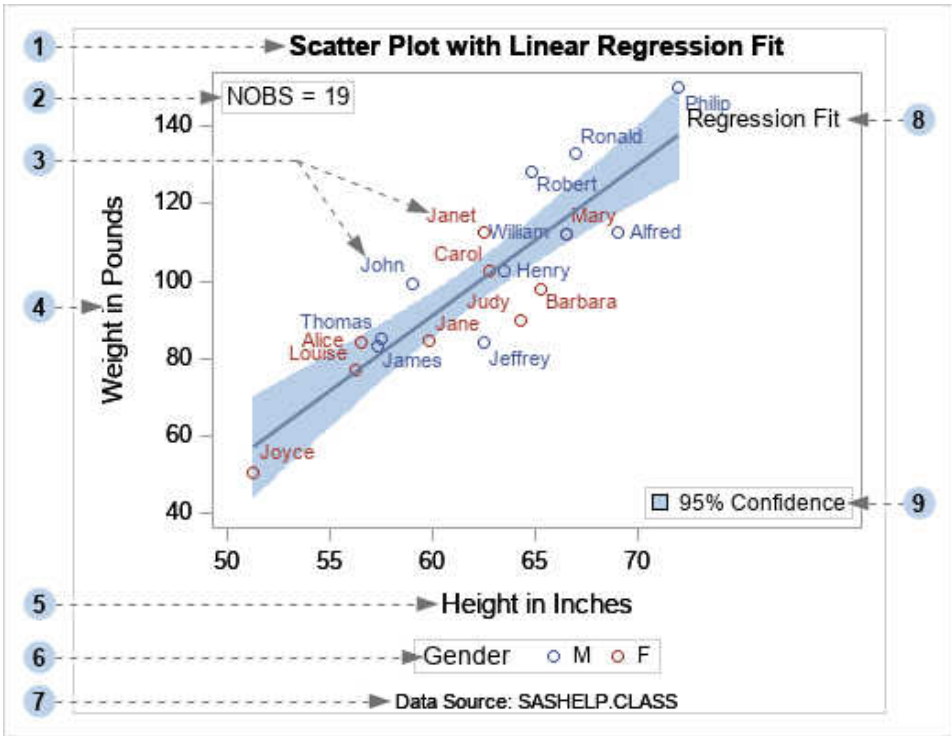
Adding Text, Legends, and Insets

19

Adding Titles, Footnotes, and Text Entries to Your Graph

Text Strings in Graphs

Using 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.



1 ENTRYTITLE statement

2 ENTRY statement

3 DATALABEL= option

4 Y-axis LABEL= option

5 X-axis LABEL= option

6 Legend statement TITLE= option

7 ENTRYFOOTNOTE statement

8 Plot statement CURVELABEL= option

9 Plot statement LEGENDLABEL= option

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*"

Task	Statement	Option
label an axis	axis statement or layout axis option	LABEL="*string*"

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:

Option	Description
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 about text-related features for axes, legends, insets, and multi-cell layouts is available in other chapters:

- For managing the text in axes, see "Managing Axes in OVERLAY Layouts " on page 103.
- For managing the text in legends, see Chapter 20, "Adding Legends to Your Graph," on page 337.
- For managing the text in insets, see Chapter 21, "Adding Insets to Your Graph," on page 383.
- For managing the text in multi-cell layouts, see Chapter 15, "Creating Lattice Graphs Using the LATTICE Layout," on page 221 and Chapter 16, "Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts," on page 255.

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=

Statement Type	Option	Supporting Text Option
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 "Options That Override Attributes for Individual Plots" on page 506 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)
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)

For more information about *text-options*, see "Text Options" on page 670.

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. This text typically uses the largest font size among fonts in the graph.

Style Element	Default Use
GraphFootnoteText	Used for all footnotes. This text typically uses a smaller font size than the titles. Sometimes, footnotes are italicized.
GraphLabelText	Used for axis labels and legend titles. This text generally uses a smaller font size than titles.
GraphValueText	Used for axis tick values and legend entries. This text 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 annotation text that is added with the Annotate facility, the DRAWTEXT statement, and 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 the 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.

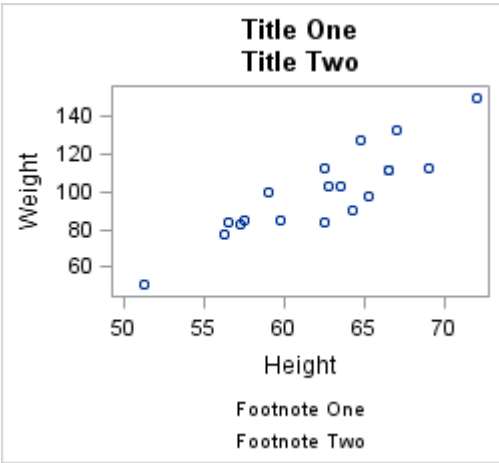
Text Statement Basics

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. You must place these statements inside the BEGINGRAPH block as children of the BEGINGRAPH block. They cannot be embedded in any other GTL statement block. The following example and output show the typical placement of these statements:

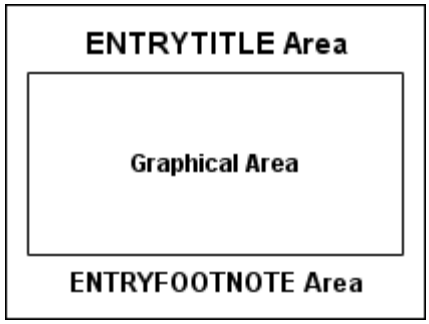
```
proc template;
  define statgraph textstatements;
    begingraph;
      entrytitle "Title One";
      entrytitle "Title Two";
      layout overlay;
        scatterplot x=height y=weight;
      endlayout;
      entryfootnote "Footnote One";
      entryfootnote "Footnote Two";
    endgraph;
  end;
run;

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



However, the statement placement in the following BEGINGRAPH block 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;
```

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.



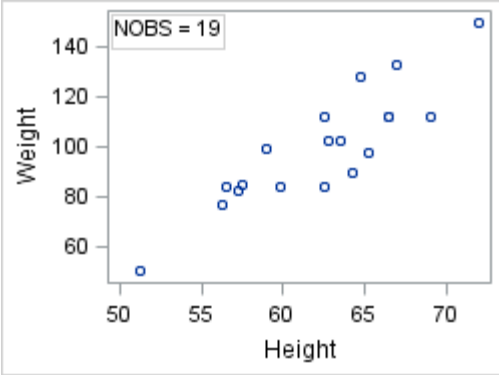
You can add as many titles and footnotes as you want. However, the space that is needed to accommodate the titles and footnotes always decreases the height of the graphical area. For graphs with extensive titles or footnotes, you should consider enlarging the graph size. For a discussion on sizing graphs, see “Controlling Graph Size” on page 581.

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:

```
proc template;
  define statgraph textentry;
    begingraph;
      layout overlay;
        scatterplot x=height y=weight;
        entry halign=left "NOBS = 19" /
          valign=top border=true;
      endlayout;
    endgraph;
 end;
run;

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

Here is the output.



You can use multiple ENTRY statements with GRIDDED layouts to create tables of text and complex insets. See Chapter 19, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 317.

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 that have 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";
```

Here is the output.

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";
```

Here is the output.

XYZ Corp. 02FEB12

Generating Text Items with Dynamic Variables, Macro Variables, and Expressions

Text items are not limited to string literals. Text items can also be defined as dynamic variables, 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 run time.

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

Here is the output.

XYZ Corp. 02FEB12

This next example shows how the GTL EVAL function causes an expression to be evaluated at run time. 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.));
```

Here is the output.

Summary for 02-02-12

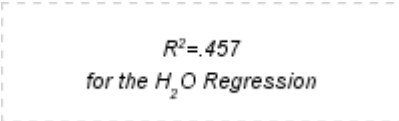
For more information about dynamic variables, macro variables, and EVAL expressions in GTL, see Chapter 30, “Using Dynamic Variables and Macro Variables in Your Templates,” on page 605 and Chapter 31, “Using Conditional Logic and Expressions in Your Templates,” on page 619.

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 dynamic variables or macro variables for the *string* portion of the text command.

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

Here is the output.



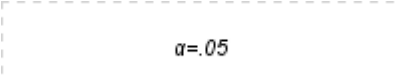
Another way to form text is to use the {UNICODE "*hex-value*"x } text command. For fonts that support Unicode code points, you can use the following syntax to render the glyph (character) corresponding to any Unicode value:

```
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 shortcuts to produce them have been added to GTL syntax. So another way of indicating `"03B1"x` is

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

Here is the output.



For a complete list of keywords that can be used with the `{unicode keyword}` notation, see “Reserved Keywords and Unicode Values” on page 647. For rules regarding specifying Unicode and other special characters, see “Rules for Unicode and Special Character Specifications” in *SAS Graph Template Language: Reference*.

In addition, 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;
```

Here is the output.

$\overline{X}$=6.78

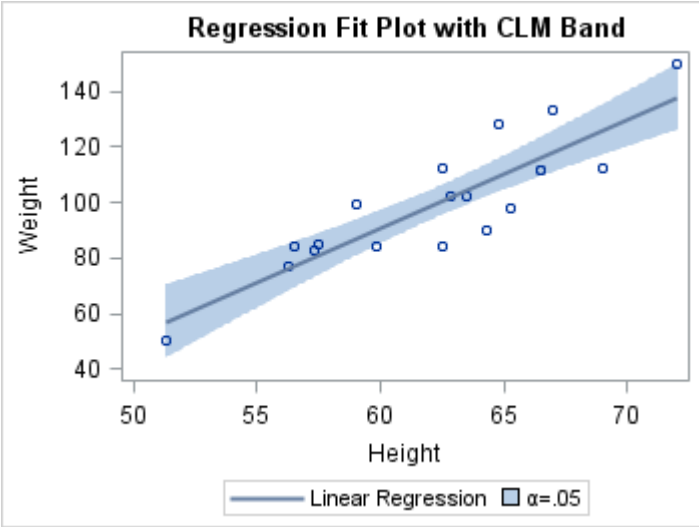
Using Unicode Values in Labels

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 use an ODS ESCAPECHAR= statement to define an escape character, and then include that escape character 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}. For rules regarding specifying Unicode characters, see "Rules for Unicode and Special Character Specifications" in *SAS Graph Template Language: Reference*.

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

```
ods escapechar="^"; /* Define an escape character */

proc template;
  define statgraph fit;
    begingraph;
      entrytitle "Regression Fit Plot with CLM Band";
      layout overlay;
        modelband "clm" / display=(fill)  name="band"
          legendlabel="^{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;
```

Here is the output.



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. For information about *line-options*, see "Line Options" on page 666.

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. For information about *text-options*, see "Text Options" on page 670. If a TEXTATTRS= prefix option is also used, it takes precedence over this statement option.

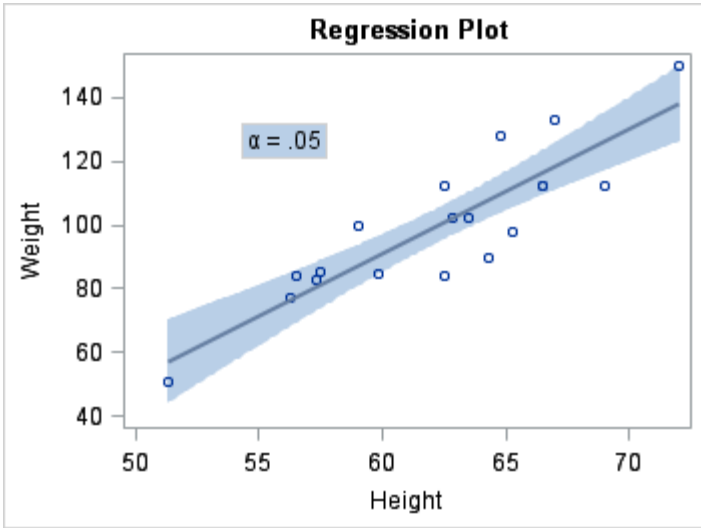
Setting Text Background, Borders, and Padding

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.

```
proc template;
define statgraph tmpl1;
  begingraph;
    entrytitle "Regression Plot";
    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;
end;

proc sgrender data=sashelp.class template=tmpl1;
run;
```

Here is the output.



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;
```

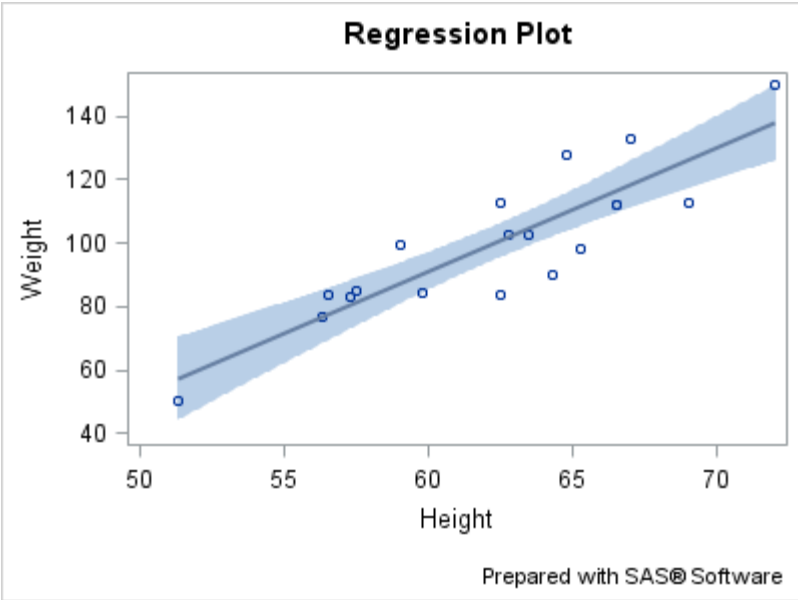
You can set the padding individually for the LEFT, RIGHT, TOP, and BOTTOM directions, or you can set the same padding in all directions as follows:

```
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. Here is an example.

```
proc template;
  define statgraph padding;
    begingraph;
      entrytitle "Regression Plot" / pad=(bottom=10px);
      entryfootnote halign=right
        "Prepared with SAS" {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;
  end;
run;

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

Here is the output.



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 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 that 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";
```

Here is the output.

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).

Here is the output when the fit policy is set to TRUNCATE.

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

The ellipsis in the output text indicates where the truncation occurs. Rather than truncating text, you can specify alternative "short" text to substitute whenever the primary text does not fit without wrapping in the available space.

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

Here is the output.

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.

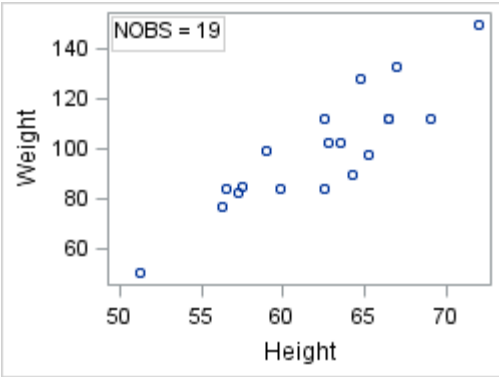
VALIGN= CENTER | TOP | BOTTOM
: Specifies the vertical alignment of the text.

Positioning ENTRY Text

By default, any ENTRY statement that is defined within a 2-D 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:

```
proc template;
  define statgraph textentry;
    begingraph;
      layout overlay / pad=0;
        scatterplot x=height y=weight;
        entry  halign=left "NOBS = 19" /
          valign=top border=true;
      endlayout;
    endgraph;
  end;
run;

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

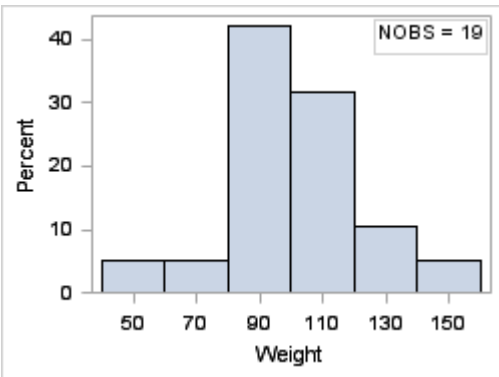
Here is the output.



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, the best location for an entry is either TOPLEFT or TOPRIGHT, depending on the skewness of the data.



With the following layout block, if the data were skewed to the right so that the entry text overlaps with the histogram, the text would automatically appear at TOPLEFT.

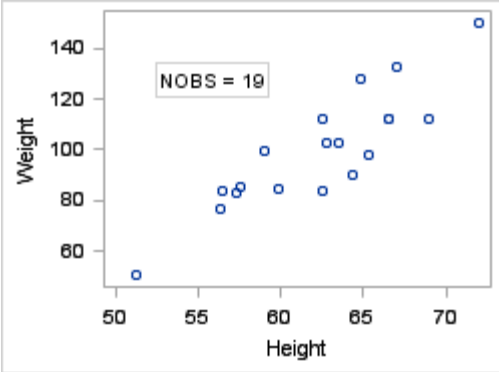
```
layout overlay;
  histogram weight;
  entry  "NOBS = 19" /
    autoalign=(topright topleft)
    border=true;
endlayout;
```

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 layout block 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;
```

Here is example output.

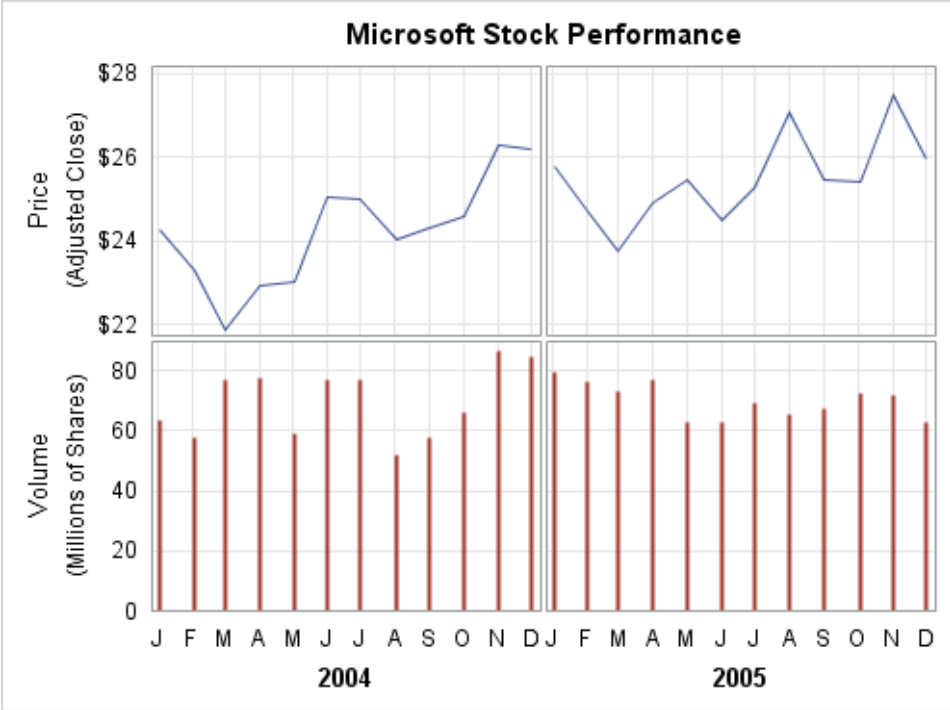


Rotating ENTRY Text

For the ENTRY statement, the ROTATE= option can be used to rotate the entry text. 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, ROWHEADERS, and COLUMNHEADERS blocks. In the following code fragment, the ROTATE= option in the ROWHEADERS block rotates the row headers 90 degrees in a LATTICE layout.

```
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;
```

Here is the result.



The complete code for this example is shown in "Example 4: Lattice with Row Headers" on page 249.

20

Adding Legends to Your 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

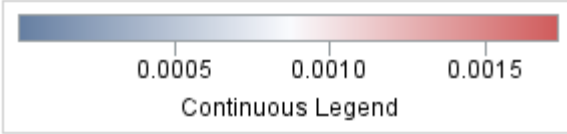
GTL supports two legend statements:

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 338.

  For details, see “General Legend Features” on page 344 and “Adding a Discrete Legend” on page 353.

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 338.

For details, see “General Legend Features” on page 344 and “Adding a Continuous Legend” on page 374.



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 that 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 does not show up in a legend.

Example Legend Coding for Common Situations

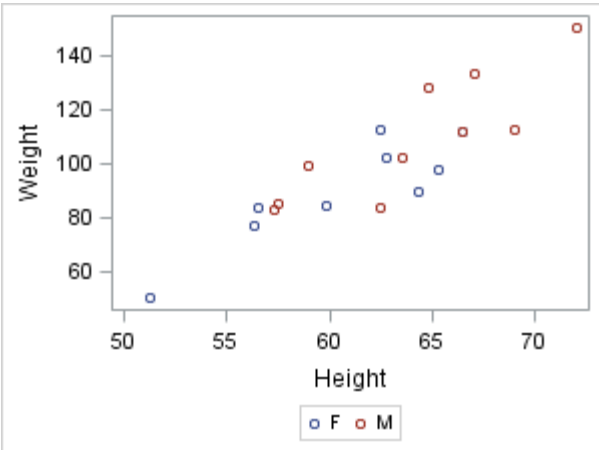
Show Group Values in a Legend

The appearance of the markers is automatically determined by the current style. The order of the legend entries is controlled by the data order as shown in the following example.

```
proc template;
  define statgraph order;
    begingraph;
      layout overlay;
```

```
        scatterplot x=height y=weight / group=sex name="scatter";
        discretelegend "scatter";
      endlayout;
    endgraph;
  end;
run;

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

Here is the output.

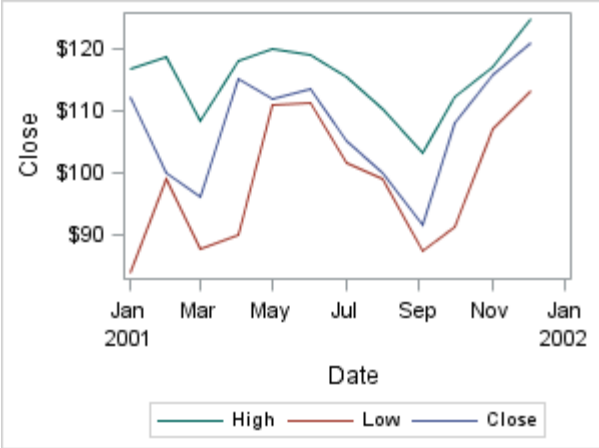


Identify Overlaid Plots in a Legend

This example illustrates that more than one plot can contribute to a legend.

```
proc template;
  define statgraph series;
    begingraph;
      layout overlay / cycleattrs=true;
        seriesplot X=date Y=close / name="sp1" ;
        seriesplot X=date Y=low / name="sp2" ;
        seriesplot X=date Y=high / name="sp3" ;
        discretelegend "sp3" "sp2" "sp1";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.stocks template=series;
  format high low close dollar.;
  where year(date)=2001 and stock="IBM";
run;
```

Here is the output.



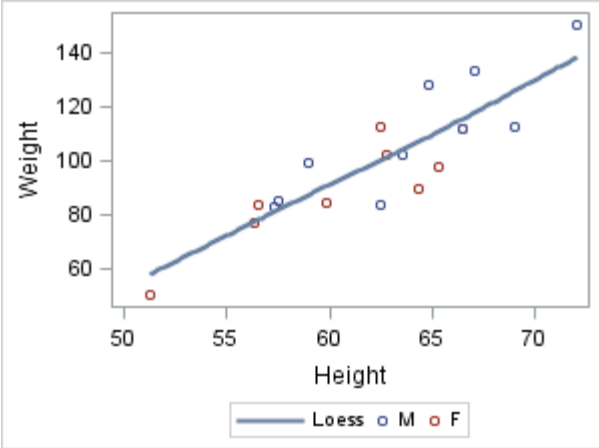
The CYCLEATTRS=TRUE option assigns different visual attributes to each plot. For more information about the CYCLEATTRS= option, see "Ordering the Legend Entries for Non-grouped Plots" on page 357. The order of the names in the DISCRETELEGEND statement determines the order of the legend entries. No label is assigned to the High, Low, and Close variables. In this case, by default, the legend entry text is determined by the response variable name. You can use the LEGENDLABEL= option in each SERIESPLOT statement to override the default legend text for each plot.

Show Group Values and Identify Plots in a Legend

A legend can show group values for multiple groups and identify one or more plots as shown in the following example.

```
proc template;
  define statgraph compoundLegend;
   begingraph;
    layout overlay;
       scatterplot x=height y=weight / group=sex name="scatter";
       loessplot x=height y=weight / name="Loess";
       discretelegend "Loess" "scatter";
    endlayout;
    endgraph;
  end;
run;
proc sgrender data=sashelp.class template=compoundlegend;
run;
```

Here is the output.



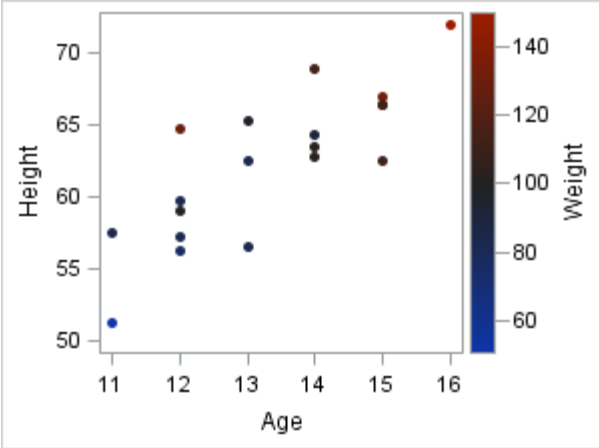
For a LOESSPLOT statement, the default legend text is the text specified by the plot's NAME= option, which is "Loess" in this example. You can use the LEGENDLABEL="*string*" option in the LOESSPLOT statement to override the NAME= text in the legend.

Show a Legend for a Continuous Response Variable (Scatter Plot)

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

```
proc template;
  define statgraph order;
    begingraph;
      layout overlay;
        scatterplot x=age y=height / name="sc"
          markercolorgradient=weight
          markerattrs=(symbol=circlefilled);
        continuouslegend "sc" / title="Weight";
      endlayout;
    endgraph;
  end;
run;

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

Here is the output.

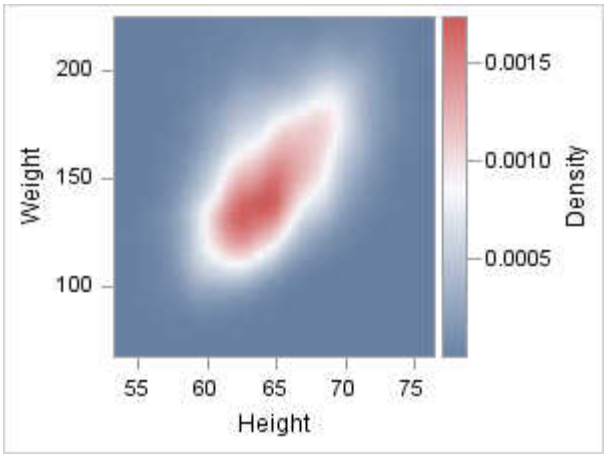


Show a Legend for a Continuous Response Variable (contour plot)

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

```
proc template;
  define statgraph mygraphs.contour;
    begingraph;
      layout overlay;
        contourplotparm x=height y=weight z=density /
          contourtype=gradient name="con";
        continuouslegend "con" / title="Density";
      endlayout;
    endgraph;
  end;
run;

ods graphics / antialiasmax=3600;
proc sgrender data=sashelp.gridded template=mygraphs.contour;
  where height>=53 and weight<=225;
run;
```

Here is the output.



General Legend Features

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

Positioning Options

Overview of the Legend Placement Options

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 14, “Creating Gridded Graphs Using the GRIDDED Layout ,” on page 207 (GRIDDED)
- Chapter 15, “Creating Lattice Graphs Using the LATTICE Layout,” on page 221 (LATTICE)
- Chapter 16, “Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts,” on page 255 (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.

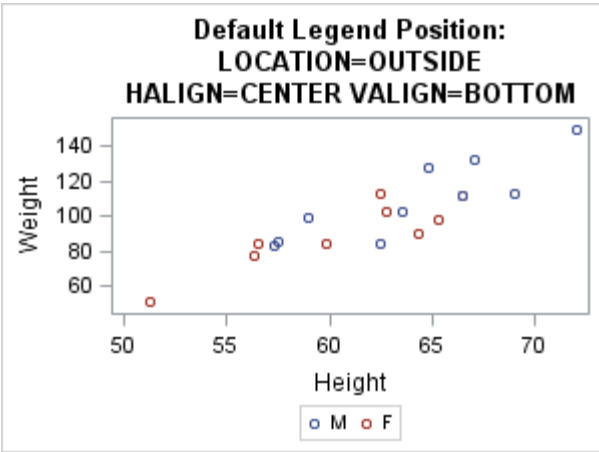
VALIGN = TOP | CENTER | BOTTOM
: determines vertical alignment. The default is BOTTOM.

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 as shown in the following example.

```
proc template;
  define statgraph location1;
    begingraph;
      entrytitle "Default Legend Position:";
      entrytitle "LOCATION=OUTSIDE";
      entrytitle "HALIGN=CENTER VALIGN=BOTTOM";
      layout overlay;
        scatterplot X=Height Y=Weight / name="sp" group=sex;
        discretelegend "sp" /
          location=outside halign=center valign=bottom;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.class template=location1;
run;
```

Here is the output.



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

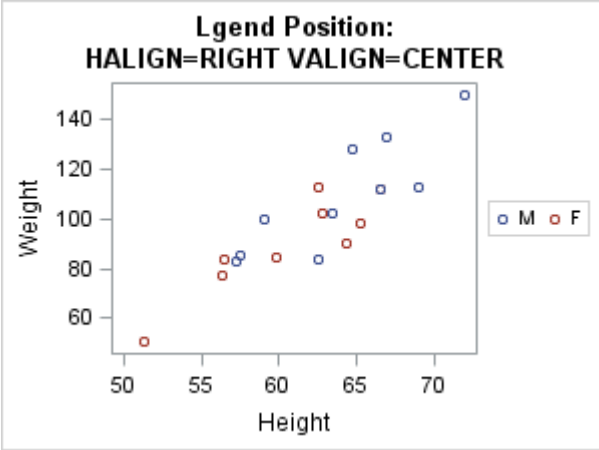
HALIGN=CENTER and VALIGN=CENTER. To accommodate the legend, the size of the plot wall is adjusted so that the legend(s) can be displayed.

Note: Sometimes with large legends, this size adjustment causes problems. Sizing issues are discussed in "Arranging Legend Entries into Columns and Rows" on page 359 and "When Discrete Legends Get Too Large" on page 371.

The following layout block 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;
```

Here is example output.

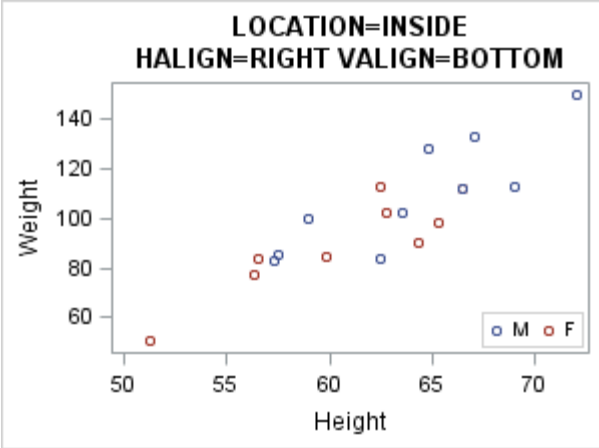


Displaying Legends inside the Plot Wall

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 layout block 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;
```

Here is example output.



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).

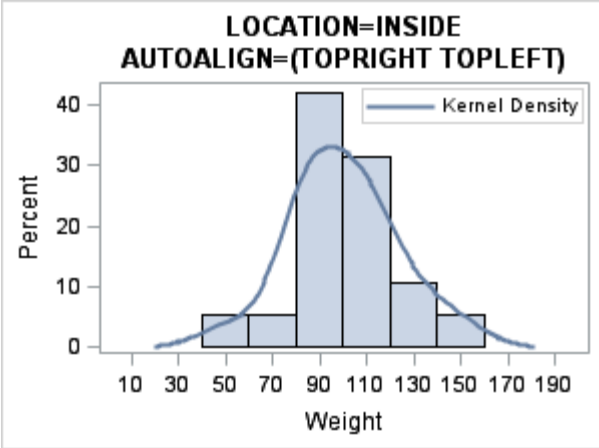
Automatically Aligning an Inside Legend

When the plot statements are specified in a 2-D 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 layout block, the best position for an inside legend is TOPRIGHT or TOPLEFT.

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

Here is example output.



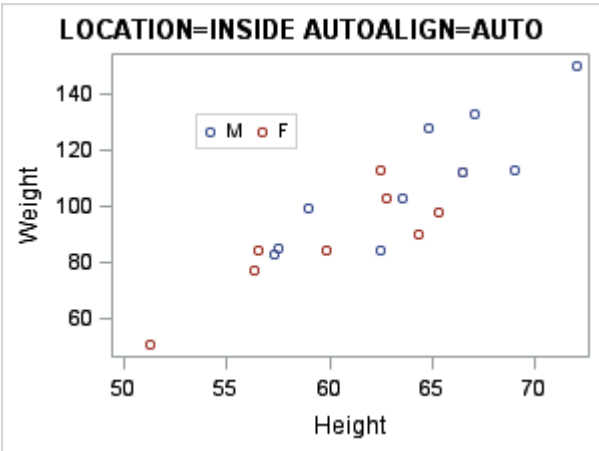
Because the AUTOALIGN= option specifies a list of preferred positions, the first of the listed positions that does not involve data collision is used. If the histogram had been skewed to the right, the TOPLEFT position would be used.

When the parent layout contains only scatter plots, you can fully automate the selection of an internal position by specifying AUTOALIGN=AUTO as shown in the following layout block.

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

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= can be used.

Here is example output.



General Appearance Options

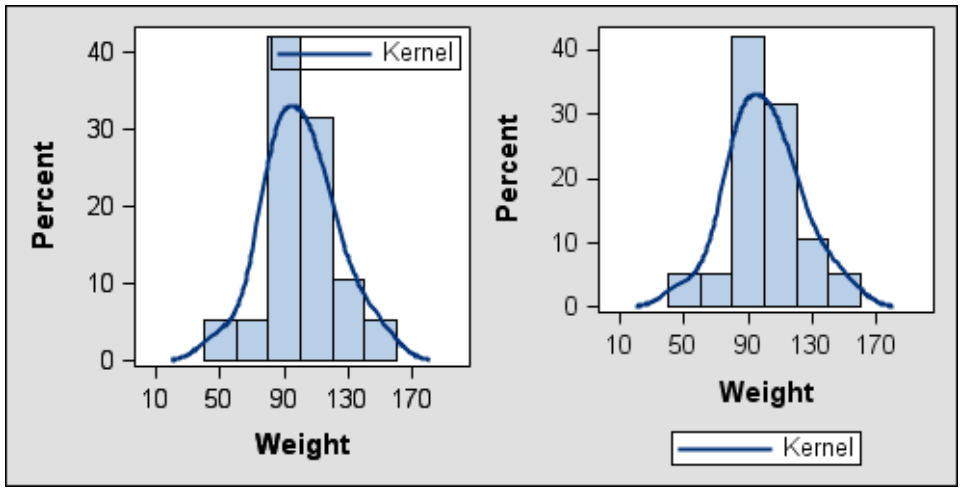
Using Background Transparency and Color

The following options control the appearance of the legend background:

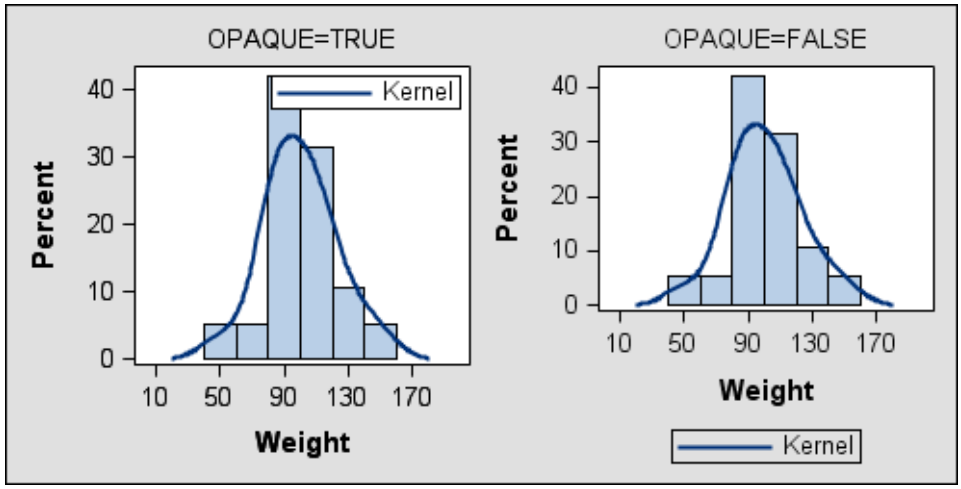
OPAQUE = TRUE | FALSE
: determines whether the legend background is 100% transparent or 0% transparent.

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.

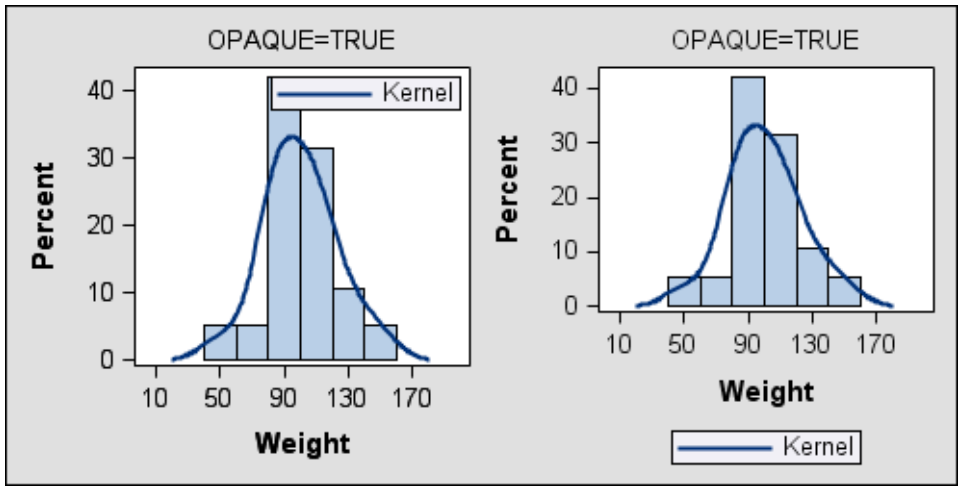
By default, OPAQUE=FALSE when LOCATION=INSIDE. This minimizes the potential for the legend to obscure the markers, lines, fills, and labels in the plot area. when When LOCATION=OUTSIDE, OPAQUE=TRUE by default. This enables the legend background color to appear. Typically, the default legend background 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 that you can 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.



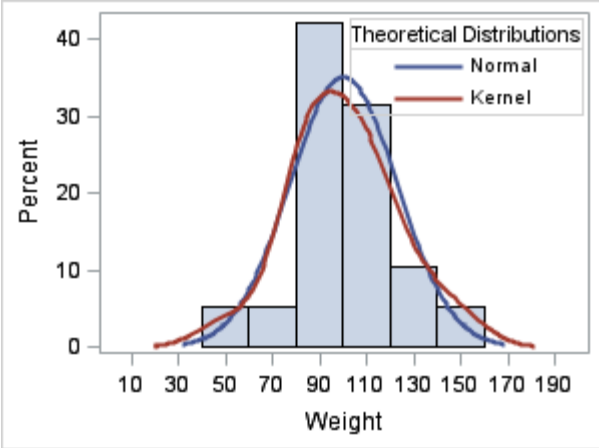
Using a Legend Title and Title Border

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. This is demonstrated in the following layout block.

```
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;
```

Here is example output.

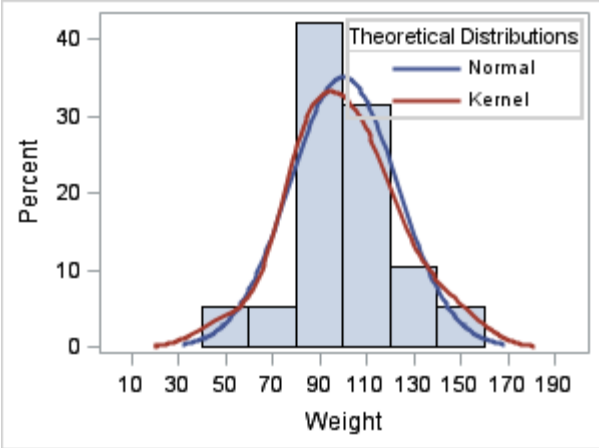


Legend Border

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 layout block 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;
```

Here is example output.



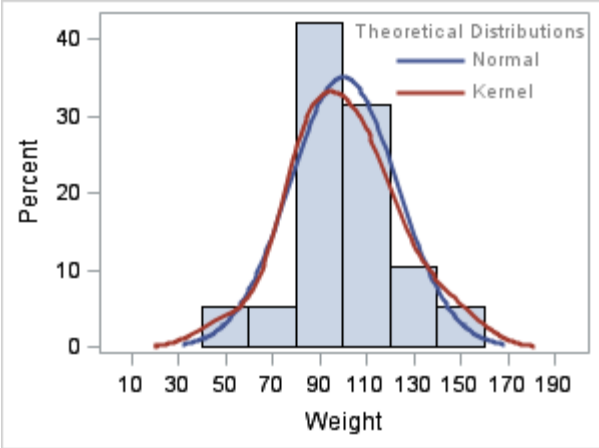
Legend Text Properties

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 layout block 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;
```

Here is example output.



Adding a Discrete Legend

Placing the Legend

You can use the AUTOALIGN, LOCATION=, HALIGN=, and VALIGN= options to place your legend. Note the following about legend placement:

- When a discrete legend is placed within the axis frame (LOCATION=INSIDE):
  - It is always placed on top of plot lines and markers.
  - Its background is fully transparent by default (OPAQUE=FALSE), meaning that underlying lines, markers, and data labels will show through the legend.
  - Its position is controlled with the AUTOALIGN= option.
- When a discrete legend is placed outside the axis frame (LOCATION=OUTSIDE):
  - Its background is fully opaque by default (OPAQUE=TRUE).
  - Its position is controlled with the HALIGN= and VALIGN= options.
- When a discrete legend is placed within nested layouts, you might have to use the ACROSS= and ORDER=ROWMAJOR options or the DOWN= and ORDER=COLUMNMAJOR options to obtain the desired legend organization.

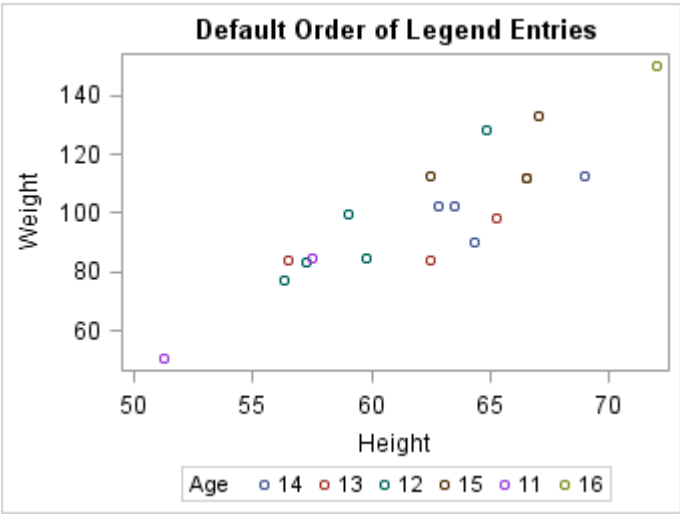
Ordering the Legend Entries for a Grouped Plot

Overview of the Group Value Default Order

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

```
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;
```

Here is the output.



Sorting the Legend Items

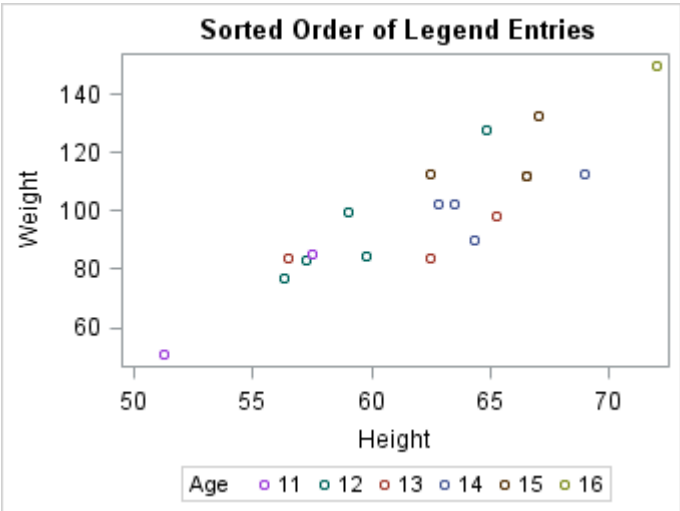
You use the SORTORDER= option in your DISCRETELEGEND statement to perform a linguistic sort on the legend items in ascending or descending order. By default, SORTORDER=AUTO, which displays the items in the order in which they are provided by the contributing plots.

Note: The SORTORDER= option overrides the ordering that is established by the GROUPORDER= option in the legend's constituent plot statements. The SORTORDER=ASCENDINGFORMATTED or SORTORDER=DESCENDINGFORMATTED options combine the entries from the contributing plots and orders them as a single list.

The following layout block sorts the legend items in ascending order.

```
layout overlay;
   scatterplot x=height y=weight / name="sp" group=age;
   discretelegend "sp" / title="Age" sortorder=ascendingformatted;
endlayout;
```

Here is example output.



Alternatively, you can create a sorted data set, and then use the sorted data to generate your graph. In that case, the legend values appear in the sorted order.

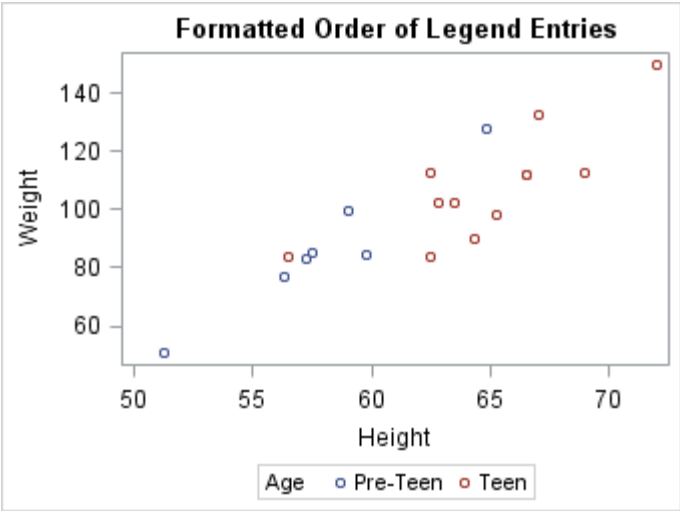
Formatting the Data

You can apply a format to a group variable 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." Because there are no adults, "Adult" does not appear in the graph. If the data were sorted in descending age order the legend entry order would be reversed. The template that

was created in "Overview of the Group Value Default Order" on page 354 is used to generate the graph.

```
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;
```

Here is the output.



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 "Using the INDEX= Option to Achieve Data-Independent Appearance for Grouped Plots" on page 535.

Ordering the Legend Entries for Non-grouped Plots

Ordering Entries from Overlaid Plots

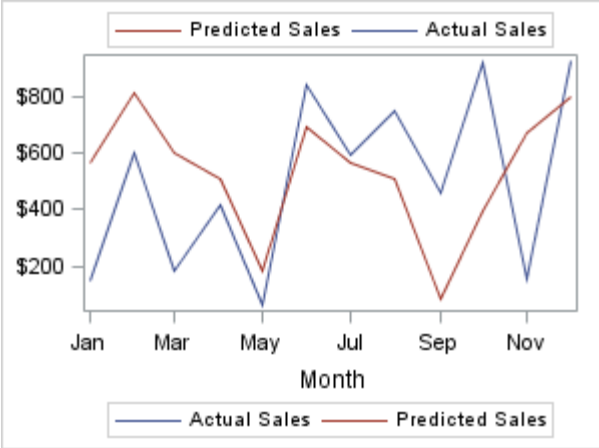
When plots are overlaid and you want to distinguish them in a legend, you must assign each plot a name and then reference the name in the legend statement. The order in which the plot names appear on the legend statement controls the ordering of the legend entries for the plots.

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`.

```
proc template;
  define statgraph series;
    begingraph;
      layout overlay / xaxisopts=(timeopts=(tickvalueformat=data))
        yaxisopts=(display=(ticks tickvalues)) cycleattrs=true;
        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;
    endgraph;
  end;
run;

proc sgrender data=sashelp.prdsale template=series;
  format month Monname3. actual predict dollar.;
  where year=1994 and product="DESK" and division="EDUCATION"
    and country="CANADA" and region="WEST";
run;
```

Here is the output.

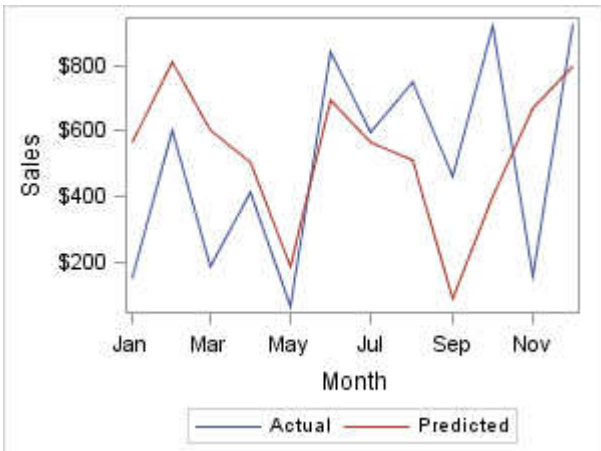


Assigning Legend Entry Labels

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 as shown in the following layout block.

```
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;
```

Here is example output.



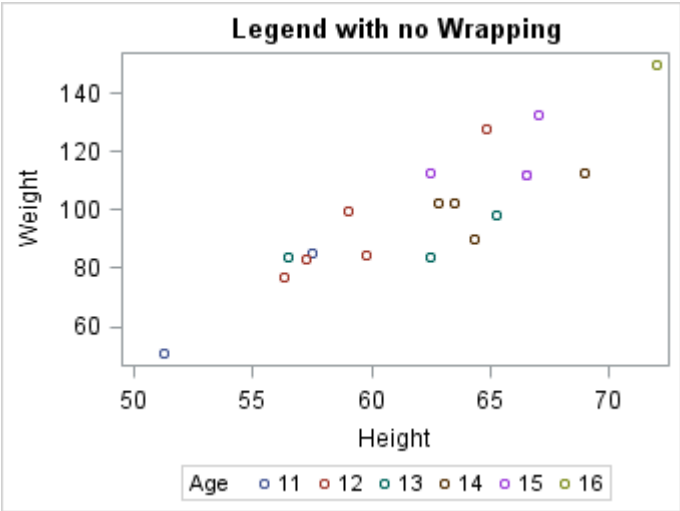
Note: Other techniques are available for labeling plots without using a legend. Many plots that render one or more lines support a CURVELABEL= option that places text inside or outside of the plot wall to label the line(s). These plots include

SERIESPLOT, STEPPLOT, DENSITYPLOT, REGRESSIONPLOT, LOESSPLOT, PBSPLINEPLOT, MODELBAND, BANDPLOT, LINEPARM, REFERENCELINE, and DROPLINE. Additional options are available to control curve label location, position, and text properties. For examples, see Chapter 27, “Managing Your Graph’s Appearance,” on page 491 and Chapter 19, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 317.

Arranging Legend Entries into Columns and Rows

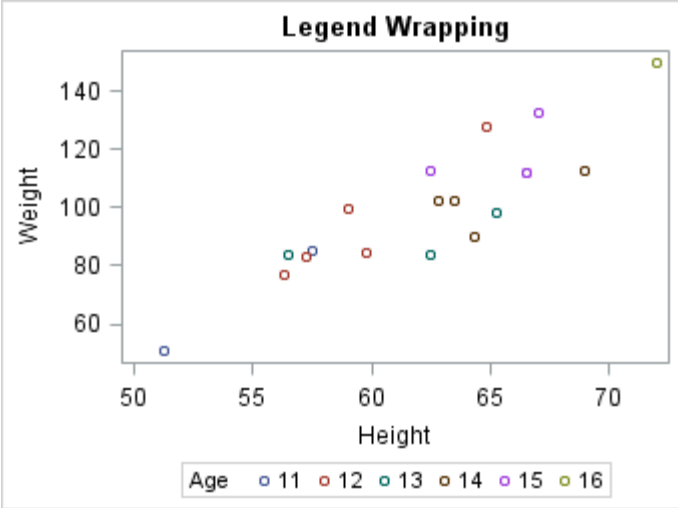
Default Legend Item Arrangement

The arrangement of the entries in a legend is affected by the number of entries in the legend, the length of the entry labels, and the size of the graph. When the graph is wide enough, all legend information can fit into one row as shown in the following figure.



Note: When all the legend entries and the legend title 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.

When the legend entries and the legend title cannot fit into one row, the legend title and entries are wrapped into multiple rows in order to fit the allotted space. 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. Used only 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.

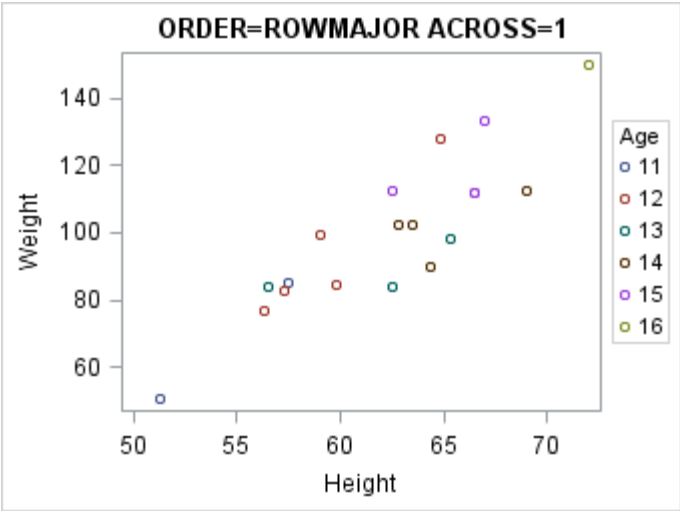
Organizing Legend Entries in a Fixed Number of Columns

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."

```
proc template;
  define statgraph order;
    begingraph;
      entrytitle "ORDER=ROWMAJOR ACROSS=1";
```

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

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

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

Here is the output.



As you increase the number of columns, the plot area decreases. In the following layout block, 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;
```

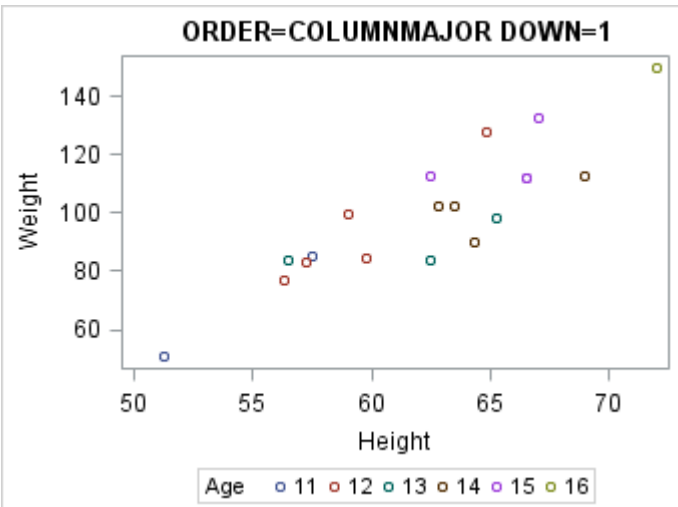
Here is example output.

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 layout block, 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;
```

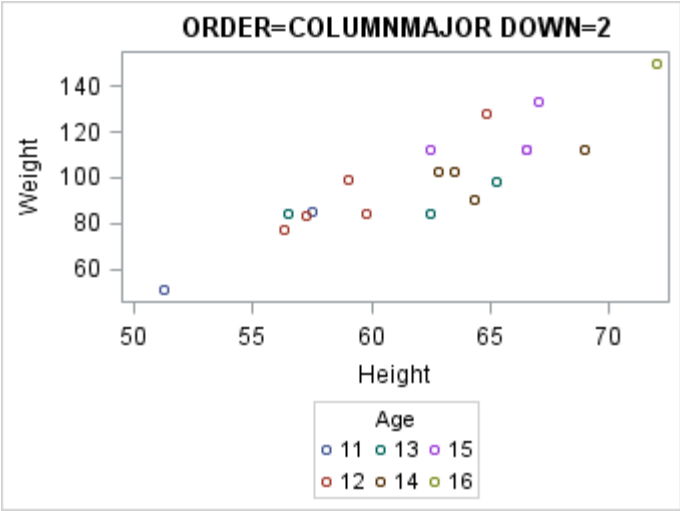
Here is example output.



As you increase the number of rows, the plot area decreases. In the following layout block, 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;
```

Here is example output.



Controlling the Label and Item Size

To control the size of the labels in your legend, include the SIZE= option in the VALUEATTRS= option list in the DISCRETELEGEND statement. By default, the size of the items in the legend, such as markers, filled squares, and filled bubbles, remain fixed regardless of the label font size. When you increase the label font size, the labels can appear out of proportion with the items. You can include the AUTOITEMSIZE=TRUE option in your DISCRETELEGEND statement to automatically size markers, filled squares, and filled bubbles in proportion to the label font size. The AUTOITEMSIZE= option does not affect line items. When AUTOITEMSIZE=TRUE, if you change the label font size, any markers, filled squares, and filled bubbles in the legend are automatically resized to maintain proportion with the resized labels. The following layout block specifies a discrete legend that has 12-point labels and markers that are sized proportionately to the labels.

```
layout overlay;
   scatterplot x=Height y=Weight / name="sp2" group=age;
   discretelegend "sp2" / title="Age"
     valueattrs=(size=12pt) autoitemsize=true;
endlayout;
```

Adding Items to a Discrete Legend

Adding Items with the LEGENDITEM Statement

You can use the LEGENDITEM statement to manually add items to your legend that do not appear in your plot. This enables you to provide additional information about your plot or to build a common legend that you can use with multiple plots. The syntax of the LEGENDITEM statement is as follows:

LEGENDITEM TYPE=*EntryType* NAME="*LegendName*" </ *options*>

You must place the LEGENDITEM statement in the global definition area of the template between the BEGINGRAPH statement and the first layout statement. Do not embed the LEGENDITEM statement in any other GTL block. It must be a child of the BEGINGRAPH statement. The TYPE=*EntryType* option specifies the type of entry that you want to add to your legend. *EntryType* can be one of the following:

LINE
MARKER
MARKERLINE
TEXT

The NAME= option specifies a name by which this statement can be referenced in a DISCRETELEGEND statement. Use options to set the desired appearance for the entry, such as color, pattern, font, and so on. Note the following about the LEGENDITEM statement:

- You can specify any supported set of attributes for a legend item regardless of its type. However, sets of attributes that are not applicable to the legend item type are ignored. For example, in the LEGENDITEM statement, if TYPE=FILL and the MARKERATTRS=( ) option is specified, the MARKERATTRS=( ) option is ignored.
- When TYPE=TEXT, if the TEXT= option is not specified in the LEGENDITEM statement, a blank space is used by default.

For more information about the LEGENDITEM statement, see "LEGENDITEM" in *SAS Graph Template Language: Reference*.

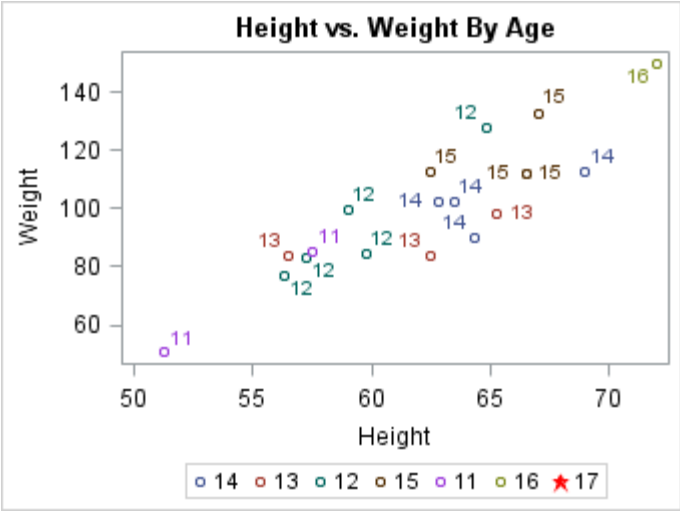
Here is an example of a LEGENDITEM statement that adds item 17 to the legend of a height-to-weight chart that is grouped by age. The new item uses a red-filled star as a marker.

```
proc template;
  define statgraph additem;
  dynamic legenditem;
    BeginGraph;
      entrytitle "Height vs. Weight By Age";
      legenditem  type=marker name="newitem" / label="17"
        lineattrs=(color=red)
        markerattrs=(symbol=starfilled color=red);
```

```
        layout overlay;
          scatterplot x=height y=weight / group=age
            name="heightweight";
          discretelegend "heightweight" "newitem";
        endlayout;
      EndGraph;
    end;
  run;

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

Notice that the DISCRETELEGEND statement specifies the name specified in the NAME= option in each of the SCATTERPLOT and LEGENDITEM statements. This combines the new legend item and the automatically generated plot legend into one legend.

Here is the output.



The new item 17 and its red-filled star marker are appended to the existing items in the legend even though 17 is not in the data. You can include multiple LEGENDITEM statements to add multiple items to your legend.

Adding Text Items with the LEGENDTEXTITEMS Statement

Starting with SAS 9.4M3, you can use the LEGENDTEXTITEMS statement to add to legend text items that are stored in the plot data. As with the LEGENDITEM statement, you can use the LEGENDTEXTITEMS statement to create a custom legend for your graphs. However, with the LEGENDTEXTITEMS statement, the items are data-driven as opposed to being hard coded in your graph template.

The syntax of the LEGENDTEXTITEMS statement is as follows:

LEGENDTEXTITEMS NAME="*LegendName*" TEXT=*column* </ *options*>

The NAME= option specifies a name by which this statement is referenced in a DISCRETELEGEND statement. The TEXT= option specifies the column in the plot

data that contains the text items. You can use the LABEL= option to specify a column that contains a label for each item. Other options are available that enable you to control the appearance of the text and labels.

The following restrictions apply to the LEGENDTEXTITEMS statement:

- You must place the LEGENDTEXTITEMS statement in the global definition area of the template between the BEGINGRAPH statement and the first layout statement. Do not embed the LEGENDTEXTITEMS statement in any other GTL block. It must be a child of the BEGINGRAPH statement.
- One item is added for each observation.
- The column should not contain any missing values.
- Grouping is not supported.

For more information about the LEGENDTEXTITEMS statement, including an example, see "LEGENDTEXTITEMS" in *SAS Graph Template Language: Reference*.

Removing Items from a Discrete Legend

Removing Items with the EXCLUDE= Option

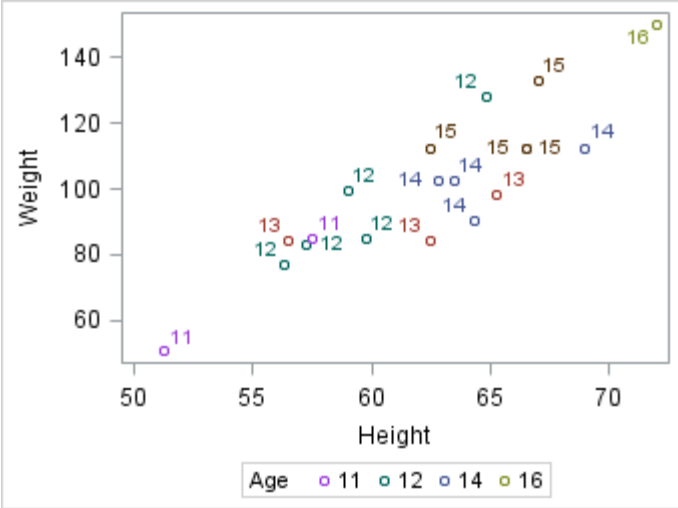
To remove one or more items from your legend, use the EXCLUDE= option on your DISCRETELEGEND statement. The EXCLUDE= specifies the label of each item that is to be removed as follows:

```
EXCLUDE=("item1Label"< "item2Label" ...>)
```

Each item label is enclosed in quotation marks and must match the formatted string of the data value. For two or more items, each label is separated by a space. String matching is case sensitive. Here is an example that removes age groups 13 and 15 from the legend.

```
proc template;
  define statgraph order;
    begingraph;
      layout overlay;
        scatterplot X=Height Y=Weight / name="sp" group=age
          datalabel=age;
        discretelegend "sp" / title="Age"
          sortorder=ascendingformatted
          exclude=("13" "15");
      endlayout;
    endgraph;
  end;
run;

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

Here is the output.



Groups 13 and 15 are removed from the legend even though they exist in the data.

Filtering Items from Multiple Plots with the TYPE= Option

When multiple plot statements contribute to the legend, you can use the TYPE= option in your DISCRETELEGEND statement to include only items of a specific type. You can specify one of the following item types:

ALL	LINE	MARKER
FILL	LINECOLOR	MARKERCOLOR
FILLCOLOR	LINEPATTERN	MARKERSYMBOL

Note: If no entries match the type specified by the TYPE= option, a legend is not drawn in the plot.

Here is an example that includes only marker entries in a legend for a series plot that displays both the lines and markers.

```
proc template;
define statgraph plots;
begingraph;
  entrytitle "Closing Price and Volume for 2002";
  Layout overlay;
    seriesplot x=date y=close / group=stock name="plot1" display=all;
    discretelegend  "plot1" / title="Stock" type=marker;
  endLayout;
endgraph;
end;
run;

proc sgrender data=sashelp.stocks template=plots;
  where date between '01JAN2002'd AND '31DEC2002'd;
run;
```

Merging Legend Items from Two Plots into One Legend

You can use the MERGEDLEGEND statement to merge the legend items from two grouped plots into one legend. The basic syntax is as follows:

```
MERGEDLEGEND "graph1" "graph2" / options;
```

The options used with the MERGEDLEGEND statement are similar to those that are used with the DISCRETELEGEND statement. The following restrictions apply to the MERGEDLEGEND statement:

- You must provide exactly two plot names.
- Each plot name must be enclosed in quotation marks.
- Only grouped plots are supported.
- Only plots with line and marker overlays are supported.

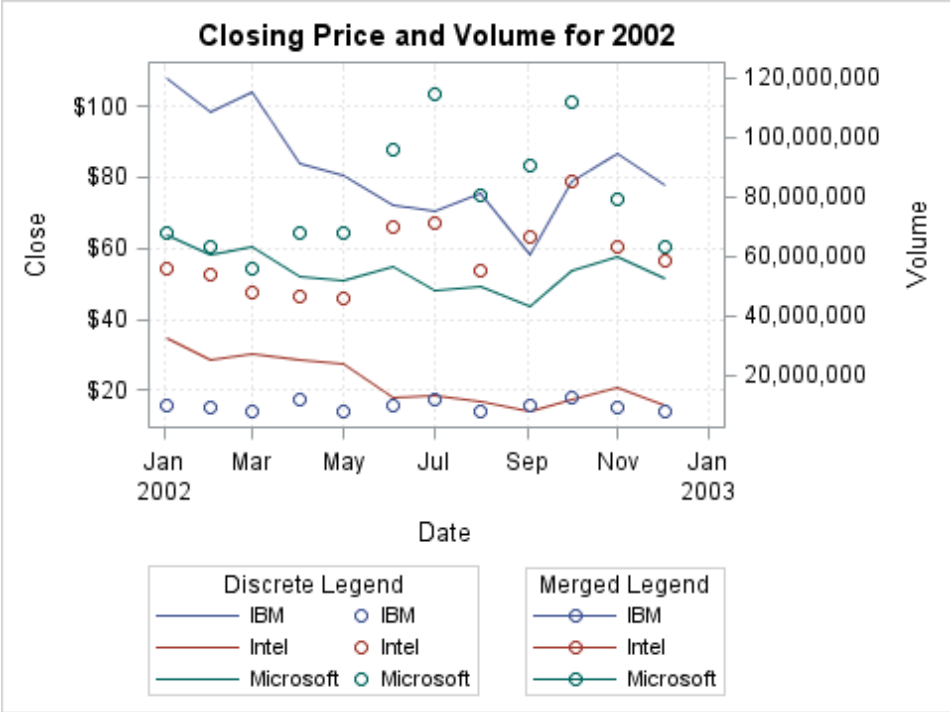
During the merge process, the group values from both plots are compared. The legend symbols for duplicate group values are combined into one legend item, which is then combined with the unique items into one legend. If the items cannot be merged, the following warning message appears in the SAS log:

```
WARNING: MERGEDLEGEND statement does not reference two plots whose legend
items can be properly merged. The legend will not be displayed.
```

If you receive this message, verify that both of the contributing plots are grouped plots that use line and marker overlays. Here is an example that overlays a series plot and a scatter plot, and combines the plot symbols into one merged legend and one discrete legend that are displayed side-by-side for comparison.

```
proc template;
define statgraph plots;
begingraph;
  entrytitle "Closing Price and Volume for 2002";
  Layout overlay /
     xaxisopts=(griddisplay=on gridattrs=(color=lightgray pattern=dot))
     yaxisopts=(griddisplay=on gridattrs=(color=lightgray
pattern=dot));
     seriesplot x=date y=close / group=stock name="plot1";
     scatterplot x=date y=volume / group=stock name="plot2" yaxis=y2;
     discretelegend  "plot1" "plot2" / title="Discrete Legend"
       across=2 down=3 valign=bottom order=columnmajor halign=left;
     mergedlegend "plot1" "plot2" / title="Merged Legend"
       sortorder=ascendingformatted across=1 valign=bottom
halign=right;
  endLayout;
endgraph;
end;
run;

proc sgrender data=sashelp.stocks template=plots;
  where date between '01JAN2002'd AND '31DEC2002'd;
run;
```

Here is the output.



In this example, the SERIESPLOT statement creates a series plot of the closing stock prices grouped by stock. The SCATTERPLOT statement creates a scatter plot of the trading volume grouped by stock. Notice that both plots are grouped, and exactly two plot names are used in the MERGEDLEGEND statement. As shown in this example, the discrete legend consists of two entries for each stock, one from each of the plots. Because SORTORDER= defaults to AUTO in this case, the items from each plot appear in the order in which they occur in the data. In contrast, the merged legend consists of only one entry for each stock, which is a combination (overlay) of the entries from both plots. You can use the SORTORDER= option in the MERGEDLEGEND statement to sort the items in the merged legend. In this case, because the data is already sorted, SORTORDER=ASCENDINGFORMATTED and SORTORDER=AUTO have the same effect.

Creating a Global Legend

When multiple discrete legends are used, you can use a LAYOUT GLOBALLEGEND block to combine all of the discrete and merged legends into one global legend. The following restrictions apply to global legends:

- Only one LAYOUT GLOBALLEGEND block is allowed for each template.
- You must include the LAYOUT GLOBALLEGEND block in the BEGINGRAPH/ENDGRAPH block.
- Any DISCRETELEGEND or MERGEDLEGEND statements that appear outside of the LAYOUT GLOBALLEGEND block are ignored.
- The individual legends in the global legend can be arranged in a single row or a single column only.

- CONTINUOUSLEGEND statements are not supported.

To combine your legends into one global legend, include all of the DISCRETELEGEND and MERGEDLEGEND statements in your LAYOUT GLOBALLEGEND block. The resulting global legend is placed at the bottom of the graph just above the footnotes. You can use the TITLE= option to add a title for the global legend. You can also use the TITLE= option on each of the DISCRETELEGEND or MERGEDLEGEND statements to add titles for the individual legends. Use the LEGENDTITLEPOSITION = option to specify the position of the individual legend titles.

The TYPE= option specifies whether the legends are arranged in a column or a row. When you specify TYPE=ROW, you can use the WEIGHTS= option to specify the amount of space that is available for each of the legends. The WEIGHTS= option can be one of the following values:

UNIFORM
specifies that all of the nested legends are given an equal amount of space (default).

PREFERRED
specifies that each nested legend is to be given its preferred amount of space.

weight-list
specifies a space-separated list of PREFERRED or *number* keywords where each keyword corresponds to a nested legend.

PREFERRED
indicates that the corresponding legend is to get its preferred size.

number
specifies a proportional weight for the corresponding legend, which determines the percentage of the available space that the legend gets. The total of the values does not need to be 1. When PREFERRED and *number* keywords are used together, the PREFERRED legends are given their preferred space. The remaining space is divided among the *number* legends based on their weighted values.

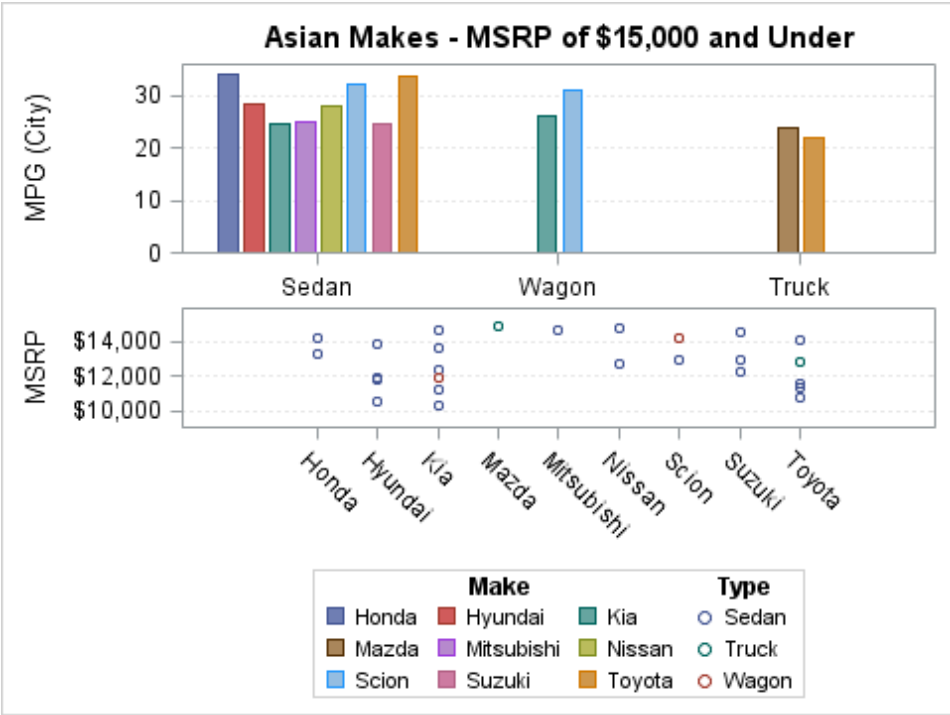
For more information, see *SAS Graph Template Language: Reference*.

Here is an example that creates a global legend for two plots.

```
proc template;
define statgraph foo;
begingraph;
   layout lattice;
      entrytitle "Asian Makes - MSRP  Under $15,000";
      Layout overlay / xaxisopts=(display=(ticks tickvalues line))
        yaxisopts=(griddisplay=on gridattrs=(color=lightgray
          pattern=dot));
        barchart category=type response=mpg_city / group=make
name="bar"
          stat=mean groupdisplay=cluster barwidth=0.75;
      endLayout;
      Layout overlay / xaxisopts=(display=(ticks tickvalues line))
        yaxisopts=(griddisplay=on gridattrs=(color=lightgray
pattern=dot));
        scatterplot x=make y=msrp / group=type name="scatter";
      endLayout;
   endlayout;
   layout globallegend / type=row weights=preferred
```

```
    legendtitleposition=top;
    discretelegend "bar" / across=3
      title="Make" titleattrs=(weight=bold);
    discretelegend "scatter" / sortorder=ascendingformatted
      title="Type" titleattrs=(weight=bold);
  endlayout;
endgraph;
end;
run;

proc sgrender data=sashelp.cars template=foo;
  where origin="Asia" && msrp <= 15000;
run;
```

Here is the output.



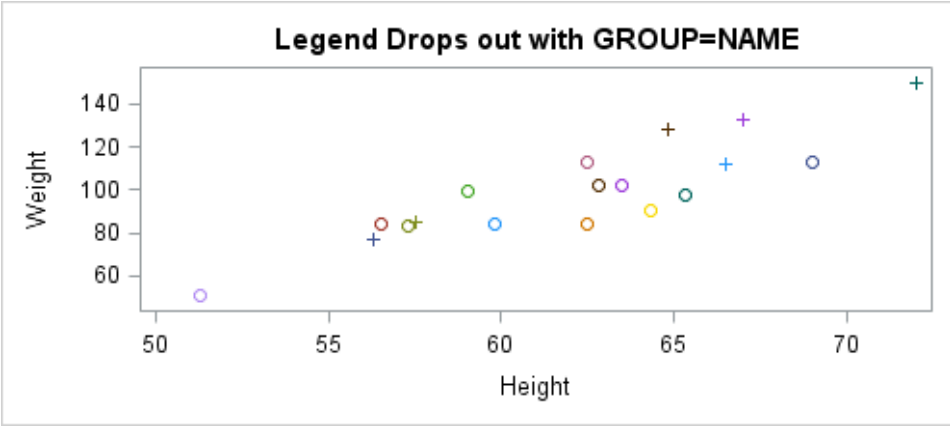
When Discrete Legends Get Too Large

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 / designwidth=495px designheight=220px;
```

```
      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;
```

Here is the output.



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;
```

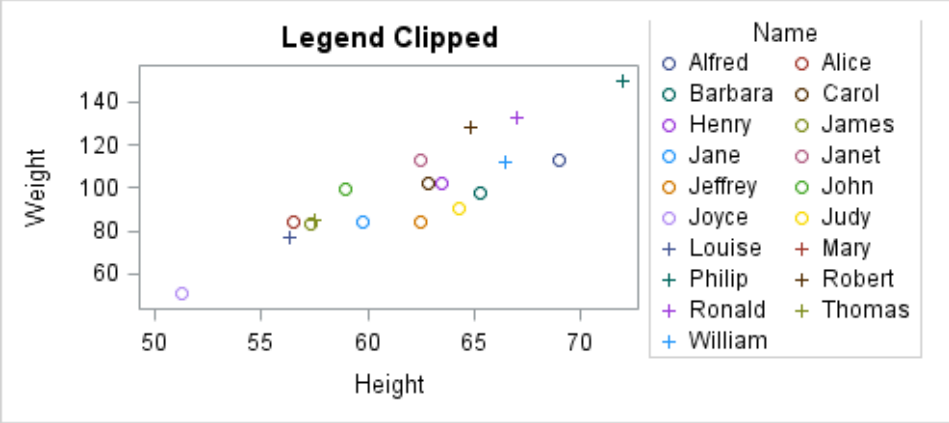
However, changing the total area that is allotted to legends might not resolve the problem if the specified legend organization does not fit in the existing size. In these cases, the legend might not be displayed and you would see the following log message:

```
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 as shown in the following DISCRETELEGEND statement.

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

This forces the legend to display so that you can visually inspect it. Here is example output.



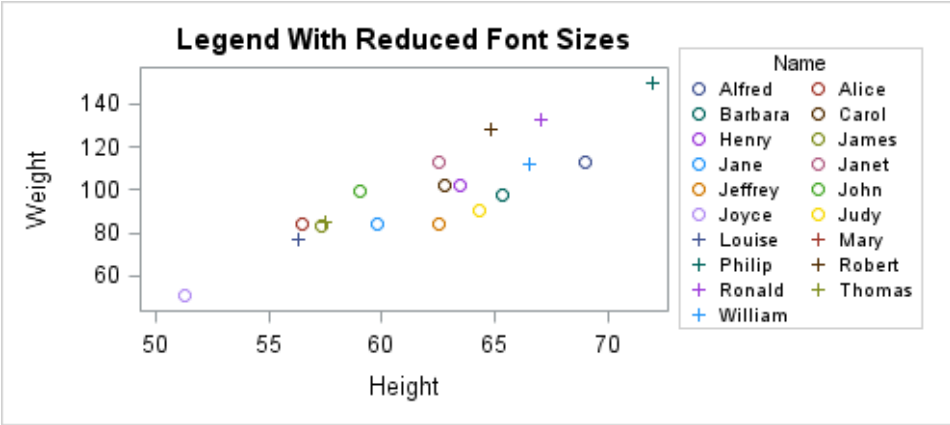
It is apparent that the height chosen for the output is not large enough to display the title and all legend entries in two columns. The problem can be fixed in any of the following ways:

- increasing the graph height (HEIGHT= in the ODS GRAPHICS statement or DESIGNHEIGHT= in 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 DISCRETELEGEND statement reduces the size of legend text:

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

Here is example output.



The AUTOITEMSIZE=TRUE option sizes the symbols proportionally to the reduced label font size.

Adding a Continuous Legend

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
BUBBLEPLOT	COLORRESPONSE=	COLORMODEL= REVERSECOLORMODEL=
CONTOURPLOTPARM	CONTOURTYPE=	COLORMODEL= REVERSECOLORMODEL= NLEVELS= NHINT=
HEATMAPPARM	COLORRESPONSE=	COLORMODEL=
SCATTERPLOT	MARKERCOLORGRADIENT=[1]	COLORMODEL= REVERSECOLORMODEL=

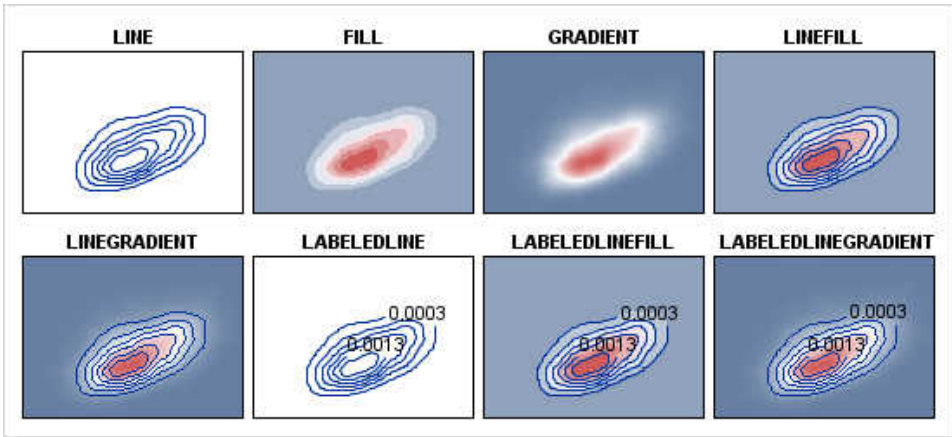
Plot Statement	Enabling Plot Option	Related Plot Options
SURFACEPLOTPARM	SURFACECOLORGRADENT =[1]	COLORMODEL= REVERSECOLORMODEL =
WATERFALLCHART	COLORRESPONSE=	COLORMODEL=
		Starting with the third maintained release for SAS 9.4, COLORSTAT=

1 In SAS 9.4M2, the MARKERCOLORGRADIENT= and SURFACECOLORGRADENT= options are deprecated and replaced with the COLORRESPONSE= option. The MARKERCOLORGRADIENT= and SURFACECOLORGRADENT= options are still honored, but SAS recommends that you use the COLORRESPONSE= option instead.

Starting with SAS 9.4M3, continuous legends can be used with the following additional plot statements when the enabling plot option is also specified.

Plot Statement	Enabling Plot Option	Related Plot Options
BARCHART	COLORRESPONSE=	COLORMODEL=
	COLORBYFREQ=	COLORSTAT= COLORMODEL=
BARCHARTPARM	COLORRESPONSE=	COLORMODEL=
HEATMAP	COLORRESPONSE=	COLORMODEL= COLORSTAT= REVERSECOLORMODEL =
HIGHLOWPLOT	COLORRESPONSE=	COLORMODEL=
LINECHART	COLORRESPONSE=	COLORMODEL=
SERIESPLOT	COLORRESPONSE=	COLORMODEL=
VECTORPLOT	COLORRESPONSE=	COLORMODEL=

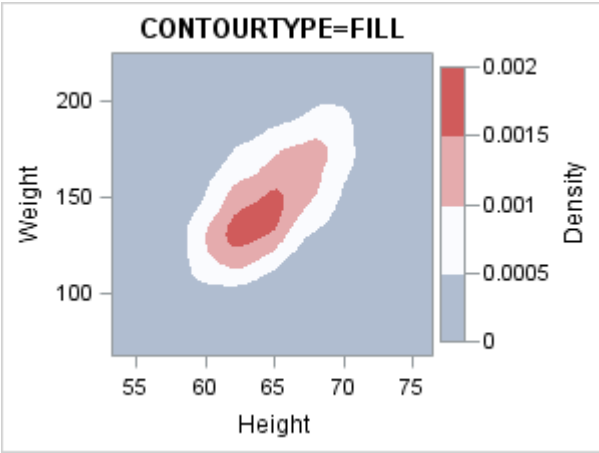
A contour plot provides the CONTOURTYPE= option, which you can use to manage the contour display. The following graph illustrates the values that are available for the CONTOURTYPE= option.



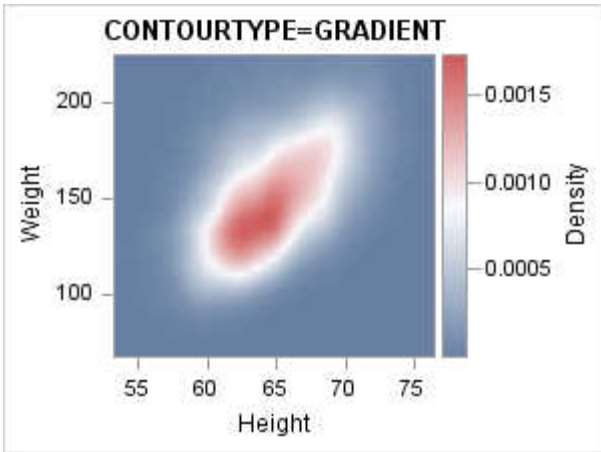
All of the variations that support color, except for LINE and LABELEDLINE, can have a legend that shows the value of the required Z= column. The following example generates a contour plot with CONTOURTYPE=FILL.

```
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;
```

Here is the output.



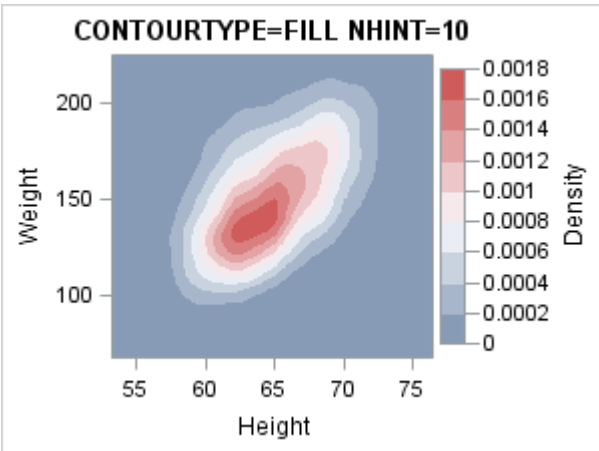
Here is the output when CONTOURTYPE=GRADIENT.



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. In the following CONTOURPLOTPARM statement, options NHINT= and NLEVELS= are used to change the number of levels (ranges) of the contour.

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

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. Here is example output.



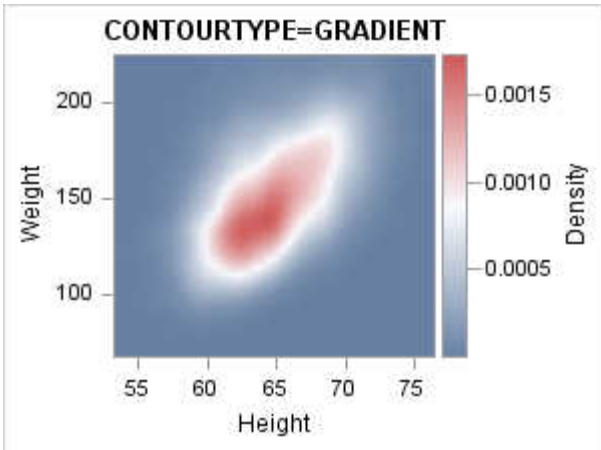
You can think of a GRADIENT contour as a FILL contour with a very large number of levels. A color ramp is displayed with an axis that shows reference points that are within the data range. The number of reference points is determined by default.

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. Here is an example.

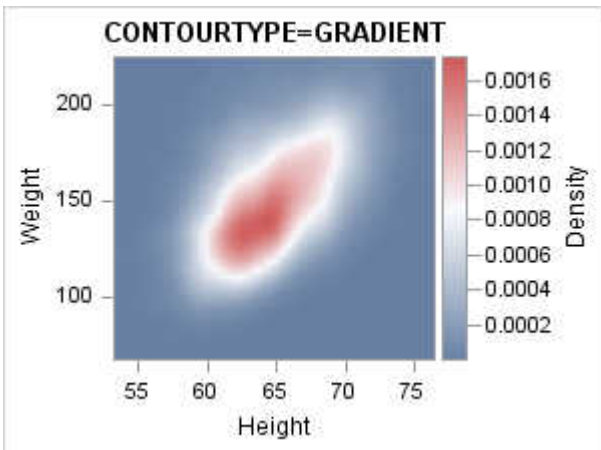
```
proc template;
  define statgraph contour;
    begingraph;
```

```
      entrytitle "CONTOURTYPE=GRADIENT";
      layout overlay / xaxisopts=(offsetmin=0 offsetmax=0)
        yaxisopts=(offsetmin=0 offsetmax=0);
        contourplotparm x=Height y=Weight z=Density / name="cont"
          contourtype=gradient;
        continuouslegend "cont" / title="Density" valuecounthint=5;
      endlayout;
    endgraph;
  end;
run;

ods graphics / antialiasmax=3600;
proc sgrender data=sashelp.gridded template=contour;
  where height>=53 and weight<=225;
run;
```

Here is the output.



Here is the output when 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

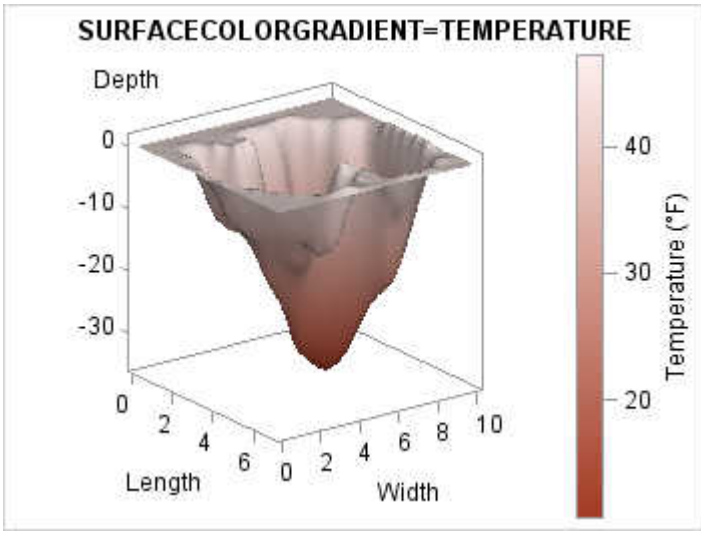
Contour plots, surface plots, and heat map plots support the use of color gradients to represent response values. For example, the SURFACEPLOTPARM statement provides the SURFACECOLORGRADIENT=*numeric-column* setting to map surface colors to a continuous gradient and enable the use of a continuous legend. All surface types (FILL, FILLGRID, and WIREFRAME) can be used. The COLORMODEL= and REVERSECOLORMODEL= options also apply. Here is an example.

```
ods escapechar="^"; /* Define an escape character */

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 (^{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;
```

Notice the coding that is used to embed a degree symbol into the legend title. For more information about using symbols in text, see “Managing the String on Text Statements” on page 324.

Here is the output.

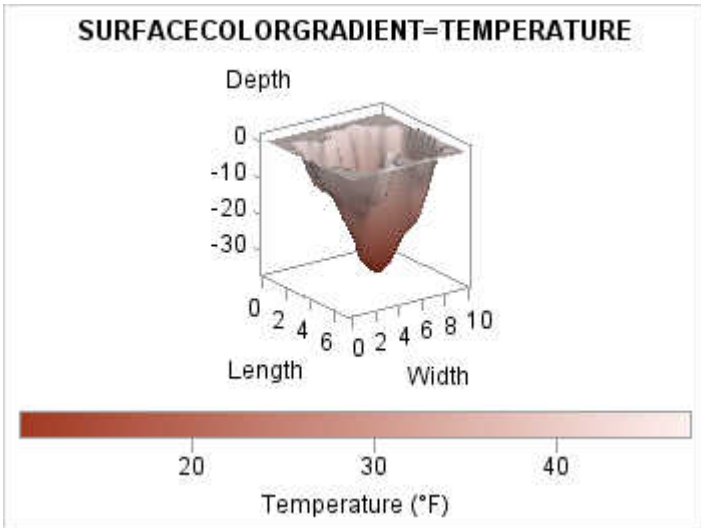


For more information about surface plots, see Chapter 13, “Creating Overlay 3-D Graphs Using the OVERLAY3D Layout ,” on page 187.

When you use VALIGN=BOTTOM or VALIGN=TOP instead of the HALIGN= option, the default orientation of the legend automatically becomes ORIENT=HORIZONTAL. Here is the modified CONTINUOUSLEGEND statement:

```
continuouslegend "surf" /
  title="Temperature (^{unicode '00B0'x}F)"
  valign=bottom;
```

Here is the output.

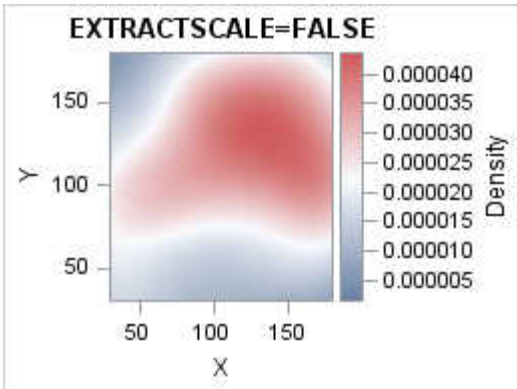


Scaling the Legend Values

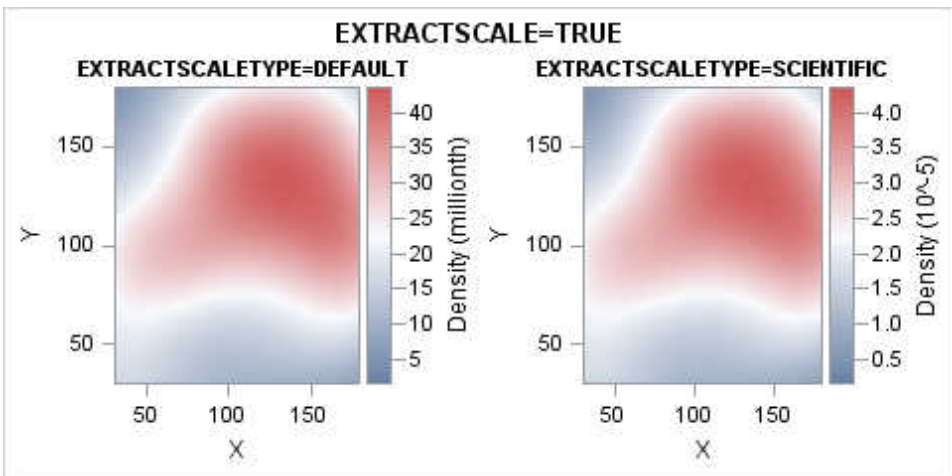
The EXTRACTSCALE= option enables you to extract a scale from the legend values in order to save space. By default, a named scale (millions, billions, and so on) is used for values of 999 trillion or less. A scientific notation scale (10^*n*) is used for values over 999 trillion. For large values, the scale factor is set to ensure that the absolute value of the largest value is greater than 1. For small fractional values, the scale factor is set to ensure that the absolute value of the smallest value is greater than 1. The EXTRACTSCALETYPE= option enables you to specify a scientific notation scale for values less than 999 trillion. The scale that is used is appended to the legend title.

Note: You must specify a title for the legend in order to use this feature.

For example, consider the legend in the following figure.



Because of the fractional density values, the legend takes up a significant amount of space in the graph. To reduce the footprint of the legend, you can specify EXTRACTSCALE=TRUE to automatically scale the density values. You can also use the EXTRACTSCALETYPE= option to select the scale type. The following figure shows the result.



21

Adding Insets to Your 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 dynamic variables or macro variables
- columns that are assigned to an INSET= option on statements that support the option

Inset information is often specified in 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 19, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 317 and Chapter 14, “Creating Gridded Graphs Using the GRIDDED Layout ,” on page 207.

Creating a Simple Inset with an ENTRY Statement

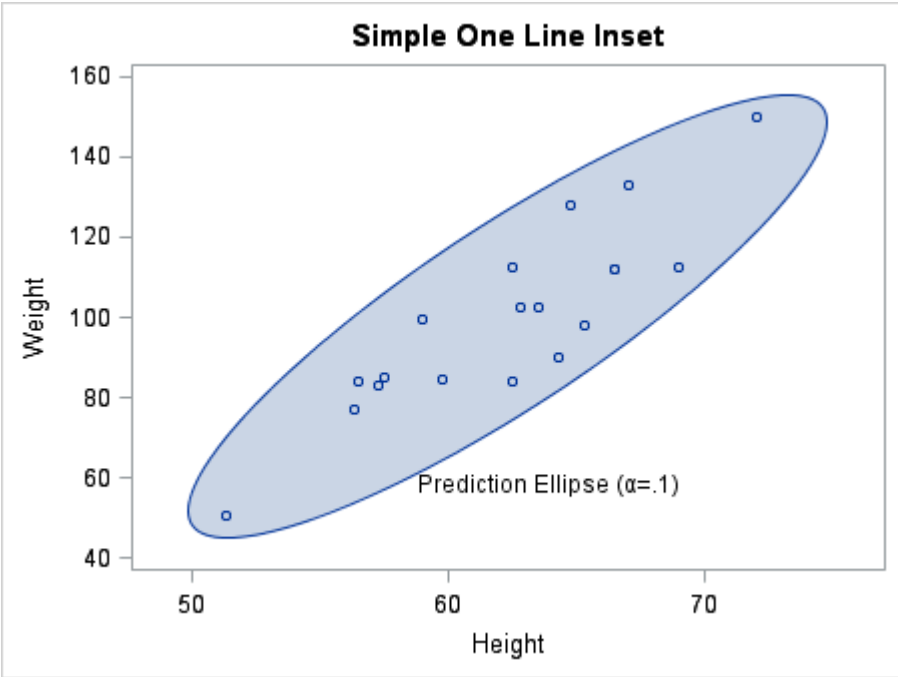
You can use an ENTRY statement to create a simple inset within most layout blocks.

If you create the insets within a 2-D 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 as shown in the following example.

```
proc template;
  define statgraph  ginset;
    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;
  end;
run;

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

Here is the output.



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 graphics 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 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 layout block 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 layout block left-justifies the text for each ENTRY statement:

```
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;
```

Here is the result.

N	5203
Mean	119.96
Std Dev	19.98

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 layout block 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 layout block illustrates how to change the column justification and add extra space between the columns with the COLUMNGUTTER= option.

```
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;
```

Borders are 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. Here is the result.

N	5203
Mean	119.96
Std Dev	19.98

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 as shown in the following layout block.

```
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;
```

Notice that the outer GRIDDED layout has one column and two rows (the nested GRIDDED layout is treated as one cell). Here is the result.

Stat Table

N	5203
Mean	119.96
Std Dev	19.98

Positioning an Inset

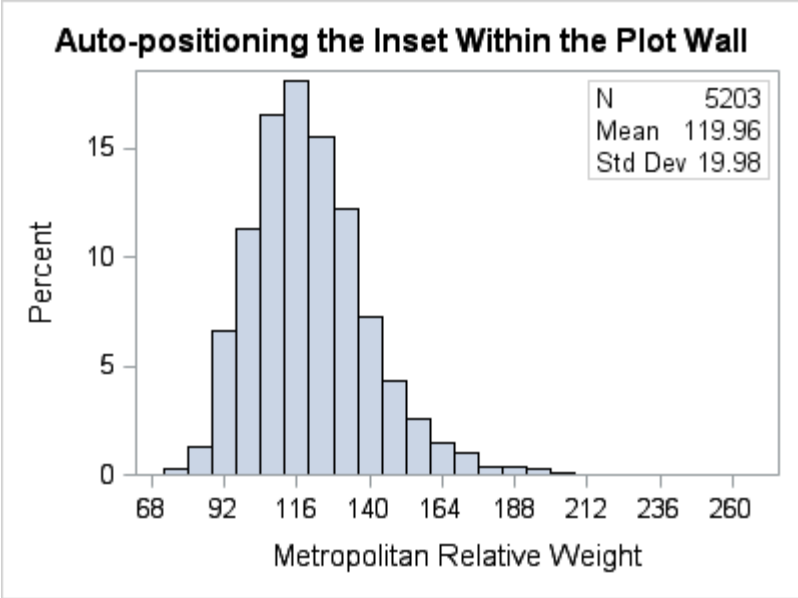
If a table of text is used as an inset within a 2-D overlay-type layout, you can position the table within the parent layout with options on the LAYOUT GRIDDED statement. You can use the AUTOALIGN= option to automatically position the inset to avoid collision with scatter points, lines, bars, and other 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;

ods graphics / reset width=400px;
proc sgrender data=sashelp.heart template=ginset3a;
run;
ods graphics / reset;
```

Here is the output.



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.

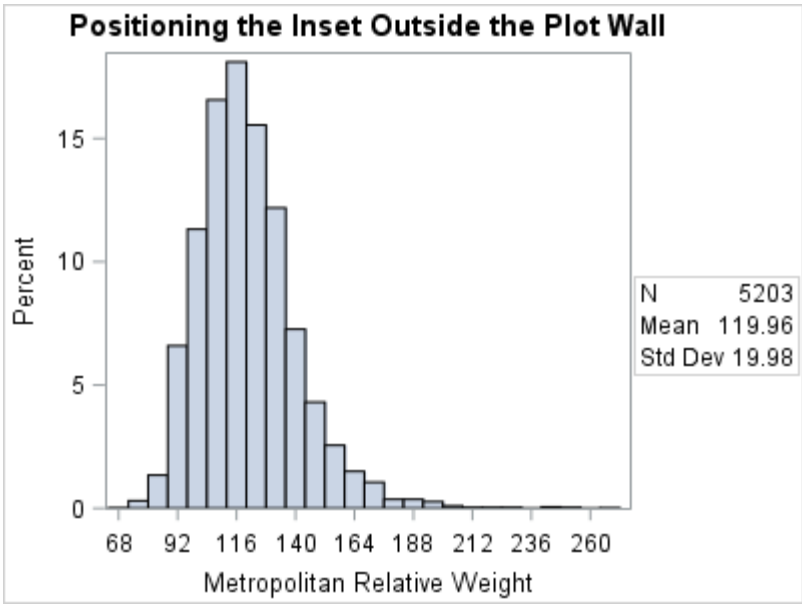
Note: The AUTO setting for the AUTOALIGN= option works best when the layout contains only scatter plots. When other plot types are present, the ENTRY text might overlap a graphics element in the plot area.

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 graphics / reset width=400px;
proc sgrender data=sashelp.heart template=ginset3b;
run;
ods graphics / reset;
```

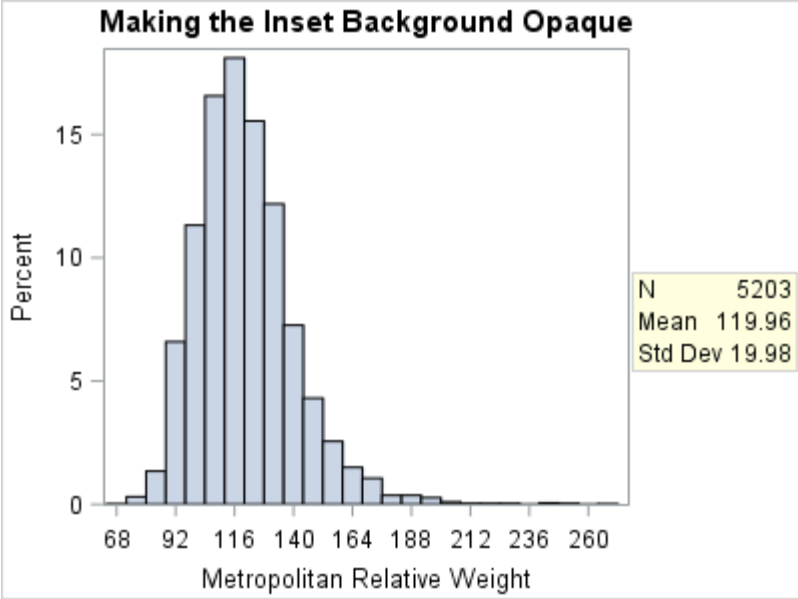
Here is the output.



By default, the background of a GRIDDED layout and the background of ENTRY statements 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 is seen through the inset. You can make the background of the insert opaque and set its background color to highlight the information, as shown in the following LAYOUT GRIDDED statement.

```
layout gridded / columns=1 border=true
  opaque=true backgroundcolor=lightyellow border=true;
```

Here is example output.



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. Hardcoding 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 run time, 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.

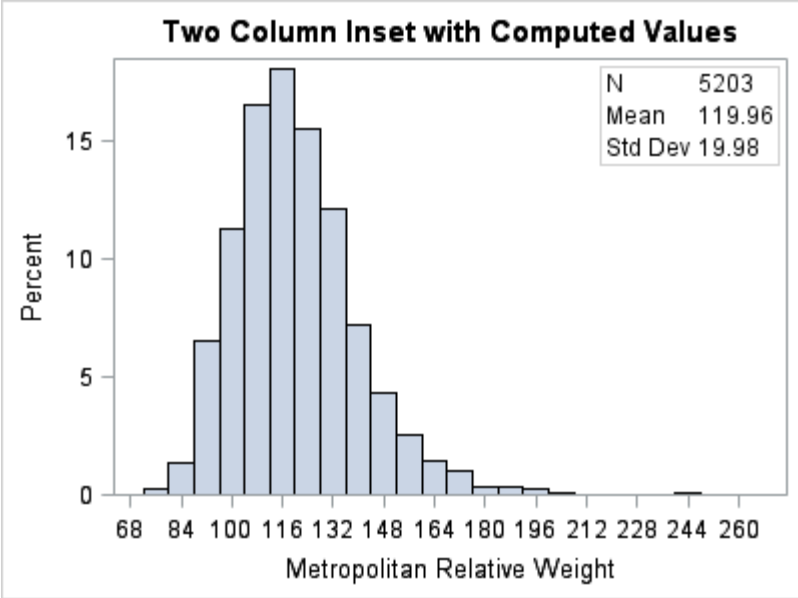
```
proc template;
 define statgraph  ginset4a;
  dynamic VAR;
  begingraph;
    entrytitle "Two Column Inset with Computed Values";
    layout overlay;
      histogram VAR;
```

```
      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;

ods graphics / reset width=400px;
proc sgrender data=sashelp.heart template=ginset4a;
  dynamic VAR="mrw";
run;
```

Note the following:

- Dynamic variable VAR is first referenced in the HISTOGRAM statement, where it is used to represent the variable that provides numeric values for the histogram.
- Dynamic variable 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 variable VAR so that it resolves at run time to column MRW from the Sashelp.Heart data set. Because the dynamic variable resolves to a column name, the value that is assigned to it is enclosed in quotation marks. (Values for dynamic variables that resolve to column names or strings should be quoted. Numeric values should not be quoted.)

Here is the output.



See Chapter 31, “Using Conditional Logic and Expressions in Your Templates,” on page 619 for more information about the functions that can be used in the EVAL function. See Chapter 30, “Using Dynamic Variables and Macro Variables in Your Templates,” on page 605 for more information about using dynamic variables and macro variables in GTL templates.

Creating an Inset from Values That Are Passed to the Template

Overview of Importing Data into a Template

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 dynamic variables or macro variables to "import" the computed values at run time.

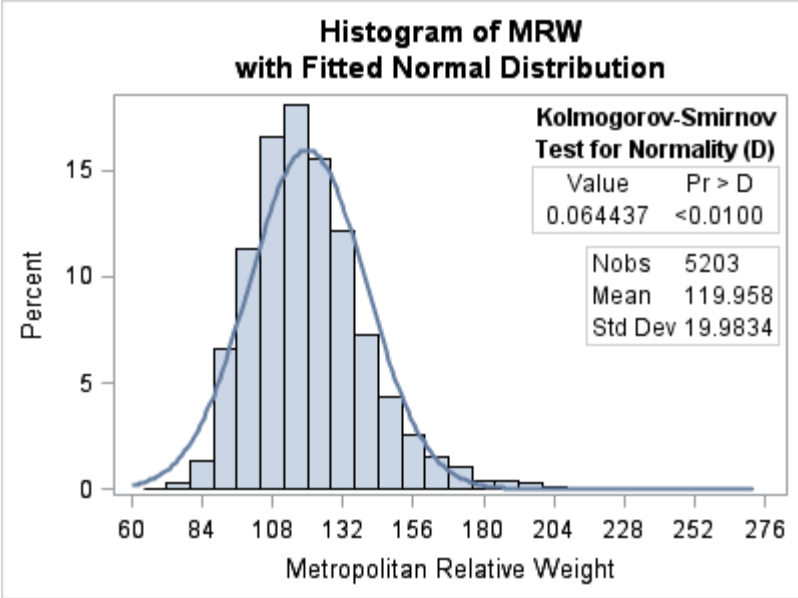
The following example explains how to create and call a macro that can pass a data set name and variable name to a previously compiled GTL template. To follow this example, you should understand the topics that are discussed in Chapter 30, “Using Dynamic Variables and Macro Variables in Your Templates,” on page 605.

For this example, a macro named HISTOGRAM is created, which accepts 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 figure shows an example.

Figure 21.1 Example Output from the %HISTOGRAM Macro



For more information about 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 21.1 on page 394. The template definition uses the MVAR statement to define macro variables that provide run-time values and labels for the graph insets. The MVAR statement also defines a macro variable named VAR, which is 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 is named 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;
```

Note the following:

- 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 you will see later when the macro is created, 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 run time.

  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 394 template GINSET was created, which declared the following macro variables:

Macro Variable	Description
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, a macro must be created that calculates values for them and invokes an SGRENDER procedure statement that specifies 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 shown in Figure 21.1 on page 394. 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;

  ods graphics / reset width=400px;
  proc sgrender data=&dsn template=ginset;
  run;
%mend;
```

Note the following:

- 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 about 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, 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 column MRW has more than 2000 observations, the Kolmogorov-Smirnov test is used in the analysis.

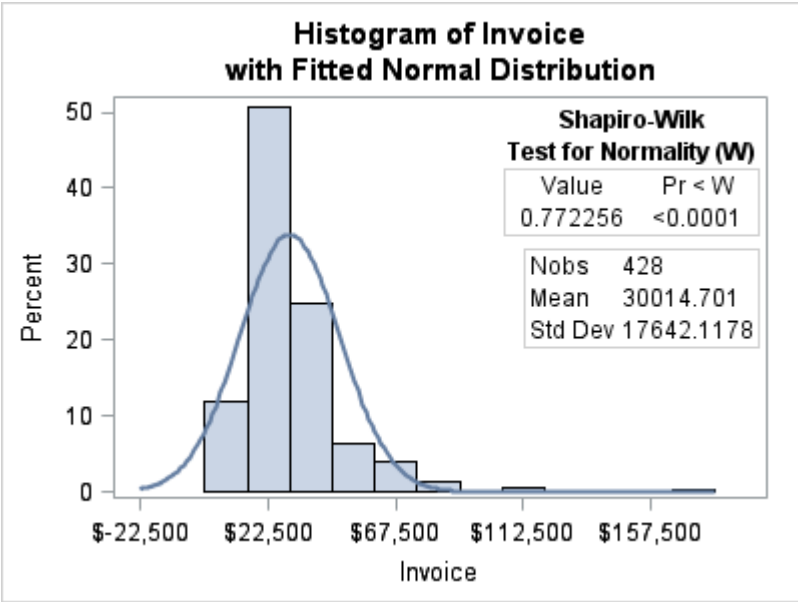
```
%histogram(sashelp.heart, mrw)
```

The output is shown in Figure 21.1 on page 394.

This next macro call passes the data set name Sashelp.Cars and the column name Invoice. Because column Invoice has 2000 or fewer observations, the Shapiro-Wilk test is used in the analysis.

```
%histogram(sashelp.cars,invoice)
```

Here is the output.



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 *SAS Graph Template Language: Reference* for complete details about these options.)

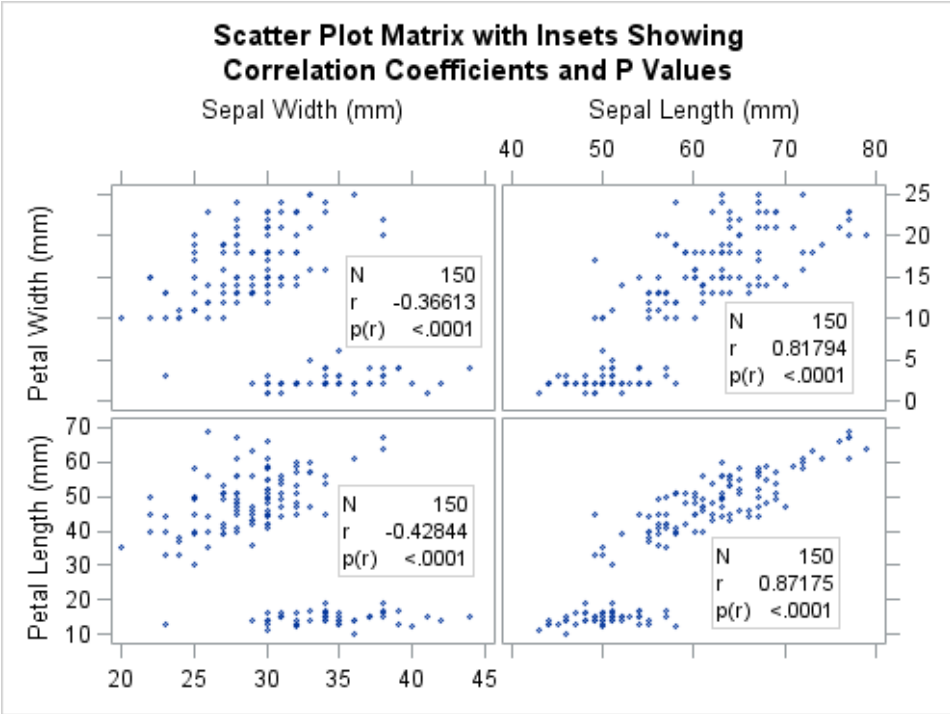
Option	Description
INSET= (*info-options*)	Determines what information is displayed in an inset. Accepts one, two, or all three of the following keywords: NOBS Number of observations PEARSON Pearson product-moment correlation PEARSONPVAL Probability value for the Pearson product-moment correlation This 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 that is displayed in the inset. Accepts TRUE (the default) or FALSE. The NOMISS=TRUE option does not exclude observations with missing values from the plot. 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.

Option	Description
	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. TITLE= specifies a title for the inset. TITLEATTRS= specifies the text properties of the inset title. 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;
```

Here is the output.



Notice that the inset position might change from cell to cell in order to avoid obscuring point markers.

Adding Insets to Classification Panels

This section requires familiarity with coding classification panels in the Graph Template Language (GTL). If you are not familiar with classification panels, see Chapter 16, “Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts,” on page 255.

About Cell Insets in 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 398. However, unlike the SCATTERPLOTMATRIX statement, predefined information is not available for the DATALATTICE and DATAPANEL layouts. Therefore, for the INSET= option, you must create the columns for the information that you want to display in the inset and merge that information with the input data before the graph is rendered. You can merge the inset data and analysis data by using match-merging (BY statement) or by using a one-to-one merging (no BY statement). The manner in which the data is merged is referred to as the data scheme for classification panel inset data. Use the DATASCHEME= option to specify the data scheme that was used to merge your inset and analysis data.

After you have merged the inset and analysis data, use the following options in your LAYOUT DATAPANEL or LAYOUT DATALATICE statement:

- Use the INSET= option to specify the name of one or more columns that contain the information that you need for your insets.
- Use the DATASCHEME= option to specify the data scheme that you used to merge your inset and analysis data. Specify LIST (one-to-one merging) or MATCH (match-merging).

Note: The match-merging data scheme is the preferred data scheme for merging the inset and analysis data.

Starting with SAS 9.4M1, you can also display an axis table or block plot in an inner margin of the PROTOTYPE layout in order to inset a table of axis-aligned values along a row or column axis. For information about creating axis-aligned insets, see "Creating an Axis-Aligned Inset in a Classification Panel" on page 420. The following sections describe how to add insets by using match-merged data and one-to-one-merged data.

Adding Insets By Using the Match-Merging Data Scheme

The match-merging data scheme merges the inset data and analysis data according to variables listed in a BY statement. It is the recommended method of generating data for your insets. By default, the LAYOUT DATAPANEL and LAYOUT DATALATICE statements assume that one-to-one merging was used to merge the data. To specify the match-merging data scheme, you must include the DATASCHEME=MATCHMERGED option in your LAYOUT DATAPANEL or LAYOUT DATALATICE statement. This example shows you how to use the match-merging data scheme to merge your inset and analysis data, and how to add the insets to your classification panel. This example uses the Sashelp.Cars data set as the data source.

Here are the high-level steps that are performed in this example:

1. Generate the inset data from the Sashelp.Cars data set and write it to the Mileage data set.
2. Sort the Sashelp.Cars data by Cylinders, Origin, and Type, and then write the sorted data to the Cars data set.
3. Use match-merging to merge the Cars and Mileage data sets by Cylinders and Origin, and then write the merged data to the AvgMileage data set.
4. Create the template for the classification panel.
5. Generate the graph by using the AvgMileage data.

Here is the SAS code that generates the inset data from the Sashelp.Cars data set, and then writes it to the Mileage data set.

```
/* Generate the inset information and write it to data
   set MILEAGE.
*/
```

```
proc summary data=sashelp.cars completetypes;
  where type in ("Sedan" "Truck" "SUV")
    and cylinders in (4 6 8);
  class cylinders origin;
  var mpg_city;
  output out=mileage MEAN=Mean N=Nobs / noinherit;
  TYPES CYLINDERS*ORIGIN;
run;
```

The SUMMARY procedure calculates the number of observations and the mean of Mpg_City for each of the classification interactions listed in the TYPES statement. Cylinder*Origin*Type is the crossing that each cell's bar chart needs. The COMPLETETYPES option creates summary observations even when the frequency of the classification interactions is zero. In addition, 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).

Here is the SAS code that sorts the data in Sashelp.Cars by Cylinders, Origin, and Type, and then writes the sorted data to the Cars data set. This data order is required for the match-merging that is completed in the next step.

```
/* Sort the analysis data by the variables that are
   used for the inset information: CYLINDERS, ORIGIN,
   and TYPE.
*/
proc sort data=sashelp.cars out=cars;
  by cylinders origin type;
run;
```

Here is the SAS code that performs the match-merging of Cars and Mileage data sets by Cylinders and Origin, and then writes the merged data to the AvgMileage data set.

```
/* Match-merge the analysis data with inset data by
   CYLINDERS and ORIGIN.
*/
data avgmileage;
  merge cars mileage;
  BY CYLINDERS ORIGIN;
  format mean mpg_city 4.1;
run;
```

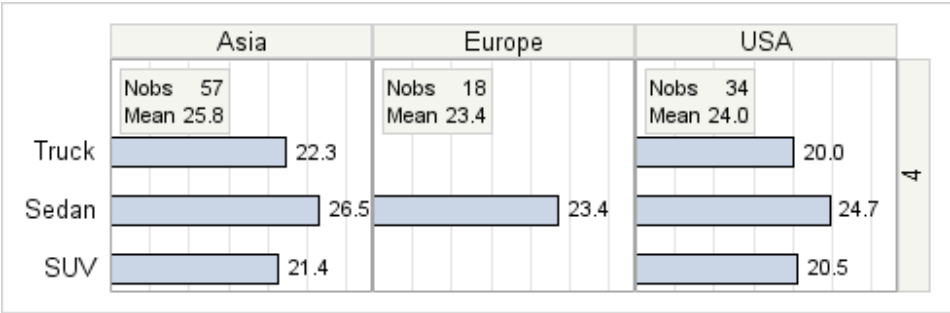
The template code in this example defines a template named PANEL. This template requires that the input data contain separate columns for the items listed in the following table.

Required Columns	Example
classification variables	`columnvar=origin rowvar=cylinders`
inset information	`inset=(nobs mean)`
bar chart data	`category=type response=mean`

The INSET=(Nobs Mean) option in this template is used to reference input data columns that are named Nobs and Mean. When the graph is rendered, the values that are stored in these columns are displayed in the inset. In the inset display in this

example, one row is displayed for each column that is listed in the INSET= option, 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 values for these columns should match the number of cells in the classification panel and the order of the values should correspond with the sequence in which the cells are populated. The DATASCHEME=MATCHED suboption in the INSETOPTS= option list specifies that the match-merging data scheme was used to merge the inset and analytic data.

This template also adds a maximum row axis offset by using the OFFSETMAX=0.4 option to make room for the insets. In this case, OFFSETMAX=0.4 is sufficient, but the setting will vary case-by-case. The first row of the classification panel with insets appears as shown in the following figure.



Here is the SAS code that creates template PANEL.

```
/* Create the graph template for the classification panel. */
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=(nobs mean)
        insetopts=(border=true datascheme=matched
        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=MPG_CITY / orient=horizontal
            STAT=MEAN barwidth=.5 barlabel=true;
        endlayout;
      endlayout;
    endgraph;
  end;
run;
```

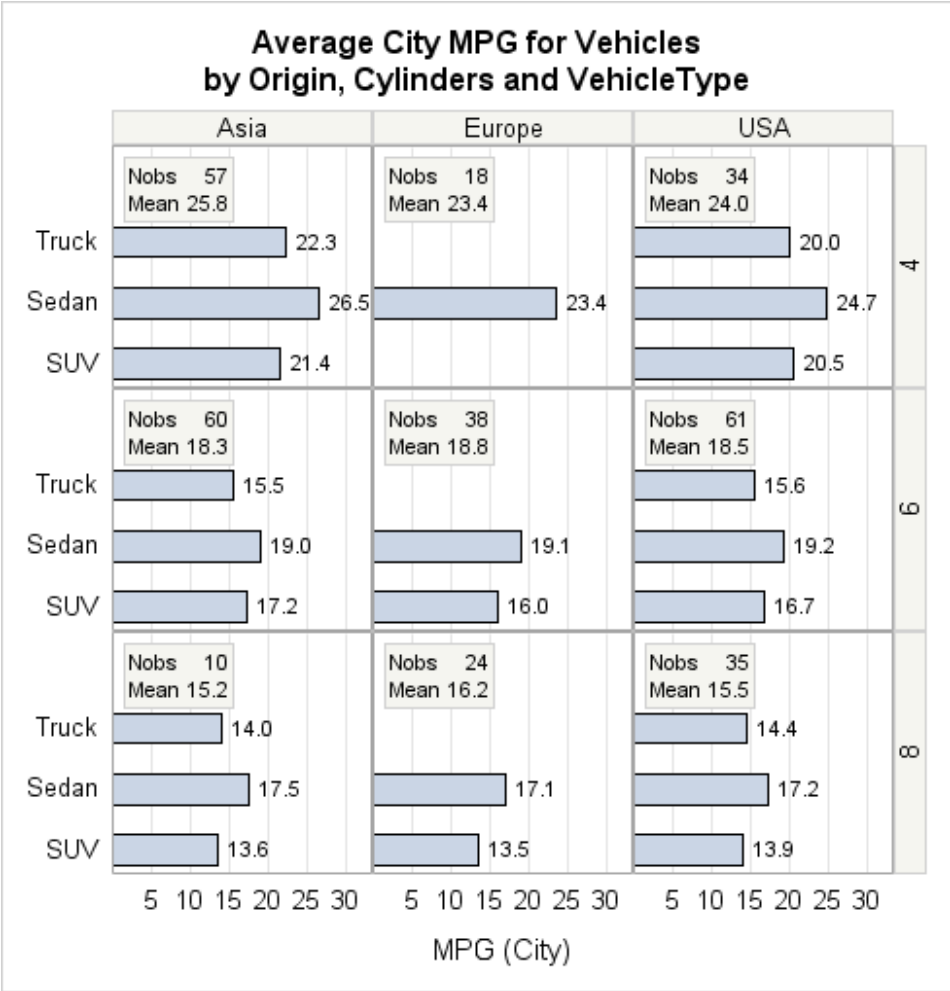
Finally, here is the SAS code that renders the graph by using the data in the AvgMileage data set.

```
/* Generate the graph using the AVGMILEAGE data. */
```

```
ods graphics / reset;
proc sgrender data=avgmileage template=panel;
  where type in ("Sedan" "Truck" "SUV")
    and cylinders in (4 6 8);
run;
```

Here is the output.

Figure 21.2 *Classification Panel with Cell Insets*



Adding Insets By Using the One-To-One-Merging Data Scheme

The one-to-one data scheme merges the inset data and the analysis data according to the observations' position in the data sets. No BY statement is used. By default, the LAYOUT DATAPANEL and LAYOUT DATALATICE statements assume that one-to-one merging was used to merge the inset data with the analysis data. In that case, the inset for each cell is populated in data order. That is, the inset data in the first observation is used for the cell 1 inset, the inset data for the second observation is used for the cell 2 inset, and so on. This process requires that the inset data be

placed first in the data set and that it be ordered to correspond correctly with the cells in the classification panel. Because of this requirement, the one-to-one merging data scheme can be more complicated than the match-merging data scheme.

Note: The match-merging data scheme is the preferred method of merging your inset data and analysis data. See "Adding Insets By Using the Match-Merging Data Scheme" on page 402.

This example shows you how to use the one-to-one merging data scheme to merge your inset and analysis data, and how to add the insets to your classification panel. The Sashelp.Cars data set is used as the data source.

Here are the high-level steps that are performed in this example:

1. Summarize the data in Sashelp.Cars for the cell insets (Cylinders*Origin) and the cell bar charts (Cylinders*Origin*Type). Write the output to the Mileage data set.
2. Extract the cell summary data (_TYPE_=6) from the Mileage data set, rename the columns, and write the data to the Overall data set.

   Note: The columns must be renamed in order to prevent overwriting any values when the data is merged in the next step.
3. Use one-to-one merging to merge the Mileage and Overall data sets to the AvgMileage data set.
4. Create the template for the classification panel.
5. Generate the graph by using the AvgMileage data.

Here is the SAS code that summarizes the Sashelp.Cars data for the cells and bar charts.

```
/* 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(drop=_freq_) mean=Mean n=Nobs / noinherit;
  types cylinders*origin cylinders*origin*type;
run;
```

The SUMMARY procedure 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 Cylinders*Origin*Type is the crossing needed for each cell's bar chart. The COMPLETETYPES option creates summary observations even when the frequency of the classification interactions is zero. In addition, 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, you must ensure that the order of the observations in the data corresponds to the column order for the CLASS statement of the SUMMARY procedure. Because the panel cells are populated across one row

before proceeding to the next row, the values of the panel's row variable (Cylinders) determine the panel order. The values 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.

The following partial listing shows the order of the observations.

Output 21.1 *Partial Listing of the Mileage Data Set*

Obs	Cylinders	Origin	Type	_TYPE_	Mean	Nobs
1	4	Asia		6	25.8421	57
2	4	Europe		6	23.4444	18
3	4	USA		6	24.0000	34
4	6	Asia		6	18.3000	60
5	6	Europe		6	18.7632	38
6	6	USA		6	18.4754	61
7	8	Asia		6	15.2000	10
8	8	Europe		6	16.1667	24
9	8	USA		6	15.5143	35
10	4	Asia	SUV	7	21.4000	5
11	4	Asia	Sedan	7	26.5102	49
12	4	Asia	Truck	7	22.3333	3
…						

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, you need to store the _TYPE_= 6 observations in new columns. To do this, you must first extract the inset data into a separate data set named Overall and rename the columns. You must rename the columns in order to prevent overwriting any values when this data is merged with the Mileage data. Here is the DATA step that writes the inset information to the Overall data set.

```
/* Extract the inset information, rename the columns,
   and write it to data set Overall.
*/
data mileage (drop=_type_)
  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;
```

Finally, you must merge the Mileage and Overall data sets into the Summary data set by using one-to-one merging (no BY statement).

```
/* Merge MILEAGE and OVERALL and write to SUMMARY. */
data summary;
  merge mileage overall;
  label Mean="MPG (City)" CellNOBS="Nobs" CellMean="Mean";
  format mean cellMean 4.1;
run;
```

The following listing shows a sample of the merged data.

Output 21.2 *Partial Listing of the Summary Data Set*

Obs	Cylinders	Origin	Type	Mean	Nobs	Cyl	Origin	Mean	Nobs
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	.		.	.
…									

The template code in this example defines a template named PANEL. This template requires that the input data contain separate columns for the items listed in the following table.

Required Columns	Example
classification variables	`columnvar=origin rowvar=cylinders`
inset information	`inset=(cellNobs cellMean)`
bar chart data	`category=type response=mean`

This template uses INSET=(CellNobs CellMean) to reference input data columns that are named CellNobs and CellMean. When the graph is rendered, the values that are stored in these columns are 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. This template also adds a maximum row axis offset using the OFFSETMAX=0.4 option to make room for the insets.

Here is the code that creates the template in this example and that renders that graph.

```
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=(cellNobs cellMean)
        insetopts=(border=true datascheme=list
          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 category=type response=mean / orient=horizontal
              barwidth=.5 barlabel=true;
          endlayout;
        endlayout;
      endgraph;
    end;
  run;

  proc sgrender data=summary template=panel;
  run;
```

The output is shown in Figure 21.2 on page 405.

Creating Axis-Aligned Insets

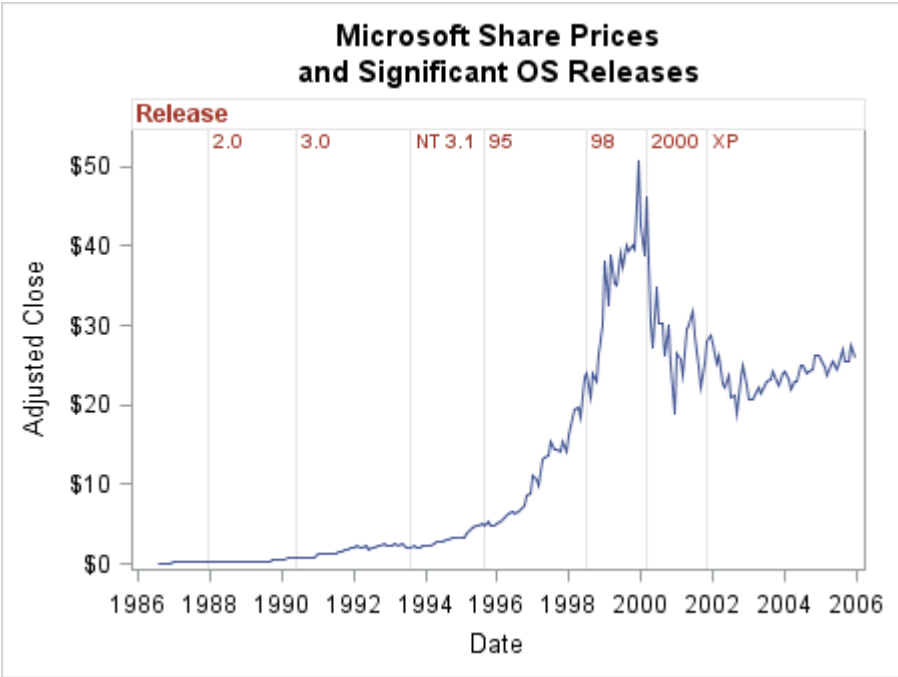
Sometimes you want an inset to provide information about values along an axis. The information is aligned with values on the axis. You can use a block plot or an axis table to display information that is aligned with axis values.

Creating an Axis-Aligned Inset with a Block Plot

Displaying Axis-Aligned Values with a Block Plot

You can use a box plot to display axis-aligned values along an axis in a plot. Here is an example that uses a box plot to display events along the X axis of a series plot. The series plot shows the yearly adjusted closing price of Microsoft stock from 1986 to 2006. A box plot is used to show along the X axis when the major releases of Microsoft Windows occurred during that period. The release dates and release information are defined in the plot data. The block plot creates strips of outlined rectangular blocks where the width of each block corresponds to the specified release dates along the X axis. In each block, the release information that corresponds to that block is displayed in the top left corner. Here is the output for this example.

Figure 21.3 *Displaying Axis-Aligned Values Using a Block Plot*



The input data for this example must contain data for both the series plot and the block plot. The simplest way to construct the appropriate data is to match-merge the Microsoft release data and the Microsoft stock data. The first step is to create data set Msevents, which contains the Microsoft release dates. The next step is to sort the Microsoft stock data by date, match-merge it with Msevents, and then write it to data set Events. Here is the code that generates the data for this example.

Example Code 21.1 *SAS Code That Generates the Events Data Set*

```
/* Create a data set of the release dates. */
data MSevents;
  input Date date9. Release $7.;
datalines;
09dec1987 2.0
22may1990 3.0
01aug1993 NT 3.1
24aug1995 95
25jun1998 98
17feb2000 2000
25oct2001 XP
;
run;

/* Sort the Microsoft stock data by date. */
proc sort data=sashelp.stocks(keep=date stock adjclose)
  out=MSstock;
  where stock="Microsoft";
  by date;
run;

/* Match-merge the release data and the stock data. */
data events;
  merge MSstock MSevents;
```

```
  by date;
run;
```

Output 21.3 *Partial Listing of the Events Data Set*

Obs	Stock	Date	Close	Release
…				
15	Microsoft	01OCT87	$0.30	
16	Microsoft	02NOV87	$0.27	
17	Microsoft	01DEC87	$0.33	
18		09DEC87	.	2.0
19	Microsoft	04JAN88	$0.34	
20	Microsoft	01FEB88	$0.36	
21	Microsoft	01MAR88	$0.34	
22	Microsoft	04APR88	$0.33	
23	Microsoft	02MAY88	$0.35	
24	Microsoft	01JUN88	$0.40	
…				

As shown in the partial listing of the Events data set, the first release event occurs in observation 18. In the observations for the release dates, the Stock and Close column values are missing. In the Microsoft stock observations, the Release column value is missing.

Here is the code that defines the template for this example and then renders the graph.

```
/* Create the template for the graph. */
proc template;
  define statgraph blockplot1;
    begingraph;
      entrytitle "Microsoft Share Prices";
      entrytitle "and Significant OS Releases";
      layout overlay / yaxisopts=(offsetmax=0.075);
        blockplot x=date block=release /
          display=(outline values label)
          valuevalign=top valuehalign=left
          labelposition=top
          extendblockonmissing=true
          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;

/* Render the graph. */
proc sgrender data=events template=blockplot1;
   format date year4.;
run;
```

In the template code, the block plot is overlaid with a series plot to create an axis-aligned inset. The OFFSETMAX= option is specified for the Y axis in order to reserve space for the block plot values. Notice that the BLOCKPLOT statement requires two input columns: one for the X= argument and another for the BLOCK=

argument. The EXTENDBLOCKONMISSING=TRUE option extends the blocks across the missing values of the Release column. The BLOCK= transition points control the boundary of each block and the text that is displayed. In this case, a new block begins at each nonmissing value of the Release column. The range of the X= values between two block transition points determine the width of each block.

The BLOCKPLOT statement supports the following options for controlling the content, position, and appearance of the blocks and text information:

DISPLAY= (<OUTLINE> <FILL> <VALUES> <LABEL>)
specifies the features to display

EXTENDBLOCKONMISSING=TRUE | FALSE
specifies whether the block continues with the previous nonmissing value or a new block is started when a missing value is encountered in the Block column

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

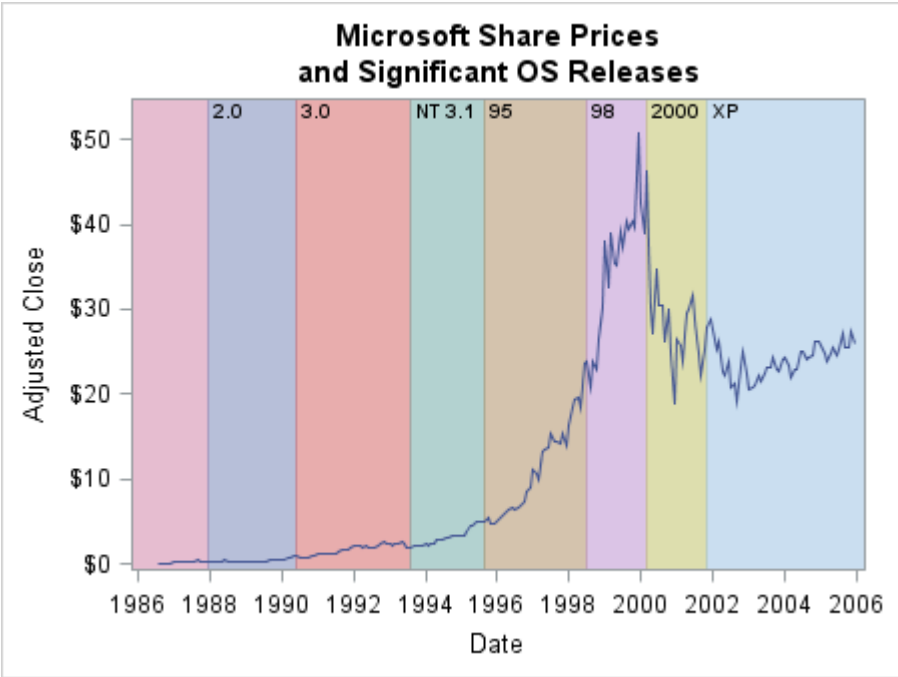
LABELATTRS=*style-element*
specifies font properties for the block label

For information about the BLOCKPLOT statement, see "BLOCKPLOT" in *SAS Graph Template Language: Reference*.

Displaying Axis-Aligned Filled Segments with a Block Plot

You can use a block plot to display axis-aligned filled segments along an axis in a plot. It is useful in a series plot, for example, for highlighting periods of time along a time axis. The following example demonstrates how to use filled segments as an alternate method to that used in "Displaying Axis-Aligned Values with a Block Plot" on page 409. To display the release information as segments along the X axis, the outlines are removed and the blocks are filled with colors. The same data (Example Code 21.1 on page 410) is used to generate the graph. Here is the output for this example.

Figure 21.4 *Displaying Axis-Aligned Filled Segments Using a Block Plot*



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).

Here is the code that defines the template for this example and then renders the graph.

```
/* Create the template for the graph. */
proc template;
  define statgraph blockplot1a;
    begingraph;
      entrytitle "Microsoft Share Prices";
      entrytitle "and Significant OS Releases";
      layout overlay / yaxisopts=(offsetmax=0.075);
        blockplot x=date block=release / display=(fill values)
          valuevalign=top valuehalign=left
          labelposition=top
          extendblockonmissing=true
          valueattrs=GraphDataText(weight=bold)
          filltype=multicolor
          datatransparency=.5;
        seriesplot x=date y=adjClose / lineattrs=GraphData1;
    endlayout;
   endgraph;
  end;
run;

/* Render the graph. */
proc sgrender data=events template=blockplot1a;
```

```
    format date year4.;
run;
```

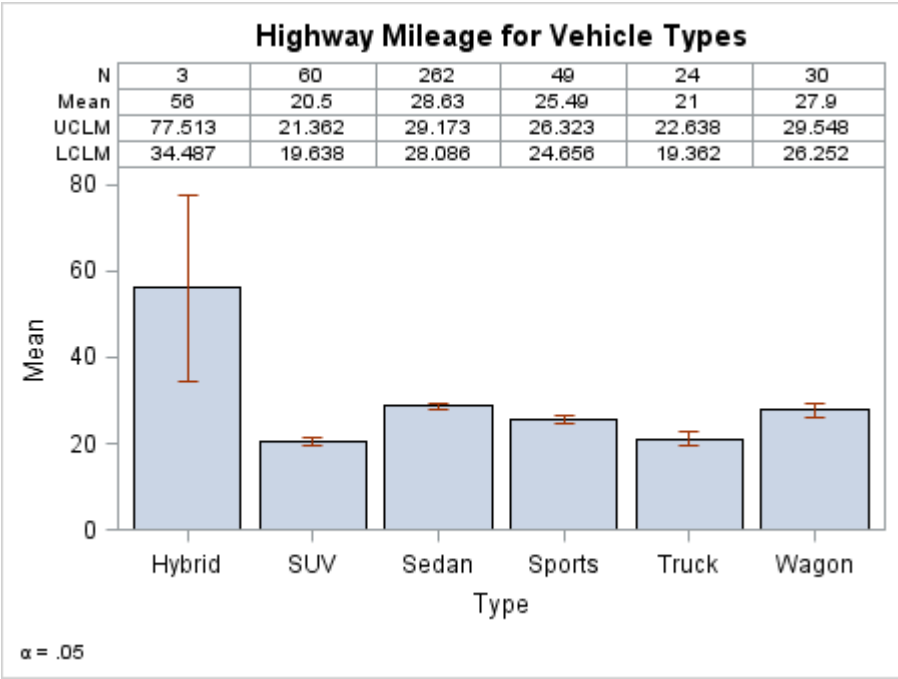
In this template, the FILLATTRS=MULTICOLOR setting ensures that the colors will be obtained from the GraphData1–GraphDataN style elements of the current style. Transparency is added to fade the colors. The block label "Release" is suppressed, and the horizontal alignment of the block values is left.

Creating an Axis-Aligned Table of Values with a Block Plot

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. The following example uses the BLOCKPLOT CLASS= option to create an axis-aligned table of statistics at the top of a bar chart. Here is the output for this example.

Here is the output for this example.

Figure 21.5 *Axis-Aligned Multi-row Table Created Using a Block Plot*



To create this graph, some data set up is necessary. First, use PROC SUMMARY to summarize the data in Sashelp.Cars and write it to data set Stats.

Example Code 21.2 *SAS Code That Generates the Stats Data Set*

```
/* 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 / noinherit;
run;
```

Output 21.4 *Listing of Stats Data Set*

Obs	Type	N	Mean	UCLM	LCLM
1	Hybrid	3	56.0000	77.5133	34.4867
2	SUV	60	20.5000	21.3620	19.6380
3	Sedan	262	28.6298	29.1732	28.0863
4	Sports	49	25.4898	26.3234	24.6562
5	Truck	24	21.0000	22.6378	19.3622
6	Wagon	30	27.9000	29.5478	26.2522

The Stats data set columns Type, Mean, UCLM, and LCLM will be used by a BARCHARTPARM statement. 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(rename=(type=type2 _name_=statname col1=stat));
  by type;
  attrib statname label=''; /* Remove the old label from statname */
  var n mean uclm lclm;
run;
```

The SAS log displays the following note when the procedure code is submitted:

Output 21.5 *Partial Listing of the Blockstats Data Set*

Obs	type2	statname	stat
1	Hybrid	N	3.000
2	Hybrid	Mean	56.000
3	Hybrid	UCLM	77.513
4	Hybrid	LCLM	34.487
5	SUV	N	60.000
6	SUV	Mean	20.500
7	SUV	UCLM	21.362
8	SUV	LCLM	19.638
9	Sedan	N	262.000
10	Sedan	Mean	28.630
11	Sedan	UCLM	29.173
12	Sedan	LCLM	28.086
…			

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 columns must be distinct variables.

```
/* Combine summary data for BARCHARTPARM with tabular data for
BLOCKPLOT */
data all;
  merge stats blockstats;
run;
```

Output 21.6 *Partial Listing of the ALL Data Set*

Obs	Type	N	Mean	UCLM	LCLM	type2	statname	stat
1	Hybrid	3	56.0000	77.5133	34.4867	Hybrid	N	3.000
2	SUV	60	20.5000	21.3620	19.6380	Hybrid	Mean	56.000
3	Sedan	262	28.6298	29.1732	28.0863	Hybrid	UCLM	77.513
4	Sports	49	25.4898	26.3234	24.6562	Hybrid	LCLM	34.487
5	Truck	24	21.0000	22.6378	19.3622	SUV	N	60.000
6	Wagon	30	27.9000	29.5478	26.2522	SUV	Mean	20.500
7		.	.	.	.	SUV	UCLM	21.362
8		.	.	.	.	SUV	LCLM	19.638
9		.	.	.	.	Sedan	N	262.000
10		.	.	.	.	Sedan	Mean	28.630
11		.	.	.	.	Sedan	UCLM	29.173
12		.	.	.	.	Sedan	LCLM	28.086
…								

Here is the code that creates the template for this example and then renders the graph.

```
/* Create the template for the graph. */
proc template;
  define statgraph blockplot2;
    begingraph;
      entrytitle "Highway Mileage for Vehicle Types";
      entryfootnote halign=left {unicode alpha} " = .05";
      layout overlay / xaxisopts=(label="Type");
        innermargin / align=top;
          blockplot x=type2  block=stat / class=statname
            includemissingclass=false
            display=(values label outline) valuehalign=center
            labelattrs=GraphDataText valueattrs=GraphDataText
            outlineattrs=(color=graphaxislines:color);
        endinnermargin;
        barchartparm x=type y=mean /
          errorlower=lclm errorupper=uclm;
      endlayout;
    endgraph;
  end;
run;

/* Render the graph. */
proc sgrender data=all template=blockplot2;
run;
```

The template for this graph uses a BLOCKPLOT statement in an INNERMARGIN block to create the inset. The INNERMARGIN block reserves sufficient space along the top of the graph (ALIGN=TOP) for the BLOCKPLOT statement output. The BLOCKPLOT statement specifies 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 column. By default, the values of the CLASS= variable appear as row labels external to the block plot.

In order to blend the block plot with the graph, the block plot label and value text attributes are set to GraphDataText. The outline color is set to GraphAxisLines:Color in order to match the axis line and graph wall outline color.

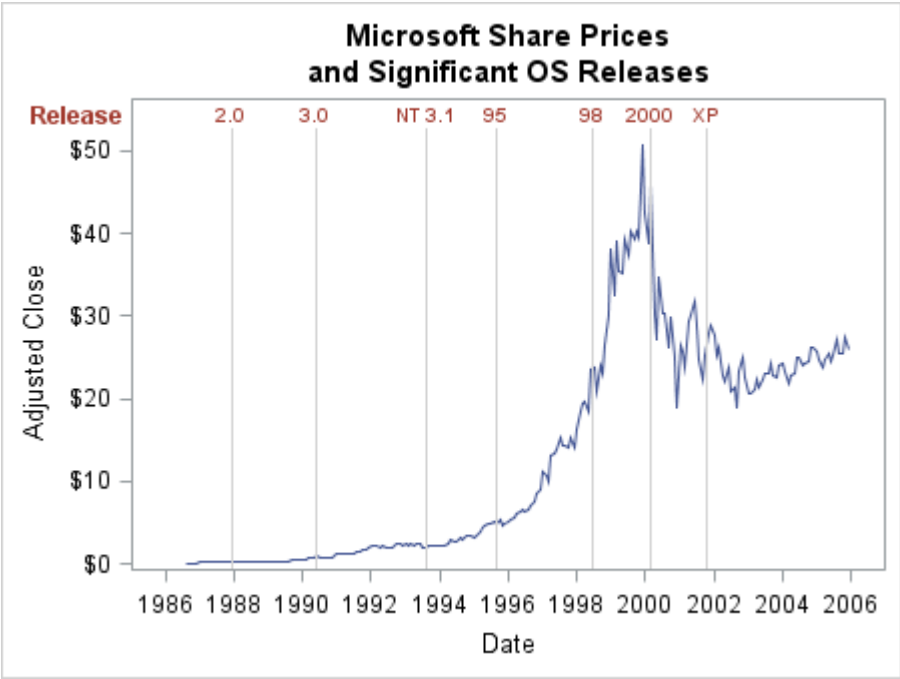
Creating an Axis-Aligned Inset with an Axis Table

Displaying Axis-Aligned Values with an Axis Table

You can use an axis table to display axis-aligned values along an axis. The following example generates a graph similar to that shown in Figure 21.3 on page 410. In this example, an axis table is used instead of a block plot. The same data (Example Code 21.1 on page 410) is used to generate the plot.

Here is the output for this example.

Figure 21.6 *An Axis-Aligned Inset Created Using an Axis Table*



Here is the code that defines the template for this example and then renders the graph.

```
/* Create the template for the graph. */
proc template;
  define statgraph axistable1;
    begingraph;
      entrytitle "Microsoft Share Prices";
      entrytitle "and Significant OS Releases";
      layout overlay;
        seriesplot x=date y=adjClose / lineattrs=GraphData1;
        referenceline x=eval(ifn(release ne " ",date,.)) /
          lineattrs=(color=lightgray);
```

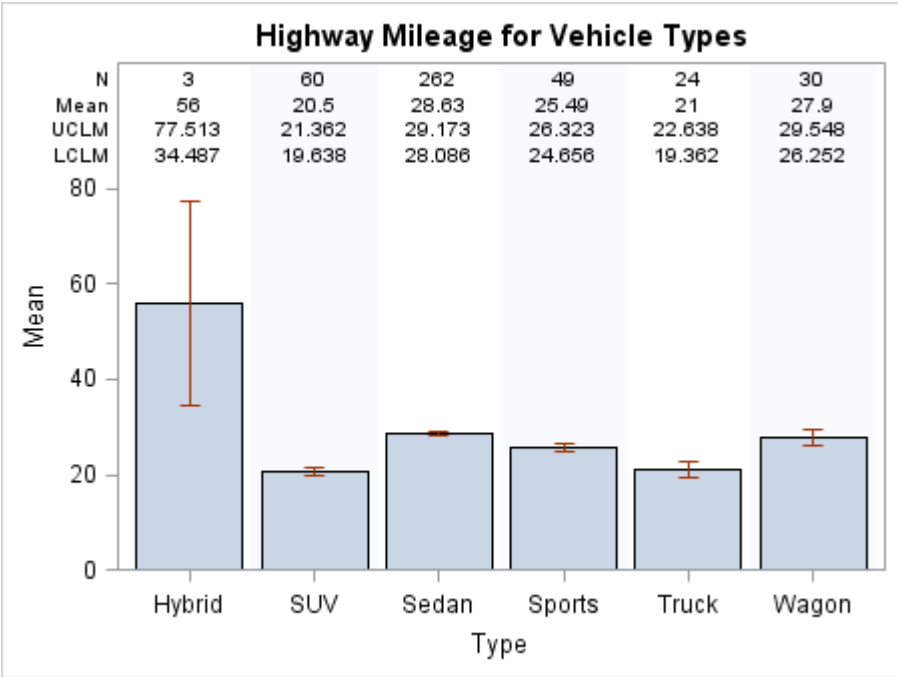
```
            innermargin / align=top opaque=true;
              axistable x=date value=release /
                display=(label)
                valueattrs=(color=GraphData2:contrastcolor weight=bold)
                labelattrs=(color=GraphData2:contrastcolor weight=bold);
            endinnermargin;
          endlayout;
        endgraph;
      end;
    run;

    /* Render the graph. */
    proc sgrender data=events template=axistable1;
    run;
```

The code defines a template named AXISTABLE1, which is used to create this graph. In this template, an inner margin is placed at the top of the graph. The inner margin block contains an AXISTABLE statement, which displays the release for each event date. The inner margin automatically reserves space at the top of the graph for the axis table. A REFERENCELINE statement draws a light-gray reference line at each release date. The IFN function is used to draw a reference line only at dates where the Release column value is a nonmissing value. Because the OPAQUE=TRUE option is in the INNERMARGIN statement, the reference lines do not pass through the axis table values. The SERIESPLOT statement draws the plot. For information about the AXISTABLE statement, see "AXISTABLE" in *SAS Graph Template Language: Reference*

Creating an Axis-Aligned Table of Values with an Axis Table

You can use an axis table to display an axis-aligned table of values along an axis. The following example generates a graph similar to that shown in Figure 21.5 on page 414. The example uses an axis table instead of a block plot. The same data (Example Code 21.2 on page 414) is used to generate the plot. Here is the output for this example.

Figure 21.7 *Axis-Aligned Multi-row Table Created Using an Axis Table*



For this example, you do not need to transpose the summarized data in the Stats data set. You can use the STATS data set as generated. See Example Code 21.2 on page 414.

Here is the code that defines the template for this example and then renders the graph.

```
/* Create the template for the graph. */
proc template;
  define statgraph axistable2;
    begingraph;
      entrytitle "Highway Mileage for Vehicle Types";
      layout overlay /
        xaxisopts=(discreteopts=(colorbands=even
          colorbandsattrs=(transparency=0.8)))
        yaxisopts=(offsetmax=0.25);
        barchartparm category=type response=mean /
           errorlower=lclm errorupper=uclm;
        axistable x=type value=n / position=0.95
           display=(label) labelattrs=GraphDataText;
        axistable x=type value=mean / position=0.90
           display=(label) labelattrs=GraphDataText;
        axistable x=type value=uclm / position=0.85
           display=(label) labelattrs=GraphDataText;
        axistable x=type value=lclm / position=0.80
           display=(label) labelattrs=GraphDataText;
      endlayout;
    endgraph;
  end;
run;

/* Render the graph. */
proc sgrender data=stats template=axistable2;
```

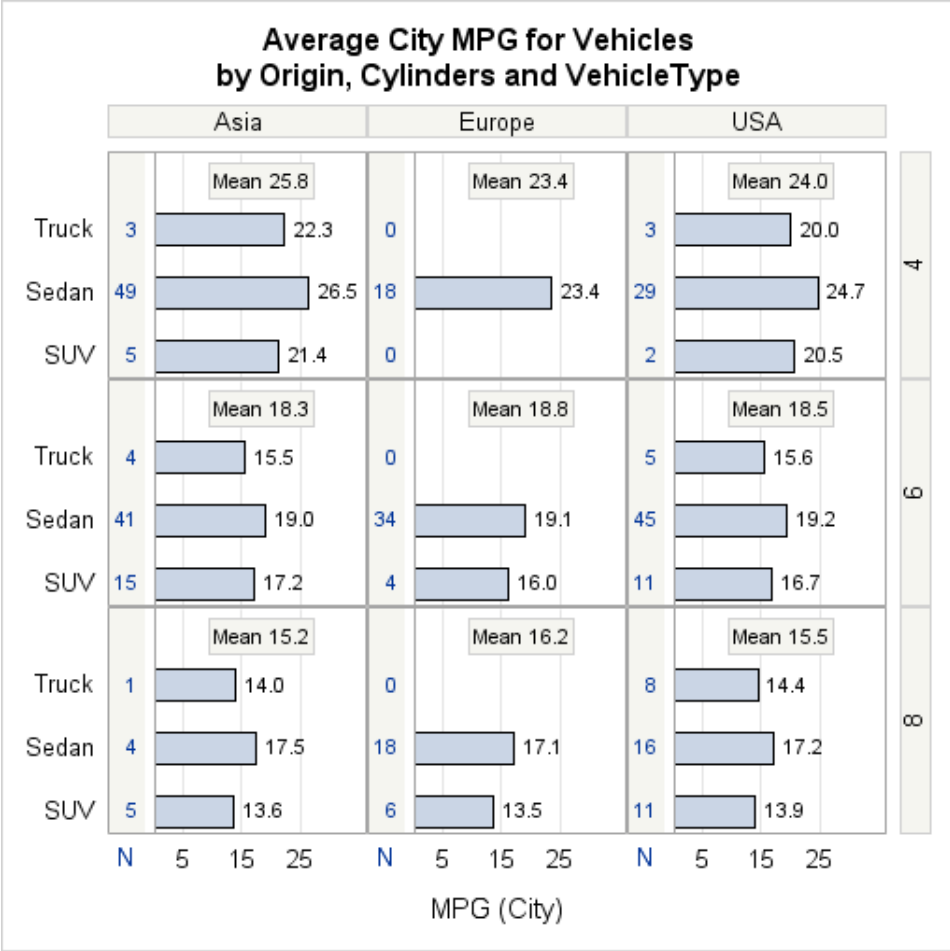
```
run;
```

In this template, the AXISTABLE statements are not placed in an inner margin block. The OFFSETMAX= option is used with the Y axis in order to reserve space at the top of the graph for the axis table. One AXISTABLE statement displays a row of statistical values in the table for each vehicle type. Four AXISTABLE statements are required to build the table in this example. The POSITION= option is used in each AXISTABLE statement to position each table row vertically. A simpler approach is to place one AXISTABLE statement in an inner margin block and include the CLASS=STATNAME option in the AXISTABLE statement. In that case, the INNERMARGIN statement automatically reserves space for the table, and the CLASS= option automatically generates one row for each statistic. However, the POSITION= option enables greater control over the vertical spacing of the table rows.

Note: The POSITION= option is ignored when the AXISTABLE statement is placed in an INNERMARGIN block.

Creating an Axis-Aligned Inset in a Classification Panel

Starting with SAS 9.4M1, you can include an AXISTABLE or BLOCKPLOT statement in a LAYOUT PROTOTYPE INNERMARGIN block. Including one of these statements enables you to inset axis-aligned information in each cell of a classification panel. For example, consider Figure 21.2 on page 405. If you want to show the number of observations for each category, you can add an axis table of observation counts along the row axis in each cell, as shown in the following figure.



The axis table is labeled N in order to identify the values as observations counts. Here is the SAS code that generated this panel.

```
/* Generate the data for the panel */
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(drop=_freq_) mean=Mean n=N / noinherit;
  types cylinders*origin cylinders*origin*type;
run;

/* Define the template for the panel */
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=(mean) insetopts=(autoalign=(top))
      insetopts=(border=true datascheme=matched
        opaque=true backgroundcolor=GraphAltBlock:color)
```

```
        rowaxisopts=(offsetmax=0.35 offsetmin=0.1
          display=(tickvalues))
        columnaxisopts=(label="MPG (City)"
          display=(label tickvalues)
          griddisplay=on offsetmin=0 offsetmax=.15
          linearopts=(tickvaluepriority=true
            tickvaluesequence=(start=5 end=30 increment=5)));
        layout prototype;
          barchart x=type y=mean / name="bar" barwidth=.5
            barlabel=true orient=horizontal;
          innermargin / backgroundcolor=GraphAltBlock:color
            opaque=true align=left;
            axistable y=type value=N / display=(label)
              valueattrs=GraphDataDefault
              labelattrs=GraphDataDefault;
          endinnermargin;
        endlayout;
      endlayout;
    endgraph;
  end;
  run;

  /* Generate the graph */
  proc sgrender data=mileage template=panel;
    format Mean 4.1;
  run;
```

The INNERMARGIN block in a PROTOTYPE layout can display an axis table in the top, right, bottom, or left margin in each cell. It can display a block plot in the top or bottom margin in each cell. The AXISTABLE statement creates a table of axis-aligned values along a column or row axis. In this example, column N in the plot data contains the observation count for each category. The AXISTABLE statement displays the N-column values along the row axis and aligns each value with its category midpoint. For a column axis, you can also use a BOXPLOT statement in an INNERMARGIN block to display axis-aligned values along the column axis.

For more information about the INNTERMARGIN block, and the AXISTABLE and BOXPLOT statements, see *SAS Graph Template Language: Reference*.

Creating an Axis-Aligned Inset with the GTL Annotation Facility

You can use the GTL annotation facility to construct an axis-aligned inset. By using the GTL annotation facility, you can position text and other graphics elements at specific coordinates in the DATAVALUE drawing space along an axis. Although more complex, the annotation facility provides a great deal of flexibility for creating custom insets. For an example of using the GTL annotation facility to create axis-aligned insets, see "Example 2: Displaying Subsets of Annotations in Axis-Aligned Insets" on page 457. For information about the GTL annotation facility, see "About the GTL Annotation Facility" in *SAS Graph Template Language: Reference*.

PART 6

Adding Custom Graphical Elements

22

Adding Code-Driven Graphics Elements to Your Graph

Overview: Adding Non-Data-Driven Graphics Elements to a Graph

The Graph Template Language (GTL) provides draw statements that enable you to draw additional graphics elements on your graphs that are independent of the graph data. The types of elements that you can draw include the following:

- text
- arrows and lines
- geometric shapes, such as ovals, rectangles, polylines, and polygons
- images

You can use the draw statements to add annotations that describe the non-data aspects of your graph. You can also use them to create custom features on your graph, such as a broken axis, that are difficult to create by other means. You can draw the elements in one of the following drawing spaces on your graph: the data area, the wall area, the layout area, or the graph area. For more information about

the drawing spaces, see “Selecting the Drawing Space and Units” on page 426. You can specify the location of each element using Cartesian coordinates, and you can specify the axis to which the coordinates are scaled. You can also specify other attributes of the graphics element, such as line color and pattern, text font, and so on.

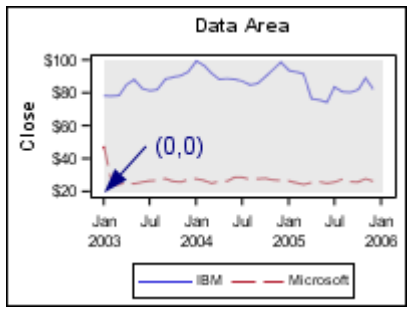
In addition to the drawing space and attributes, you can also choose the layer on which your graphics elements are drawn. Two layers are available that are relative to the graph: front and back. The front layer appears in front of the graph. The back layer appears behind the graph wall for graphs that have axes or behind the graph for graphs that do not have axes. By default, the graphics elements are drawn on the front layer. You can use the LAYER=BACK option on your draw statement to draw the elements on the back layer.

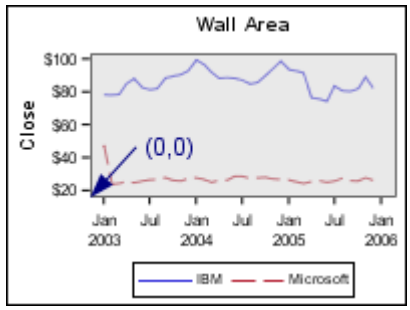
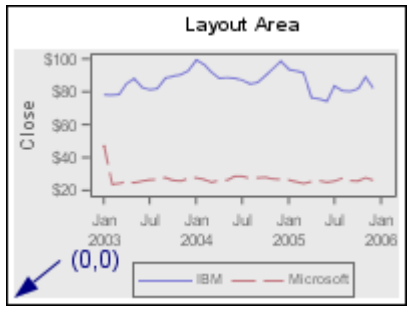
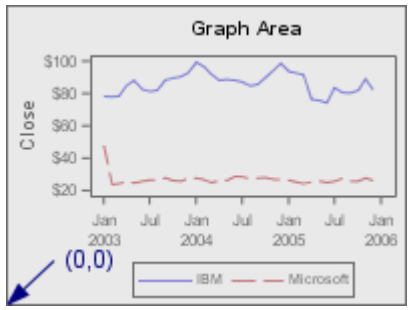
For plots that have axes, if the graph wall area is filled (default), graphics elements that are drawn on the back layer are obscured by the wall fill. To make the elements visible in that case, include the WALLDISPLAY=NONE or WALLDISPLAY=(OUTLINE) option in your layout statement. You can also use the TRANSPARENCY= option in the plot statement to adjust the transparency of the graph in order to allow the graphics elements underneath to show through.

For information about using the draw statements to add graphics elements to your graph, see “Adding Graphics Elements to Your Graph” on page 429.

Selecting the Drawing Space and Units

When you draw graphics elements, you can specify the drawing space and the drawing units in your draw statements. The drawing space is the area of the graph in which the elements are drawn, which can be data, wall, layout, or graph. The drawing areas are described in the following table.

Drawing Space	Description	Example
Data	The area of the graph in which the data is displayed. The data area is indicated by the shaded area in the figure on the right. The origin of the drawing space X and Y coordinates, (0,0), is in the lower left corner as shown. **Note:** The data area does not apply to graphs that do not have axes, such as pie charts, that must be drawn in a REGION layout.	

Drawing Space	Description	Example
Wall	The area of the graph that is bound by the X and Y axes, including the secondary axes, if it is used. **Note:** The wall area does not apply to graphs that do not have axes, such as pie charts, that must be drawn in a REGION layout.	
Layout	The entire area of the layout container that is the immediate parent container of the draw statement. The figure on the right shows the case where a LAYOUT OVERLAY is the parent.	
Graph	The area in which the entire graph is displayed. **Note:** In a multi-cell layout such as GRIDDED or LATTICE, the GRAPHPERCENT and GRAPHPIXEL units span the entire graph, which includes all of the cells in the layout.	

The drawing space and units are specified in a single value that is in the following format:

```
<DrawingSpace><Units>
```

DrawingSpace can be DATA, WALL, LAYOUT, or GRAPH. For the WALL, LAYOUT, and GRAPH areas, *Units* can be PIXEL or PERCENT. For the DATA drawing space, *Units* can be PIXEL, PERCENT, or VALUE. PIXEL indicates that the coordinates are expressed in pixels in the drawing space. PERCENT indicates that the coordinates are expressed as a percentage of the drawing space. For example, DATAPERCENT indicates that the coordinates are expressed as a percentage of the DATA drawing space.

For the DATA drawing space, VALUE indicates that the coordinates are expressed as values along the axis. When you specify the DATA drawing space, you can use the XAXIS=, YAXIS=, X1AXIS=, Y1AXIS=, X2AXIS=, and Y2AXIS= options in the draw statement, as applicable, to specify the axis to which the coordinates are scaled.

Note: A draw statement is discarded if the XAXIS=, YAXIS=, X1AXIS=, Y1AXIS=, X2AXIS=, and Y2AXIS= options specify an axis that does not exist in the plot. It is also discarded if the DRAWSPACE=, XSPACE=, YSPACE=, X1SPACE=, Y1SPACE=, X2SPACE=, and Y2SPACE= options specify a drawing space that is not valid for the draw statement's layout container.

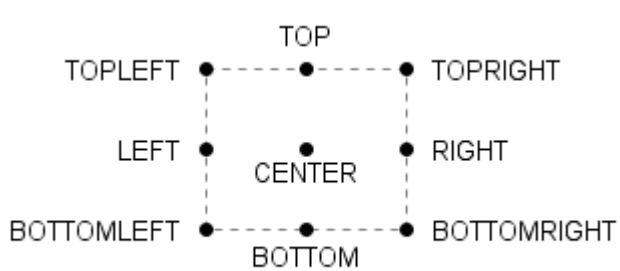
You can specify a common drawing space and units for all of the X and Y coordinates. You can also specify a different drawing space and units for the coordinates individually. To specify a common space and units, use the DRAWSPACE= option on each draw statement or in the BEGINGRAPH statement. When you use the DRAWSPACE= option on a draw statement, the space and units that you specify are applied to the coordinates for that statement only. This includes the X and Y coordinates or the X1, Y1, X2, and Y2 coordinates. When you use the DRAWSPACE= option in the BEGINGRAPH statement, the space and units that you specify are applied to all of the draw statements within the BEGINGRAPH/ENDGRAPH block.

To specify the drawing space and units for the X and Y coordinates individually, use the XSPACE=, YSPACE=, X1SPACE=, Y1SPACE=, X2SPACE=, and Y2SPACE= options, as applicable, on each draw statement.

Note: The XSPACE=, YSPACE=, X1SPACE=, Y1SPACE=, X2SPACE, and Y2SPACE= options override the DRAWSPACE= option.

How the Graphics Elements Are Anchored

When you specify the X and Y coordinates for a graphics element, the element is drawn from an anchor point that is placed in the drawing area at the X and Y coordinates that you specify. For lines and arrows, the anchor point is the starting point of the line or arrow, which is specified with the X1 and Y1 options on the draw statement. For elements that have height and width, the anchor point can be one of the points shown in the following figure.



By default, the anchor point is CENTER. You can use the ANCHOR= option on the draw statements to change the anchor point of your graphics elements.

Note: When you select the X and Y coordinates and the anchor point, make sure that when the graphics element is drawn, it does not extend beyond the boundaries of the drawing area. Any part of the element that is outside of the drawing area is clipped.

Adding Graphics Elements to Your Graph

The Draw Statements

The following table lists the GTL draw statement or statement block that you can use to draw each type of element.

To Draw this Type of Graphics Element	Use this GTL Statement or Block
Text	DRAWTEXT
An arrow	DRAWARROW
A line	DRAWLINE
An oval or circle	DRAWOVAL
A square or rectangle	DRAWRECTANGLE
A polyline	DRAW statements within a BEGINPOLYLINE block
A polygon	DRAW statements within a BEGINPOLYGON block
An image	DRAWIMAGE

This guide provides an overview of how to use these statements. For detailed information about these statements, see *SAS Graph Template Language: Reference*.

Adding Text

Use a DRAWTEXT statement to add a text element to your graph. The basic syntax is as follows:

```
DRAWTEXT <TEXTATTRS=(text-options)> "text" / X=x Y=y <options>
```

In your DRAWTEXT statement, specify in "*text*" the text that you want to appear in your text element. You must enclose the text in quotation marks. If you want to change any attributes of the text such as the font family, font size, or font color, include the necessary options in the TEXTATTRS= option in the DRAWTEXT statement.

Note: The TEXTATTRS= option must be placed before "*text*" in the DRAWTEXT statement.

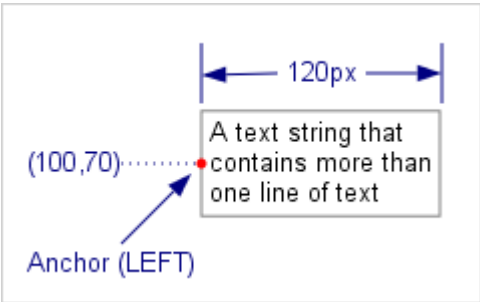
Use the X= and Y= options in the DRAWTEXT statement to specify the coordinates where you want to place the text. By default, the coordinate units are GRAPHPERCENT, and the text element is anchored on its center point at the specified coordinates. In *options* , you can include the DRAWSPACE= option or the XSPACE= and YSPACE= options to specify different units, and the ANCHOR= option to change the anchor point.

If you want text to wrap within a specified area, include the WIDTH= option in *options* . The WIDTH= option specifies the maximum width of the text area in PERCENT units by default. You can include the WIDTHUNIT= option in *options* to specify PIXEL or DATA units instead. If you want to add a border around your text, include the BORDER=TRUE and the BORDERATTRS= options in *options* .

Here is an example of a DRAWTEXT statement that adds a 120-pixel-wide block of text with a gray border.

```
drawtext "A text string that contains more than one line of text" /
  x=100 y=70 drawspace=graphpixel
  width=120 widthunit=pixel
  anchor=left
  border=true borderattrs=(color=gray pattern=1);
```

The X= and Y= options specifies the coordinates of the anchor point as (100, 70) in GRAPHPIXEL units. The ANCHOR= option specifies that the text is to be anchored at the LEFT anchor point as shown in the following figure. The WIDTH= and WIDTHUNIT= options specify the maximum width of the text block as 120 pixels.



Adding Arrows and Lines

Use a DRAWARROW or DRAWLINE statement to draw an arrow or line on your graph. The basic syntax is as follows:

```
DRAWARROW X1=x1 Y1=y1 X2=x2 Y2=y2 / <options>
```

The syntax for the DRAWLINE statement is similar.

The X1=, Y1=, X2=, and Y2= options on the DRAWARROW and DRAWLINE statements specify the coordinates for each endpoint of the arrow or line. By default, the coordinate units are GRAPHPERCENT. You can include the DRAWSPACE= option, or the X1SPACE=, Y1SPACE=, X2SPACE=, and Y2SPACE= options in *options* to specify different units. If you specify DATAVALUE as the units and you want to scale your arrow or line to the secondary axis, you must also include the X1AXIS=X2, Y1AXIS=Y2, X2AXIS=X2, and Y2AXIS=Y2 options.

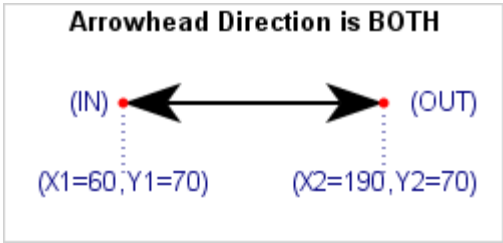
For both arrows and lines, include the LINEATTRS= option in *options* to specify the line pattern, thickness, and color. For arrows, open arrowheads that point in the outward direction are the default. To change the arrowhead shape, include the ARROWHEADSHAPE= option and specify CLOSED, FILLED, or BARBED. You can also include the ARROWHEADSCALE= option to scale the arrowhead based on the thickness of the arrow line. The scale factor is 1 by default. You can scale the arrowhead from a minimum of 0.5 to a maximum of 2.

To change the arrowhead direction, include the ARROWHEADDIRECTION= option in *options* and specify IN or BOTH. The IN direction positions the arrowhead on the (X1, Y1) endpoint and points it inward toward the (X1, Y1) endpoint. The BOTH direction includes both IN and OUT arrowheads forming a two-way arrow.

Here is an example of a DRAWARROW statement that adds a two-way dashed arrow.

```
drawarrow x1=60 y1=70 x2=190 y2=70 / drawspace=graphpixel
  lineattrs=(color=black thickness=2px)
  arrowheadshape=barbed arrowheadscale=2 arrowheaddirection=both;
```

The arrow is drawn from endpoint (60, 70) to endpoint (190, 70) in GRAPHPIXEL units as shown in the following figure.



The LINEATTRS= option specifies a black line that is two pixels wide. The ARROWHEADSHAPE= option specifies a barbed arrowhead, and the ARROWHEADSCALE= option specifies a scale factor of 2 (maximum size). The ARROWHEADDIRECTION= option specifies a two-way arrow.

To draw a line, in your DRAWLINE statement, use the X1=, Y1=, X2=, and Y2= options to specify the location of the endpoints. Include the LINEATTRS= option in *options* to specify the line pattern, color, and thickness. Here is the previous example modified to draw a dashed line instead of a dashed arrow at the same coordinates.

```
drawline x1=60 y1=70 x2=190 y2=70 / drawspace=graphpixel
  lineattrs=(pattern=3 thickness=1px);
```

Adding Geometric Shapes

Ovals

Use a DRAWOVAL statement to add ovals to your graph. The basic syntax is as follows:

```
DRAWOVAL X=x Y=y WIDTH=width HEIGHT=height / <options>
```

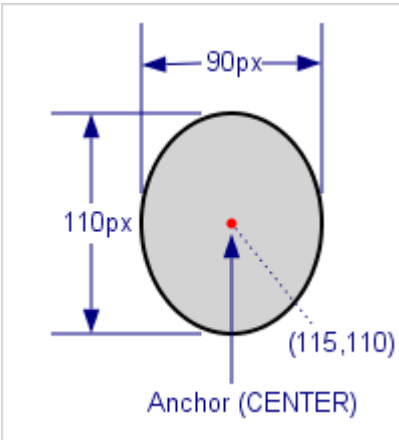
The X= and Y= options in the DRAWOVAL statement specify the coordinates of the oval anchor point. By default, the coordinate units are GRAPHPERCENT, and the oval is anchored on its center point. You can include the DRAWSPACE= option, or the XSPACE= and YSPACE= options in *options* to specify different units. If you specify DATAVALUE as the coordinate units and you want to scale the oval to the secondary axis, you must also include the XAXIS=X2 and YAXIS=Y2 options. You can include the ANCHOR= option to change the anchor point.

The WIDTH= and HEIGHT= options in the DRAWOVAL statement specify the dimensions of the oval in PERCENT units by default. You can include the WIDTHUNIT= and HEIGHTUNIT= options in *options* to specify PIXEL or DATA units instead.

You can change other attributes of the oval, such as the fill color, the outline color, and the outline pattern. Include the FILLATTRS= option in *options* to change the fill color, and include the OUTLINEATTRS= option to change the outline color and pattern.

Here is an example of a DRAWOVAL statement that adds a 90 pixel wide by 110 pixel high oval at coordinates (115, 110) in GRAPHPIXEL units.

```
drawoval x=115 y=110 width=90 height=110 / drawspace=graphpixel
  widthunit=pixel heightunit=pixel
  anchor=center
  display=all
  fillattrs=(color=lightgray)
  outlineattrs=(color=black thickness=2px);
```



The ANCHOR= option sets the oval anchor point to CENTER, which centers the oval at coordinates (115, 110). The DISPLAY=ALL option displays the outline and fill. The FILLATTRS= option specifies a light gray fill, and the OUTLINEATTRS= option specifies a black outline that is two pixels wide.

Rectangles

Use a DRAWRECTANGLE statement to add rectangles to your graph. The basic syntax is as follows:

```
DRAWRECTANGLE X=x Y=y WIDTH=width HEIGHT=height / <options>
```

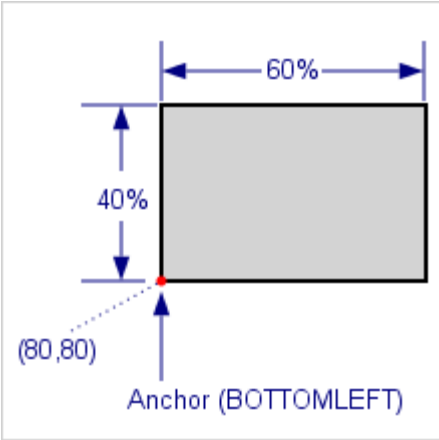
The X= and Y= options in the DRAWRECTANGLE statement specify the coordinates of the anchor point of the rectangle. By default, the coordinate units are GRAPHPERCENT, and the rectangle is anchored on its center point. You can include the DRAWSPACE= option, or the XSPACE= and YSPACE= options in *options* to specify different units. If you specify DATAVALUE as the coordinate units and you want to scale the rectangle to the secondary axis, you must also include the XAXIS=X2 and YAXIS=Y2 options. You can include the ANCHOR= option to change the anchor point.

The WIDTH= and HEIGHT= options in the DRAWRECTANGLE statement specify the dimensions of the rectangle in PERCENT units by default. You can include the WIDTHUNIT= and HEIGHTUNIT= options in *options* to specify PIXEL or DATA units instead.

You can change other attributes of the rectangle, such as the fill color, the outline color, and the outline pattern. Include the FILLATTRS= option in *options* to change the fill color, and include the OUTLINEATTRS= option to change the outline color and pattern.

Here is an example of a DRAWRECTANGLE statement that adds a 60% wide by 40% high rectangle. In this example, percent refers to the percentage of the drawing area, at coordinates (80,80) in GRAPHPIXEL units.

```
drawrectangle x=80 y=80 width=60 height=40 / drawspace=graphpixel
  widthunit=percent heightunit=percent
  anchor=bottomleft
  display=all
  fillattrs=(color=lightgray)
  outlineattrs=(color=black thickness=2px);
```



The ANCHOR= option specifies the rectangle anchor point as BOTTOMLEFT, which positions the lower left corner at coordinates (80, 80). The DISPLAY=ALL option displays the outline and fill. The FILLATTRS= option specifies a light gray fill, and the OUTLINEATTRS= option specifies a black outline that is two pixels wide.

Polylines

Use a BEGINPOLYLINE block to add a polyline to your graph. The basic syntax is as follows:

```
BEGINPOLYLINE X=origin-x Y=origin-y / <options>;
  DRAW X=x1 Y=y1;
  DRAW X=x2 Y=y2;
  ...more DRAW statements...
  DRAW X=Xn Y=Yn;
ENDPOLYLINE;
```

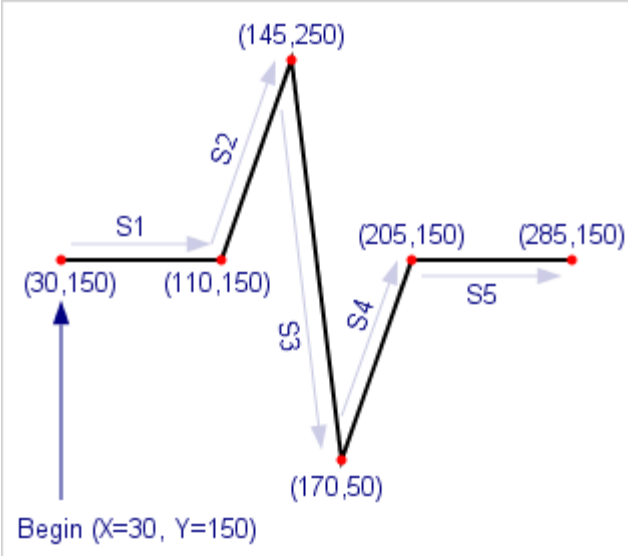
Use a BEGINPOLYLINE statement to open the block. In the BEGINPOLYLINE statement, use the X= and Y= options to specify the coordinates of the beginning point of the polyline. By default, the coordinate units are GRAPHPERCENT. You can include the DRAWSPACE= option, or the XSPACE= and YSPACE= options in *options* to specify different units. If you specify the units as DATAVALUE and you want to scale the polygon to the secondary axis, you must also include the XAXIS=X2 and YAXIS=Y2 options. To change the line color, pattern, or thickness, include the LINEATTRS= option.

Following the BEGINPOLYLINE statement are the individual DRAW statements. Each DRAW statement draws a straight line from the previous point to the endpoint that is specified in the DRAW statement's X= and Y= options. For the first DRAW statement, the previous point is the starting point that is specified in the BEGINPOLYLINE statement. For subsequent DRAW statements, the previous point is the endpoint that is specified in the previous DRAW statement. Add a DRAW statement for each segment in your polyline. You can add as many segments as you need. After the DRAW statements, add an ENDPOLYLINE statement to close the block.

Here is an example that draws a five-segment polyline beginning at the coordinates (30, 150). The coordinates are specified in GRAPHPIXEL units.

```
beginpolyline x=30 y=150 / xspace=graphpixel yspace=graphpixel
  lineattrs=(color=black thickness=2px);
  draw x=110 y=150; /* Draw S1 */
  draw x=145 y=250; /* Draw S2 */
  draw x=170 y=50;  /* Draw S3 */
  draw x=205 y=150; /* Draw S4 */
  draw x=285 y=150; /* Draw S5 */
endpolyline;
```

The following figure shows how this polyline is drawn.



The X= and Y= options in the BEGINPOLYLINE statement specify the starting point of the polyline, (30, 150), in GRAPHPIXEL units. The LINEATTRS= option specifies a black line that is two pixels wide. The first DRAW statement draws segment 1 (S1) between the starting point (30, 150) and endpoint (110, 150). The second DRAW statement draws S2 between the endpoint (110, 150) of the first DRAW statement to endpoint (145, 250). This pattern continues for the remaining DRAW statements in the block. The ENDPOLYLINE statement closes the block.

Polygons

Use a BEGINPOLYGON block to add a polygon to your graph. The basic syntax is as follows:

```
BEGINPOLYGON X=origin-x Y=origin-y / <options>;
  DRAW X=x1 Y=y1;
  DRAW X=x2 Y=y2;
  ...more DRAW statements...
  DRAW X=origin-x Y=origin-y;
ENDPOLYGON;
```

Use a BEGINPOLYGON statement to open the block. In the BEGINPOLYGON statement, use the X= and Y= options to specify the coordinates of the beginning point of the polygon. By default, the coordinate units are GRAPHPERCENT. You can include the DRAWSPACE= option, or the XSPACE= and YSPACE= options in *options* to specify different units. If you specify the units as DATAVALUE and you want to scale the polygon to the secondary axis, you must also include the XAXIS=X2 and YAXIS=Y2 options. To change the line color, pattern, or thickness, include the LINEATTRS= option.

Following the BEGINPOLYGON statement are the DRAW statements. Each DRAW statement draws a straight line from the previous point to the endpoint that is specified in the DRAW statement's X= and Y= options. For the first DRAW statement, the previous point is the starting point that is specified in the BEGINPOLYGON statement. For subsequent DRAW statements, the previous point is the endpoint that is specified in the previous DRAW statement. Add a DRAW statement for each side of your polygon. You can add as many sides as you need.

The last DRAW statement typically ends at the starting point of the polygon in order to close the polygon.

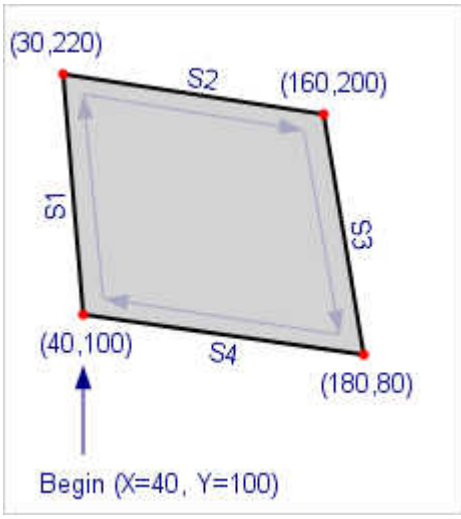
Note: If the last DRAW statement does not end at the starting point of the polygon, a line is drawn automatically that connects the endpoint of the last DRAW statement to the starting point in order to close the polygon.

After the DRAW statements, add an ENDPOLYGON statement to close the block.

Here is an example that draws a four-sided polygon that begins at the coordinates (40, 100). All of the coordinates are specified in GRAPHPIXEL units.

```
beginpolygon x=40 y=100 / xspace=graphpixel yspace=graphpixel
  display=all fillattrs=(color=lightgray)
  outlineattrs=(thickness=2px color=black);
  draw x=30  y=220; /* Draw S1 */
  draw x=160 y=200; /* Draw S2 */
  draw x=180 y=80;  /* Draw S3 */
  draw x=40  y=100; /* Draw S4 */
endpolygon;
```

The following figure shows how the polygon is drawn.



The BEGINPOLYGON statement X= and Y= options specify the starting point of the polygon, (40, 100). The DISPLAY=ALL option displays the outline and fill. The FILLATTRS= option specifies a light gray fill while the OUTLINEATTRS= option specifies a black outline that is two pixels wide. The first DRAW statement draws side 1 (S1) between the starting point (40, 100) and endpoint (30, 220). The second DRAW statement draws S2 between endpoint (30, 220) of the first DRAW statement to endpoint (160, 200). This pattern continues for the remaining DRAW statements. The last DRAW statement connects endpoint (180, 80) to the starting point, (40, 100), which closes the polygon. The ENDPOLYGON statement closes the block.

TIP In this example, you can omit the DRAW statement for the last segment (S4). In that case, the fourth segment is drawn automatically to close the shape.

Adding Images

Use a DRAWIMAGE statement to place a JPG or PNG image on your graph.

Note: The DRAWIMAGE statement supports the JPG and PNG image formats only.

The basic syntax is as follows:

```
DRAWIMAGE "image-file.ext" / X=x Y=x <options>
```

The image file is specified as *image-file.ext*, which is an absolute or relative path to the image file on the file system. The path must be enclosed in quotation marks, and it must include the image filename and file extension, such as image.jpg or image.png.

Note: The image file must be accessible on the file system. URL access is not supported.

The X= and Y= options in the DRAWIMAGE statement specify the coordinates of the image anchor point. By default, the coordinate units are GRAPHPERCENT, and the image is anchored on its center point. You can include the DRAWSPACE= option, or the XSPACE= and YSPACE= options in *options* to specify different units. If you specify DATAVALUE as the units and you want to scale the image to the secondary axis, you must also include the XAXIS=X2 and YAXIS=Y2 options. You can include the ANCHOR= option to change the anchor point.

You can use the HEIGHT= and WIDTH= options in the DRAWIMAGE statement to create a bounding box in which the image is drawn. The default units for the height and width are PERCENT. You can include the SIZEUNIT= option in *options* to specify PIXEL or DATA units instead. You can also include the SCALE= option to specify how the image is scaled within the bounding box. By default, the image is scaled to fit the box. You can also fit the image by height or width, or you can tile the image. If you want to draw a border around your image, you can include the BORDER=Y and BORDERATTRS= options.

Here is an example that adds the SAS logo at coordinates (70, 60) in GRAPHPIXEL units. The image is anchored at the bottom left corner, and a black border is drawn around the image.

```
drawimage ".\images\saslogo.png" / x=70 y=60 drawspace=graphpixel
  anchor=bottomleft
  border=true borderattrs=(thickness=1px);
```

The following figure shows how the image is placed on the graph.

23

Adding Data-Driven Annotations to Your Graph

Overview: Adding Data-Driven Annotations to a Graph

The Graph Template Language (GTL) provides an annotation facility that enables you to add graphics elements to a graph by using ODS Statistical Graphics (SG) annotation instructions that are stored in a SAS data set. It is similar to the SAS/GRAPH Annotate facility. Unlike adding graphics elements using the GTL draw statements, the annotation facility enables you to separate the annotation graphics instructions from your template code. To modify your annotations, you can specify a different annotation data set. You do not have to modify your template code.

Like the GTL draw statements, you can use the annotation facility to add the following graphics elements to your graphs:

- text
- arrows and lines
- geometric shapes, such as ovals, rectangles, polylines, and polygons
- images

You can draw the annotations in one of the following drawing spaces on your graph: the data area, the wall area, the layout area, or the graph area. For more information about the drawing spaces, see "Selecting the Drawing Space and Units" on page 426. You can specify the location of each annotation using Cartesian coordinates, and you can specify the axis to which the coordinates are scaled. You can also specify other attributes of the annotations, such as line color and pattern, text font, and so on.

In addition to the drawing space and attributes, you can also choose the layer on which your annotations are drawn. Two layers are available that are relative to the graph: front and back. The front layer appears on top of the graph. The back layer appears behind the graph, behind the background areas such as the layout or legend background. By default, the annotations are drawn on the front layer. You can specify the LAYER=BACK option for your annotations to draw them on the back layer.

For plots that have axes, if the plot wall area is filled (default) or the layout background is opaque (OPAQUE=TRUE), annotations that are drawn on the back layer are obscured by the plot wall fill or by the layout background. To make the annotations visible in either case, include the WALLDISPLAY=NONE or WALLDISPLAY=(OUTLINE) option in your layout statement, or specify OPAQUE=FALSE for the layout. You can also use the TRANSPARENCY= option in the plot statement to adjust the transparency of the plot in order to allow the graphics elements underneath to show through.

To use the GTL annotation facility, you must complete the following tasks:

- Create a SAS data set that contains your SG annotation instructions. You must store the annotation instructions in a SAS data set. Annotation instructions stored in a CAS in-memory table are not supported.
- Include at least one ANNOTATE statement in your graph template.
- Include the SGANNO= option in your SGRENDER statement to specify the SG annotation data set by name.

For more information about these requirements, see "About the GTL Annotation Facility" in *SAS Graph Template Language: Reference*.

Creating an SG Annotation Data Set

Adding Observations for Text Annotations

To add a text annotation to your SG annotation data set:

1 Add a TEXT function observation that specifies the following:
   - TEXT in the Function column
   - the annotation text in the Label column

     Note: You can use the ODS escape sequence to include superscripts, subscripts, and Unicode characters in the Label column as you would in quoted strings for a text command. For more information, see "Using Text Commands" in *SAS Graph Template Language: Reference*.

   - the coordinates of the text location if it is not the default location (center of the graph area). Specify numeric coordinates in the X1 and Y1 columns or character coordinates in the Xc1 and Yc1 columns.
   - the text anchor point in the Anchor column if it is not the default (CENTER).
   - the width of the text box in the Width column if it is not the default (determined by the system).
   - other columns that specify other text attributes such as font, size, color, and so on. For more information, see "TEXT Function" in *SAS ODS Graphics: Procedures Guide*.

2 If you want to continue the text using different text attributes, immediately follow the TEXT function observation with one or more TEXTCONT function observations that specify the following:
   - TEXTCONT in the Function column
   - the continuing text in the Label column
   - other columns that specify the new text attributes. For more information, see "TEXTCONT Function" in *SAS ODS Graphics: Procedures Guide*.

The TEXT function observations and subsequent TEXTCONT function observations, if any, must be contiguous in the annotation data set.

Here is an example of a DATA step that creates the data for a text annotation that contains normal text, blue text, and a simple equation. The equation contains a Unicode character and a superscript character.

```
data textdata;
   length function $9 label $60;
   function="text";
   drawspace="graphpixel"; x1=100; y1=70;
   width=190; widthunit="pixel"; anchor="left";
   border="true"; linecolor="gray"; linepattern="solid";
   label="A text string that contains normal text and";
   output; /* Write normal-text observation. */
   function="textcont";
   textcolor="blue";
   label=" rich text with Unicode and superscript characters:";
   output; /* Write blue-text observation. */
   label=" A = (*ESC*){unicode pi}R(*ESC*){sup '2'}";
   output; /* Write Unicode and superscript observation. */
run;
```

Note: Starting with SAS 9.4M1, you can also use the %SGTEXT macro to generate the text annotation observations. See "Using the SG Annotation Macros to Create Your SG Annotation Data Set" on page 451.

Here is a listing of the data.

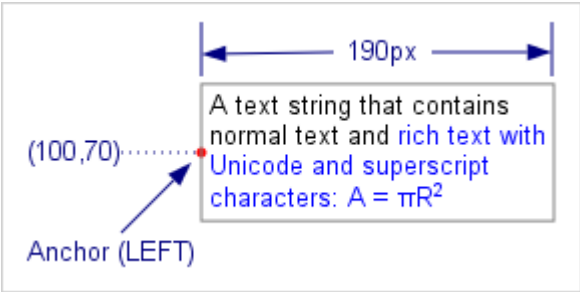
```
Data Set TEXTDATA

Obs function                          label                          drawspace
 1  text      A text string that contains normal text and           graphpixel
 2  textcont   rich text with Unicode and superscript characters:   graphpixel
 3  textcont   A = (*ESC*){unicode pi}R(*ESC*){sup '2'}             graphpixel

Obs  x1 y1 width widthunit anchor border linecolor linepattern textcolor
 1  100 70  190    pixel    left   true    gray       solid
 2  100 70  190    pixel    left   true    gray       solid      blue
 3  100 70  190    pixel    left   true    gray       solid      blue
```

In the first observation, the Function column specifies TEXT for a text annotation using the default text attributes. The DrawSpace column specifies that all coordinates are expressed in graph-area pixels. The X1 and Y1 columns specify the location as 100 on the X axis and 70 on the Y axis. The Width, WidthUnit, and Anchor columns specify a 190-pixel-wide text box that is anchored on the left side. The Border, LineColor, and LinePattern columns specify a solid gray border around the text box. The Label column specifies the normal-text portion of the text for this annotation.

In the second observation, the Function column specifies the TEXTCONT function in order to continue the text for this annotation. The TextColor column specifies blue for the text color. The Label column specifies the text for this part of the annotation text. In the third observation, the Function column remains TEXTCONT. The Label column uses the ODS escape sequence, which is (*ESC*), with {UNICODE} and {SUP} to specify the following equation:

$$A = \pi R^2$$

Here is how this annotation is placed in a graph.



Adding Observations for Arrows and Lines

To add an arrow or line to your SG annotation data set, you must specify, at a minimum, ARROW or LINE in the Function column, the coordinates of the arrow or line starting point, and the coordinates of the arrow or line ending point. For an

arrow, by default, the arrowhead shape is OPEN and the arrow direction is OUT. You can add the Shape and Direction columns to your observation to specify different values. You can add other columns to specify other attributes of the arrow such as arrowhead scale, line color, line pattern, line thickness, and so on. For information about other columns that you can add, see "ARROW Function" in *SAS ODS Graphics: Procedures Guide*.

Here is an example of a DATA step that creates the data for an arrow annotation.

```
data arrowdata;
  function="arrow";
  drawspace="graphpixel";
  x1=60; y1=70; x2=190; y2=70;
  shape="barbed"; scale=1.5; direction="both";
  linepattern="solid"; linecolor="black"; linethickness=2;
run;
```

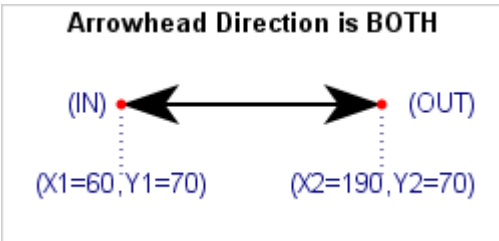
Note: Starting with SAS 9.4M1, you can also use the %SGARROW and %SGLINE macros to generate the line and arrow annotation observations. See "Using the SG Annotation Macros to Create Your SG Annotation Data Set" on page 451.

Here is a listing of the data.

```
Data Set ARROWDATA

Obs    function    drawspace     x1    y1     x2    y2    shape     scale
 1      arrow      graphpixel    60    70    190    70    barbed     1.5

Obs    direction    linepattern    linecolor    linethickness
 1       both          solid         black            2
```

In this observation, the Function column specifies ARROW for an arrow annotation. The DrawSpace column specifies that all coordinates are expressed in graph-area pixels. The X1 and Y1 columns specify the arrow starting location as 60 on the X axis and 70 on the Y axis. The X2 and Y2 columns specify the arrow ending location as 190 on the X axis and 70 on the Y axis. The Shape and Scale columns specify the shape and size of the arrowheads. The Direction column specifies the direction as BOTH. The LinePattern, LineColor, and LineThickness columns specify the arrow line as a solid black line that is two pixels wide.

Here is how this annotation is placed in a graph.



An observation for a line is similar to that of an arrow. At a minimum, you must specify LINE in the Function column, the line starting coordinates, and the line ending coordinates. You can add other columns to specify different attributes of the line, such as color, pattern, and thickness. For information about other columns that you can add, see "LINE Function" in *SAS ODS Graphics: Procedures Guide*.

Here is the previous arrow observation modified to draw a dashed line instead of an arrow.

```
data linedata;
  function="line";
  drawspace="graphpixel";
  x1=60; y1=70; x2=190; y2=70;
  linepattern="shortdash"; linecolor="black"; linethickness=2;
run;
```

Here is a listing of the data.

```
Data Set LINEDATA

Obs function drawspace  x1 y1  x2 y2 linepattern linecolor linethickness
 1    line   graphpixel 60 70 190 70  shortdash    black         2
```

Adding Observations for Polylines and Polygons

Multiple observations are required to construct a polyline or polygon. For a polyline annotation, the first observation must be the POLYLINE function, which specifies the starting point of the polyline. One or more POLYCONT function observations must immediately follow the POLYLINE function observation to specify the segments of the polyline. The POLYLINE function observation and subsequent POLYCONT function observations, if any, must be contiguous in the annotation data set. A polygon is similar to a polyline except that the segments of the polygon must form a closed shape.

To specify the starting point, you must specify, at a minimum, POLYLINE or POLYGON in the Function column, and the coordinates of the polyline or polygon starting point. You can add other columns to specify attributes of the lines, such as color, pattern, and thickness. For information about other columns that you can add, see "POLYLINE Function" in *SAS ODS Graphics: Procedures Guide* and "POLYGON Function" in *SAS ODS Graphics: Procedures Guide*.

For each segment of your polyline or polygon, you must add one observation for each segment. Each segment observation specifies POLYCONT in the Function column and the ending coordinates for that segment. For the first segment, this observation draws a line from the beginning point to the ending point that is specified in the first segment observation. For the second and subsequent segments, the observation draws a line from the ending point of the previous segment to the ending point that is specified in the current segment observation.

Here is an example of a DATA step that creates the data for a five-segment polyline annotation.

```
data polylinedata;
  function="polyline";
  drawspace="graphpixel";
  linecolor="black"; linethickness=2;
  x1=30; y1=150;  /* Begin polyline */
  output;
  function="polycont";
  x1=110; y1=150; /* Draw Segment 1 */
  output;
```

```
  x1=145; y1=250; /* Draw Segment 2 */
  output;
  x1=170; y1=50;  /* Draw Segment 3 */
  output;
  x1=205; y1=150; /* Draw Segment 4 */
  output;
  x1=285; y1=150; /* Draw Segment 5 */
  output;
run;
```

Note: Starting with SAS 9.4M1, you can also use the SG annotation macros to generate the polyline and polygon annotation observations. See "Using the SG Annotation Macros to Create Your SG Annotation Data Set" on page 451.

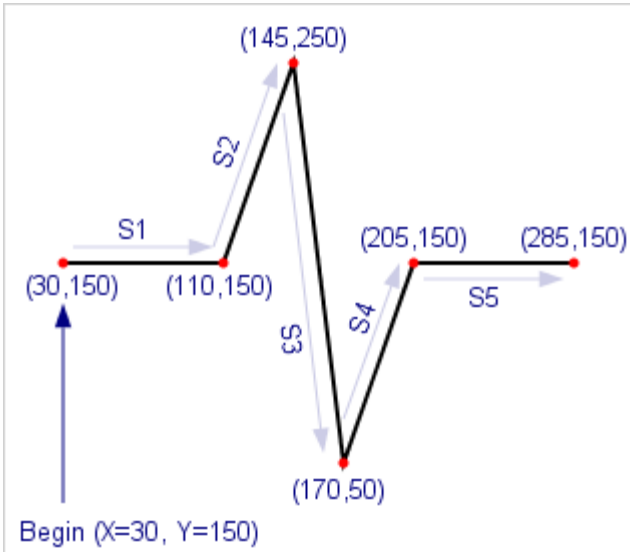
Here is a listing of the data.

Data Set POLYLINEDATA

Obs	function	drawspace	linecolor	linethickness	x1	y1
1	polyline	graphpixel	black	2	30	150
2	polycont	graphpixel	black	2	110	150
3	polycont	graphpixel	black	2	145	250
4	polycont	graphpixel	black	2	170	50
5	polycont	graphpixel	black	2	205	150
6	polycont	graphpixel	black	2	285	150

Notice that the first observation is the POLYLINE function followed by one POLYCONT function observation for each of the five segments. In all of the observations, the DrawSpace column specifies that the coordinates are expressed in graph-area pixels. In the first observation (POLYLINE), the X1 and Y1 columns specify the coordinates of the polyline starting point. In the subsequent observations, the X1 and Y1 columns specify the ending point for each segment.

Here is how this polyline is placed in a graph and how each segment is drawn.



The first segment (S1) is drawn from the polyline starting point to the ending point that is specified in the first segment observation. The second segment (S2) is drawn

from the ending point of S1 to the ending point that is specified in the S2 observation. This pattern repeats for the remaining segments.

A polygon is similar to a polyline except that the segments must form a closed shape. If the segments that you specify result in an open shape, a segment is drawn automatically from the ending point of the last segment back to the beginning point of the polygon in order to close the shape.

Here is an example of a DATA step that creates the data for a four-segment polyline annotation.

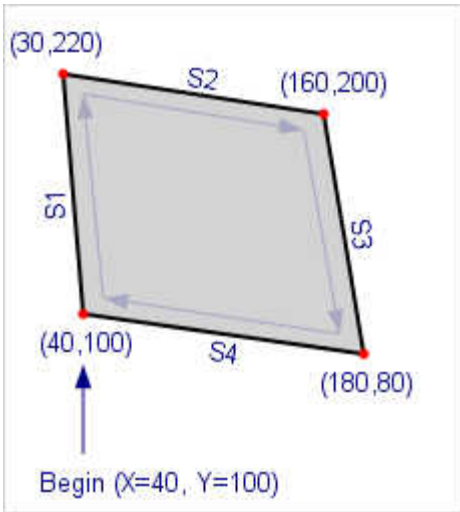
```
data polygondata;
  length function $8;
  function="polygon";
  drawspace="graphpixel";
  x1=40; y1=100;  /* Begin polygon */
  display="all"; fillcolor="lightgray";
  linecolor="black"; linethickness=2;
  output;
  function="polycont";
  x1=30; y1=220;  /* Draw Segment 1 */
  output;
  x1=160; y1=200; /* Draw Segment 2 */
  output;
  x1=180; y1=80;  /* Draw Segment 3 */
  output;
  x1=40; y1=100;  /* Draw Segment 4 */
  output;
run;
```

Here is a listing of the data.

```
Data Set POLYGONDATA

Obs function drawspace   x1  y1 display fillcolor linecolor linethickness
 1  polygon  graphpixel  40 100   all   lightgray   black         2
 2  polycont graphpixel  30 220   all   lightgray   black         2
 3  polycont graphpixel 160 200   all   lightgray   black         2
 4  polycont graphpixel 180  80   all   lightgray   black         2
 5  polycont graphpixel  40 100   all   lightgray   black         2
```

Because a polygon is a closed shape, you can add the FillColor column and specify a fill color for the polygon. By default, only the polygon outline is displayed. To display the fill, you must add the Display column and specify ALL or FILLED. For more information about these columns and other columns that you can add, see "POLYGON Function" in *SAS ODS Graphics: Procedures Guide*.

Here is how this polygon is placed in a graph and how each segment is drawn.



TIP In this example, you can omit the observation for the fourth segment (S4). In that case, the fourth segment is drawn automatically to close the shape.

Adding Observations for Ovals

To add an oval to your SG annotation data set, you must specify, at a minimum, OVAL in the Function column, the coordinates of the center point, the width, and the height. By default, the height and width are expressed in PERCENT units. To use different units such as PIXEL, you can add the HeightUnit and WidthUnit columns to specify the new units. You can specify other attributes such as the line color, line pattern, line thickness, and so on. For information about other columns that you can add, see "OVAL Function" in *SAS ODS Graphics: Procedures Guide*.

Here is an example of a DATA step that creates the data for a 90-pixel by 110-pixel oval annotation.

```
data ovaldata;
   function="oval";
   drawspace="graphpixel";
   x1=115; y1=110;
   width=90; widthunit="pixel"; height=110; heightunit="pixel";
   display="all"; fillcolor="lightgray";
   linecolor="blue"; linepattern="shortdash";
run;
```

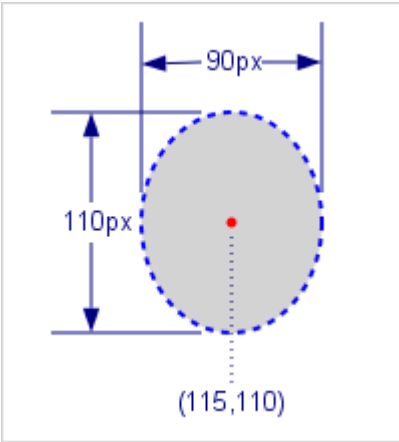
Note: Starting with SAS 9.4M1, you can also use the %SGOVAL macro to generate the oval annotation observations. See "Using the SG Annotation Macros to Create Your SG Annotation Data Set" on page 451.

Here is a listing of the data.

```
Data Set OVALDATA

Obs  function  drawspace    x1   y1  width  widthunit  height  heightunit
 1     oval    graphpixel  115  110    90     pixel      110     pixel

Obs  display    fillcolor    linecolor    linepattern    layer
 1     all      lightgray      blue        shortdash     back
```

In this observation, the Function column specifies OVAL for an oval annotation. The DrawSpace column specifies that all coordinates are expressed in graph-area pixels. The X1 and Y1 columns specify the oval center location as 115 on the X axis and 110 on the Y axis. The Width and WidthUnit columns specify a width of 90 pixels, and the Height and HeightUnit columns specify a height of 110 pixels. The FillColor column specifies a fill color for the oval. By default, only the oval outline is displayed. The Display column specifies ALL, which displays both the outline and the fill. The LineColor and LinePattern columns specify a blue short-dashed outline.

Here is how this annotation is placed in a graph.



Adding Observations for Rectangles

To add a rectangle to your SG annotation data set, you must specify, at a minimum, RECTANGLE in the Function column, the coordinates of the anchor point, the width, and the height. By default, the height and width are expressed in PERCENT units. To use different units such as PIXEL, you can add the HeightUnit and WidthUnit columns to specify the new units. You can specify other attributes such as the anchor, line color, line pattern, line thickness, and so on. For information about other columns that you can add, see "RECTANGLE Function" in *SAS ODS Graphics: Procedures Guide*.

Here is an example of a DATA step that creates the data for 60%-by-40% rectangle annotation.

```
data rectangledata;
  function="rectangle";
  drawspace="graphpixel";
  x1=80; y1=80; width=60; height=40; anchor="bottomleft";
  fillcolor="lightgray"; display="all";
  linecolor="black"; linethickness=2;
```

```
run;
```

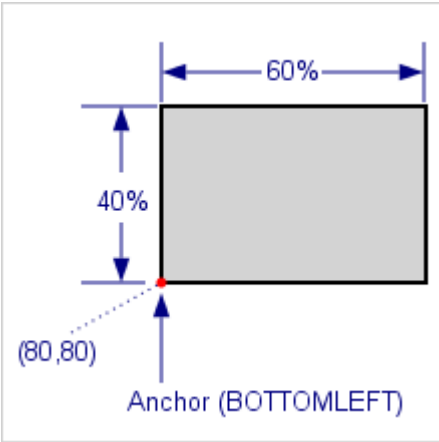
Note: Starting with SAS 9.4M1, you can also use the %SGRECTANGLE macro to generate the rectangle annotation observations. See "Using the SG Annotation Macros to Create Your SG Annotation Data Set" on page 451.

Here is a listing of the data.

```
Data Set RECTANGLEDATA

Obs    function     drawspace     x1    y1    width    height      anchor
 1     rectangle    graphpixel    80    80     60       40      bottomleft

Obs    fillcolor    display    linecolor    linethickness
 1     lightgray      all        black            2
```

In this observation, the Function column specifies RECTANGLE for a rectangle annotation. The DrawSpace column specifies that all coordinates are expressed in graph-area pixels. The X1 and Y1 columns specify the oval center location as 80 on the X axis and 80 on the Y axis. The Width column specifies a width of 60% (default unit), and the Height column specifies a height of 40%. The Anchor column specifies the anchor at the bottom left corner of the rectangle. The FillColor column specifies a fill color for the rectangle. By default, only the rectangle outline is displayed. The Display column specifies ALL, which displays both the outline and the fill. The LineColor and LineThickness columns specify a black outline that is two pixels wide.

Here is how this annotation is placed in a graph.



Adding Observations for Images

To add an image to your SG annotation data set, you must specify, at a minimum, IMAGE in the Function column and the path to the file that contains the image in the Image column. By default, the image is located in the center of the graph area and is anchored at CENTER. You can add the X1 and Y1 columns to specify numeric coordinates or the Xc1 and Yc1 columns to specify character coordinates for a new location. You can add the Anchor column to specify a different anchor. You can add other columns to specify other attributes such as a border, the image height, width,

scale, and so on. For information about the columns that you can add, see "IMAGE Function" in *SAS ODS Graphics: Procedures Guide*.

Here is an example of a DATA step that creates the data for a SAS logo.

```
data imagedata;
  function="image";
  drawspace="graphpixel";
  x1=70; y1=60;
  anchor="bottomleft";
  border="true";
  image=".\images\saslogo.png";
run;
```

Note: Starting with SAS 9.4M1, you can also use the %SGIMAGE macro to generate the image annotation observations. See "Using the SG Annotation Macros to Create Your SG Annotation Data Set" on page 451.

Here is a listing of the data.

```
Data Set IMAGEDATA

Obs  function  drawspace   x1  y1    anchor    border         image
 1    image    graphpixel  70  60  bottomleft   true   .\images\saslogo.png
```

In this observation, the Function column specifies IMAGE for an image annotation. The DrawSpace column specifies that all coordinates are expressed in graph-area pixels. The X1 and Y1 columns specify the location as 70 on the X axis and 60 on the Y axis. The Anchor column specifies the anchor at the bottom left corner. The Border column specifies a border around the image. The Image column specifies the relative path to the image file.

Note: The image file must be accessible on the file system. URL access is not supported.

Here is how this annotation is placed in a graph.

Using the SG Annotation Macros to Create Your SG Annotation Data Set

Table 23.1 SG Annotation Macros

Macro Name[1]	Description
%SGANNO	Compiles the available macros and makes them available for you to use. **Note:** You must run the %SGANNO macro in your SAS session before you can use any of the other annotation macros that are listed in this table.
%SGANNO_HELP	Displays help information for a specified annotation macro.
%SGARROW	Creates an observation that draws an arrow.
%SGIMAGE	Creates an observation that displays an image.
%SGLINE	Creates an observation that draws a line.
%SGOVAL	Creates an observation that draws an oval.
%SGPOLYCONT	Creates an observation that draws a polygon or polyline segment.
%SGPOLYGON	Creates an observation that specifies the starting point of a polygon. **Note:** Use the %SGPOLYCONT macro to draw the segments.
%SGPOLYLINE	Creates an observation that specifies the starting point of a polyline. **Note:** Use the %SGPOLYCONT macro to draw the segments.
%SGRECTANGLE	Creates an observation that draws a rectangle.
%SGTEXT	Creates an observation that draws a single line of text or the first line of text in a multiline annotation. **Note:** For multiple lines of text, use the %SGTEXTCONT macro for the subsequent lines.

Macro Name[1]	Description
%SGTEXTCONT	Creates an observation that draws a continuing line of text in a multiline annotation.

1 The macros listed in this table are valid starting with SAS 9.4M1. You can run these macros in a DATA step to simplify the process of creating your SG annotation data sets.

For more information about these macros, see *SAS ODS Graphics: Procedures Guide*.

Rendering a Graph with Annotations

In order to render a graph with annotations, you must include at least one ANNOTATE statement in the graph template. The ANNOTATE statement causes the annotations to be read from the SG annotation data set and drawn on the graph. You can place an ANNOTATE statement anywhere in the template. However, the drawing spaces that are specified for the annotations must be valid in the context of the ANNOTATE statement location. The order in which the annotations are drawn with respect to the graph elements depends on the LAYER option for the annotations. By default, annotations are always drawn on the front layer. In that case, the annotations are drawn after the graph elements are drawn and appear on top of the plots and other graph elements. If you specify BACK for the annotation layer, the annotations are drawn before the graph elements are drawn and appear behind the plots and other graph elements.

When you render your graph, you must include the SGANNO= option in the SGRENDER statement to specify the name of your SG annotation data set.For examples, see “Example 1: Creating Custom Labels” on page 453 and “Example 2: Displaying Subsets of Annotations in Axis-Aligned Insets” on page 457.

Subsetting Annotations

By default, the ANNOTATE statement draws all of the annotations that are stored in the SG annotation data set. You can use the ANNOTATE statement ID= option to display a subset of these annotations. Subsetting your annotations enables you to use one SG annotation data set with multiple plots. To use the ID= option, the observations for the annotations must contain an ID column. The ID column is a character column that stores a unique character value that identifies the subset to which each annotation belongs. When the ID="*identifier*" option is used in the ANNOTATE statement, only the annotations with ID column values that match the *identifier* value are displayed. The remaining annotations are ignored. For more information about the ANNOTATE statement ID= option, see “ANNOTATE” in *SAS Graph Template Language: Reference*. For an example, see “Example 2: Displaying Subsets of Annotations in Axis-Aligned Insets” on page 457.

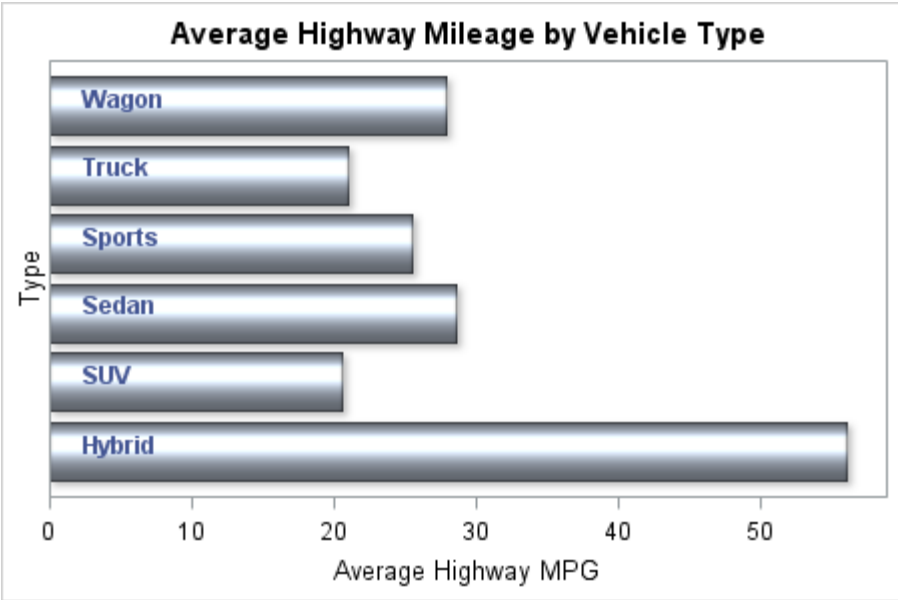
Examples

Example 1: Creating Custom Labels

Overview

This example demonstrates how to use the Graphic Template Language (GTL) annotation facility to create custom category labels for a horizontal bar chart. The bar chart plots the average highway mileage (response) by vehicle type (category). Ordinarily, the category labels for a horizontal bar chart appear on the Y axis to the left of each bar. This example demonstrates how to print the labels on the left end of each bar instead. This example also uses the sheen data skin on the bars. Because of the reflection on the sheen data skin, the labels are raised slightly to center the label in the reflection on each bar. Finally, the label text color is set to the graph contrast color.

The following figure shows the final graph.



Program

```
/* Summarize the highway mileage data in SASHELP.CARS. */
proc summary data=sashelp.cars nway;
  class type;
  var mpg_highway;
  output out=mileage mean(mpg_highway) = mpg_highway;
run;

/* Create the annotation data set. */
data anno;
  retain function "text" drawspace "datavalue"
    textfont "Arial" textweight "bold"
    textcolor "GraphData1:contrastColor"
    width 100 widthunit "pixel"
    anchor "left" x1 2
    discreteoffset 0.1;
  set mileage(keep=type);
  rename type=yc1;
  length label $12;
  label=type;
run;

/* Create the template. */
proc template;
  define statgraph barchart;
    begingraph;
      entrytitle "Average Highway Mileage by Vehicle Type";
      layout overlay /
        yaxisopts=(display=(label))
        xaxisopts=(label="Average Highway MPG" offsetmax=0.05);
        barchartparm category=type response=mpg_highway /
          orient=horizontal dataskin=sheen;
        annotate;
      endlayout;
   endgraph;
 end;

/* Render the graph with the Anno annotation data set */
proc sgrender data=mileage template=barchart sganno=anno;
run;
```

Program Description

Summarize the highway mileage data in Sashelp.Cars. Because a label is needed for each unique value of vehicle type, the data in Sashelp.Cars is first summarized for the Mpg_Highway column using the Type column as the class variable. This step generates a data set that contains one observation for each unique value of Type. See “Listing of the Mileage Data Set” on page 456.

```
/* Summarize the highway mileage data in SASHELP.CARS. */
proc summary data=sashelp.cars nway;
  class type;
  var mpg_highway;
```

```
  output out=mileage mean(mpg_highway) = mpg_highway;
run;
```

Create the SG annotation data set. The Mileage data set is used to create the annotation data set Anno. The DATA step in the Anno data set reads the observations from the Mileage data set. The Type column is used to set the Label column and is then renamed to Yc1. The remaining columns from the Mileage data set are then dropped. The X1 column is added and set to 2 in order to position the labels on the left end of each bar. The DiscreteOffset column is added and set to 0.1 in order to center the labels in the sheen data skin reflection on each bar. Additional columns are added to specify other attributes of the labels. See "Listing of the Anno Data Set" on page 456.

```
/* Create the annotation data set. */
data anno;
  retain function "text" drawspace "datavalue"
    textfont "Arial" textweight "bold"
    textcolor "GraphData1:contrastColor"
    width 100 widthunit "pixel"
    anchor "left" x1 2
    discreteoffset 0.1;
  set mileage(keep=type);
  rename type=yc1;
  length label $12;
  label=type;
run;
```

Create the graph template and include an ANNOTATE statement. The BARCHARTPARM statement is used to generate the horizontal bar chart from the summarized mileage data. Because the annotation drawing space is DATAVALUE, the ANNOTATE statement is placed in the BARCHARTPARM statement layout block in the template. In this location, the DATAVALUE values are in the context of the BARCHARTPARM data.

```
/* Create the template. */
proc template;
  define statgraph barchart;
    begingraph;
      entrytitle "Average Highway Mileage by Vehicle Type";
      layout overlay /
        yaxisopts=(display=(label))
        xaxisopts=(label="Average Highway MPG" offsetmax=0.05);
        barchartparm category=type response=mpg_highway /
          orient=horizontal dataskin=sheen;
        annotate;
      endlayout;
   endgraph;
 end;
```

Render the graph with the annotations. To render the graph with the annotations, the SGANNO=ANNO option is included in the SGRENDER statement.

```
/* Render the graph with the Anno annotation data set */
proc sgrender data=mileage template=barchart sganno=anno;
run;
```

Using the %SGTEXT Macro in the DATA Step

Starting with SAS 9.4M1, you can use the %SGTEXT annotation macro to simplify the DATA step for the Anno data set in this example. Here is the DATA step for the Anno data set modified to use the %SGTEXT macro.

```
%sganno; /* Compile the annotation macros */
data anno;
  set mileage;
  %sgtext(label=type,
    x1=2,yc1=type,drawspace="datavalue",
    textfont="Arial",textweight="bold",
    textcolor="GraphData1:contrastColor",width=100,
    widthunit="pixel",anchor="left",discreteoffset=0.1);
run;
```

In order to use the %SGTEXT macro, you must first run the %SGANNO macro to compile the SG annotation macros. The Type column in the Mileage data set is assigned to the Label and Yc1 columns in the annotation data set.

For more information about the SG annotation macros, see "SG Annotation Macro Dictionary" in *SAS ODS Graphics: Procedures Guide*. For another example of using the %SGTEXT macro, see "Example 2: Displaying Subsets of Annotations in Axis-Aligned Insets" on page 457.

Listing of the Mileage Data Set

Here is a listing of the Mileage data set.

Obs	Type	_TYPE_	_FREQ_	mpg_city	mpg_highway
1	Hybrid	1	3	55.0000	56.0000
2	SUV	1	60	16.1000	20.5000
3	Sedan	1	262	21.0840	28.6298
4	Sports	1	49	18.4082	25.4898
5	Truck	1	24	16.5000	21.0000
6	Wagon	1	30	21.1000	27.9000

Listing of the Anno Data Set

Here is a listing of the Anno data set.

Obs	function	drawspace	textfont	textweight	textcolor
1	text	datavalue	Arial	bold	GraphData1:contrastColor
2	text	datavalue	Arial	bold	GraphData1:contrastColor
3	text	datavalue	Arial	bold	GraphData1:contrastColor
4	text	datavalue	Arial	bold	GraphData1:contrastColor
5	text	datavalue	Arial	bold	GraphData1:contrastColor
6	text	datavalue	Arial	bold	GraphData1:contrastColor

Obs	discreteoffset	width	widthunit	anchor	x1	yc1	label
1	0.1	100	pixel	left	2	Hybrid	Hybrid
2	0.1	100	pixel	left	2	SUV	SUV
3	0.1	100	pixel	left	2	Sedan	Sedan
4	0.1	100	pixel	left	2	Sports	Sports
5	0.1	100	pixel	left	2	Truck	Truck
6	0.1	100	pixel	left	2	Wagon	Wagon

Example 2: Displaying Subsets of Annotations in Axis-Aligned Insets

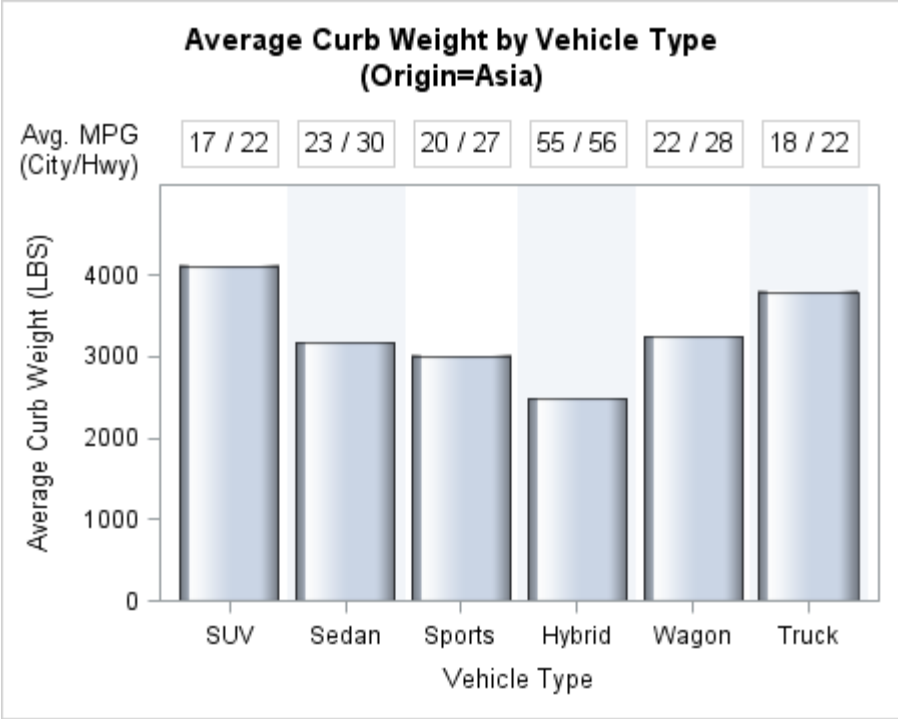
Overview

This example shows you how to do the following:

- subset annotations in an SG annotation data set and display specific subsets of annotations in a graph
- create axis-aligned insets outside of the plot wall area by using the GTL annotation facility
- use the %SGTEXT annotation macro to generate observations for text annotations

Note: The %SGTEXT macro is valid in SAS 9.4M1 and in later releases.

This example creates a vertical bar chart of vehicle type and average curb weight by vehicle origin. The following figure shows the first graph.



Above each bar in the bar chart, an axis-aligned inset displays the average city and highway mileages for that vehicle type. A label above the Y-axis area describes the inset content. The dynamic variable _BYLINE_ is used to specify the origin in the chart title. The annotation data for this example requires a subset of data for each vehicle type and an additional subset for the inset label.

Program

```
/* Sort Sashelp.Cars by Origin and summarize the mileage data */
proc sort data=sashelp.cars out=cars;
  by origin;
run;

proc summary data=cars nway mean;
  class type;
  by origin;
  var mpg_city mpg_highway;
  output out=mileage(drop=_type_ _freq_)
    mean(mpg_city mpg_highway) = mpg_city mpg_highway;
run;

/* Create variables for the inset positions. */
%let toprowpos=80; /* Y position of the insets (GRAPHPERCENT). */
%let labelpos=16;  /* X position of the inset label (GRAPHPERCENT) */

/* Compile the annotation macros */
%sganno;

/* Create the annotation data set. */
data anno;
  set mileage end=_LAST; /* Base on Mileage data. */
```

```
  %sgtext(id=origin,
    label=put(round(mpg_city), F2.0) || " / " ||
      put(round(mpg_highway), F2.0),
    xc1=type,x1space="datavalue",
    y1=&toprowpos,y1space="graphpercent",
    border="true",anchor="center",justify="center",width=90);

  /* Add observations for the inset label to the end of the data. */
  if (_LAST) then do;
    %sgtext(reset=all,id="INSETLABEL",label="Avg. MPG",
      x1=&labelpos,x1space="graphpercent",
      y1=%eval(&toprowpos+1),y1space="graphpercent",
      border="false",anchor="right",width=90,justify="left");
    %sgtext(label="(City/Hwy)",y1=%eval(&toprowpos-3));
  end;
run;

/* Create the graph template and include the ANNOTATE statements. */
proc template;
  define statgraph barchart;
    dynamic _BYVAL_ _BYLINE_;
    begingraph / designwidth=450 designheight=360;
      entrytitle "Average Curb Weight by Vehicle Type";
      entrytitle "(" _BYLINE_ ")";
      layout overlay;
        layout overlay / pad=(top=40)
          yaxisopts=(label="Average Curb Weight (LBS)" offsetmax=0.2)
          xaxisopts=(label="Vehicle Type"
            discreteopts=(colorbands=even
              colorbandsattrs=(transparency=0.6)));

          /* Generate the mileage bar chart */
          barchart x=type y=weight / stat=mean dataskin=gloss;

          /* Draw the mileage annotations for this origin */
          annotate / ID="INSETLABEL"; /* Draw the inset label. */
          annotate / ID=_BYVAL_;    /* Draw the bar insets. */
        endlayout;
      endlayout;
    endgraph;
  end;
run;

/* Render the graph by Origin with the Anno annotation data set */
options nobyline; /* Suppress the default BY line. */
proc sgrender data=cars template=barchart sganno=anno;
  by origin;
run;
options byline; /* Restore the default BY Line. */
```

Program Description

Sort Sashelp.Cars by Origin and summarize the mileage data. The data in Sashelp.Cars is first sorted by Origin. Next, the sorted data is summarized for the Mpg_City and Mpg_Highway columns by Origin, using Type as the classifier. The

Freq and _Type_ columns are dropped, and the output is written to the data set Mileage. See "Listing of the Mileage Data Set" on page 461.

```
/* Sort Sashelp.Cars by Origin and summarize the mileage data */
proc sort data=sashelp.cars out=cars;
  by origin;
run;

proc summary data=cars nway mean;
  class type;
  by origin;
  var mpg_city mpg_highway;
  output out=mileage(drop=_type_ _freq_)
    mean(mpg_city mpg_highway) = mpg_city mpg_highway;
run;
```

Create variables for the inset positions. To make it easier to adjust the inset positions, macro variables are created in order to specify the inset X and Y positions as graph percentages.

```
/* Create variables for the inset positions. */
%let toprowpos=80; /* Y position of the insets (GRAPHPERCENT). */
%let labelpos=16;  /* X position of the inset label (GRAPHPERCENT) */
```

Compile the annotation macros. Before the annotation macros can be used, the %SGANNO macro must be run to compile them.

```
/* Compile the annotation macros */
%sganno;
```

Create the SG annotation data set. The bar inset data is based on the data set Mileage. The %SGTEXT annotation macro generates the observations for the bar insets. The column Origin in the data set Mileage is specified in the ID= argument, and the column Type is specified in the Xc1= argument. The LABEL= argument specifies the inset text as *cc* / *hh*, where *cc* is Mpg_City and *hh* is Mpg_Highway.

```
/* Create the annotation data set. */
data anno;
  set mileage end=_LAST; /* Base on Mileage data. */
  %sgtext(id=origin,
    label=put(round(mpg_city), F2.0) || " / " ||
      put(round(mpg_highway), F2.0),
    xc1=type,x1space="datavalue",
    y1=&toprowpos,y1space="graphpercent",
    border="true",anchor="center",justify="center",width=90);
```

Add observations for the inset label to the end of the data. The %SGTEXT macro generates the label observations. The RESET=ALL argument in the first macro call resets all of the argument values from the previous macro call. Because only the label text and the Y position values change for the second label observation, the second macro call specifies only the LABEL= and Y1= arguments. The values specified in the previous macro call apply to the unspecified arguments in the second macro call.

```
/* Add observations for the inset label to the end of the data. */
if (_LAST) then do;
  %sgtext(reset=all,id="INSETLABEL",label="Avg. MPG",
    x1=&labelpos,x1space="graphpercent",
    y1=%eval(&toprowpos+1),y1space="graphpercent",
    border="false",anchor="right",width=90,justify="left");
  %sgtext(label="(City/Hwy)",y1=%eval(&toprowpos-3));
```

```
  end;
run;
```

Create the graph template and include the ANNOTATE statements. The BY line is displayed in the graph title. The LAYOUT OVERLAY statement PAD= option reserves space for the bar insets. Because the annotation XSPACE is DATAVALUE, the ANNOTATE statements are placed in the BARCHART statement's layout block. The first ANNOTATE statement draws the inset label, and the second draws the bar insets. The _BYVAL_ values (Origin) are used as the bar inset IDs.

```
/* Create the graph template and include the ANNOTATE statements. */
proc template;
  define statgraph barchart;
    dynamic _BYVAL_ _BYLINE_;
    begingraph / designwidth=450 designheight=360;
      entrytitle "Average Curb Weight by Vehicle Type";
      entrytitle "(" _BYLINE_ ")";
      layout overlay;
        layout overlay / pad=(top=40)
          yaxisopts=(label="Average Curb Weight (LBS)" offsetmax=0.2)
          xaxisopts=(label="Vehicle Type"
            discreteopts=(colorbands=even
              colorbandsattrs=(transparency=0.6)));

          /* Generate the mileage bar chart */
          barchart x=type y=weight / stat=mean dataskin=gloss;

          /* Draw the mileage annotations for this origin */
          annotate / ID="INSETLABEL"; /* Draw the inset label. */
          annotate / ID=_BYVAL_;    /* Draw the bar insets. */
        endlayout;
      endlayout;
    endgraph;
  end;
run;
```

Render the graph with the annotations. The SGANNO=ANNO option is included in the SGRENDER statement. Because the BY line is included in the graph title, the default BY line is suppressed. The BY statement generates a graph for each unique value of Origin.

```
/* Render the graph by Origin with the Anno annotation data set */
options nobyline; /* Suppress the default BY line. */
proc sgrender data=cars template=barchart sganno=anno;
  by origin;
run;
options byline; /* Restore the default BY Line. */
```

Listing of the Mileage Data Set

Here is a listing of the Mileage data set.

Obs	Origin	Type	mpg_city	mpg_highway
1	Asia	Hybrid	55.0000	56.0000
2	Asia	SUV	17.3200	21.6800
3	Asia	Sedan	22.8404	29.9681
4	Asia	Sports	20.2353	26.6471
5	Asia	Truck	17.8750	22.0000
6	Asia	Wagon	22.3636	28.1818
7	Europe	SUV	14.5000	18.7000
8	Europe	Sedan	19.5128	27.1154
9	Europe	Sports	17.6522	25.1304
10	Europe	Wagon	19.2500	26.5833
11	USA	SUV	15.5200	20.0400
12	USA	Sedan	20.6111	28.5444
13	USA	Sports	16.8889	24.2222
14	USA	Truck	15.8125	20.5000
15	USA	Wagon	22.2857	29.7143

Partial Listing of the Anno Data Set

Here are the last seven observations from the Anno data set.

Obs	ID	FUNCTION	LABEL	BORDER	ANCHOR	JUSTIFY
…						
11	USA	TEXT	16 / 20	true	center	center
12	USA	TEXT	21 / 29	true	center	center
13	USA	TEXT	17 / 24	true	center	center
14	USA	TEXT	16 / 21	true	center	center
15	USA	TEXT	22 / 30	true	center	center
16	INSETLABEL	TEXT	Avg. MPG	false	right	left
17	INSETLABEL	TEXT	(City/Hwy)	false	right	left

Obs	WIDTH	X1	XC1	X1SPACE	Y1	Y1SPACE
…						
11	90	.	SUV	datavalue	80	graphpercent
12	90	.	Sedan	datavalue	80	graphpercent
13	90	.	Sports	datavalue	80	graphpercent
14	90	.	Truck	datavalue	80	graphpercent
15	90	.	Wagon	datavalue	80	graphpercent
16	90	16		graphpercent	81	graphpercent
17	90	16		graphpercent	77	graphpercent

Note: The actual column order is different from that presented here.

Example 3: Creating a Macro That Generates Annotation Data

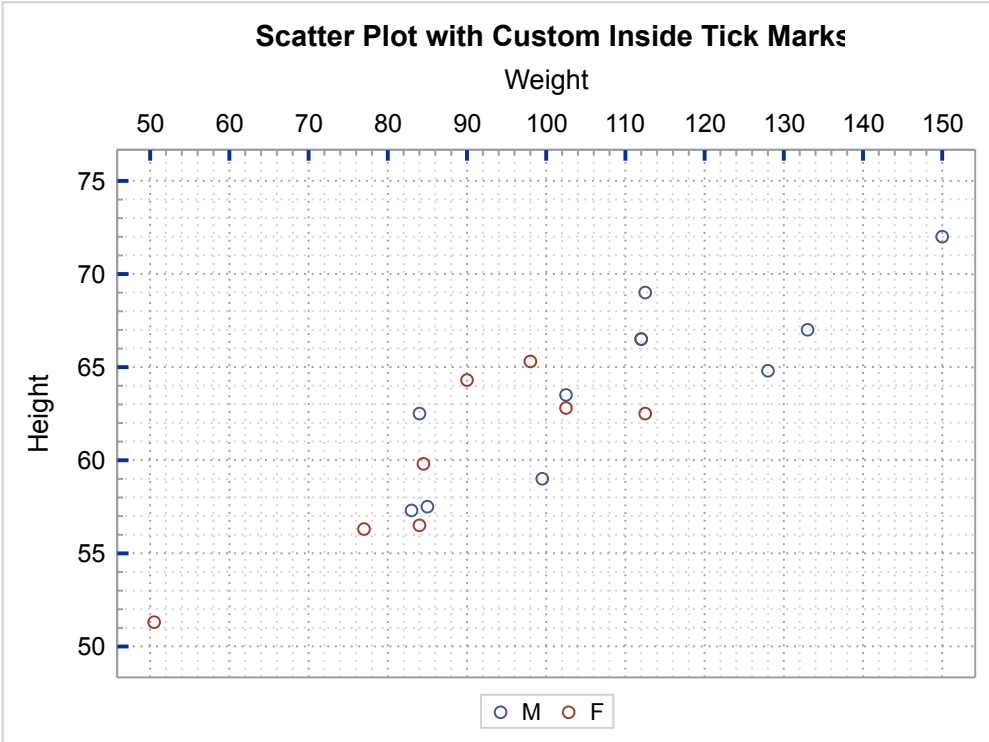
Overview

This example shows you how to create a macro that generates annotation data for your graphs. It is based on the solution in post Creating an axis in GTL with both major and minor tick marks shown inside on the Graphics Programming community page. This example shows you how to apply the solution to GTL and how to use a macro to generate annotation data.

The SAS macro facility programming language provides an extensive set of programming features that you can use to perform tasks and to generate SAS code. Macros can be compiled and stored for later use and can be shared with other users. You can find detailed information about the SAS macro facility and examples of how to use it in *SAS Macro Language: Reference*.

This example creates macro %GENTICKANNODATA, which generates annotation data that draws custom tick marks inside a graph. Here is the output for this example.

Figure 23.1 *Plot with Custom Inside Tick Marks*



This example also shows you how to reference macro variables in GTL template code using NMVAR run-time macro variable reference symbols.

About the %GENTICKANNODATA Macro

Macro %GENTICKANNODATA in this example generates data for a single axis per execution: X, X2, Y, or Y2. It accepts several parameters that are used to generate the annotation data. The data from multiple executions can be accumulated in a single data set, which is helpful when you want to customize more than one axis. Here is the syntax for macro %GENTICKANNODATA:

%GENTICKANNODATA(AXIS= X | X2 | Y | Y2 START=*nnn* END=*nnn* BY=*nnn* <DSN=*name*> <APPEND=Y | N> <MINORTICKCOUNT=*n*> <BASELINE=*n*> <LENGTH=*n*> <MAJORTICKCOLOR=*color-name* | *style-reference*> <MAJORTICKTHICKNESS=*n*> <MINORTICKCOLOR=*color-name* | *style-reference*> <MINORTICKTHICKNESS=*n*>)

The following table describes the macro parameters.

Parameter[1]	Description	Default
AXIS= X | X2 | Y | Y2	Specifies the axis. This parameter is required.	
START=*nnn*	Specifies the starting value on the axis. This parameter is required.	
END=*nnn*	Specifies the ending value on the axis. This parameter is required.	
BY=*nnn*	Specifies the major tick mark interval. This parameter is required.	
DSN=*name*	Specifies the output data set name.	TICKANNO
APPEND=Y | N	Specifies whether the data is appended to existing data (Y) or replaces existing data (N).	N
MINORTICKCOUNT=*n*	Specifies the number of minor tick marks.	0 (no minor ticks)
BASELINE=*n*	Specifies the starting baseline for the tick marks as a percentage of the graph wall area. This value can be used to make minor adjustments to the tick mark positions, if necessary.	0
LENGTH=*n*	Specifies the length of the major tick marks as a percentage of the graph wall area. The length of the minor tick marks is one-half of this value.	2

Parameter[1]	Description	Default
MAJORTICKCOLOR=*color-name* | *style-reference*	Specifies the color of the major tick marks.[2]	style reference GraphAxisLines:contrastColor
MAJORTICKTHICKNESS=*n*	Specifies the thickness of the major tick marks in pixels.	1
MINORTICKCOLOR=*color-name* | *style-reference*	Specifies the color of the minor tick marks.[2]	style reference GraphAxisLines:contrastColor
MINORTICKTHICKNESS=*n*	Specifies the thickness of the minor tick marks in pixels.	1

1 Character values are not enclosed in quotation marks.

2 For *color-name*, see "Predefined Colors" on page 685 and "Color-Naming Schemes" on page 673. For *style-reference*, see Appendix 2, "Graph Style Elements Used by ODS Graphics," on page 651.

The Macro %GENTICKANNODATA Code

Here is the code for macro %GENTICKANNODATA.

```
%macro genTickAnnoData(axis=, start=, end=, by=,                /* 1 */
   dsn=tickanno, append=n, minortickcount=0, baseline=0, length=2,
   majortickcolor=GraphAxisLines:contrastColor, majorticktickness=1,
   minortickcolor=GraphAxisLines:contrastColor, minorticktickness=1);

   %if %upcase(&append) eq Y %then %let name = &dsn._tmp;       /* 2 */
   %else %let name = &dsn;

   data &name;                                                  /* 3 */
      length id $2 x1space x2space y1space y2space $15
         linecolor $30 xaxis yaxis $4;
      retain id function x1space x2space y1space y2space linecolor
         linethickness gap xaxis yaxis;
      id = upcase(trim("&axis"));
      function = "line";
      gap = &by / (&minortickcount + 1);

      if (substr(id,2,1) eq "2") then primary = 0;              /* 4 */
      else primary = 1;

      if (id eq "X" or id eq "X2") then do;                     /* 5 */
         x1space = "datavalue"; x2space = "datavalue";
         y1space = "wallpercent"; y2space = "wallpercent";
         linecolor = "&majortickcolor";
         linethickness = &majorticktickness;
         if (primary) then do;
            xaxis="X";
            do i = &start to &end by &by;
               x1 = i; y1 = &baseline;
               x2 = x1; y2 = &baseline + &length; output;
            end;
```

```
      end;
      else do;
         xaxis="X2";
         do i = &start to &end by &by;
            x1 = i; y1 = 100 - (&baseline + &length);
            x2 = x1; y2 = 100 - &baseline; output;
         end;
      end;
      linecolor = "&minortickcolor";
      linethickness = &minorticktickness;
      do i = &start to &end - &by by &by;
         if (primary) then do;
            do z = 1 to &minortickcount;
               x1 = i + (gap * z); y1 = &baseline;
               x2 = x1; y2 = &baseline + (&length / 2); output;
            end;
         end;
         else do;
            do z = 1 to &minortickcount;
               x1 = i + (gap * z);
               y1 = 100 - (&baseline + (&length / 2));
               x2 = x1; y2 = 100 - &baseline; output;
            end;
         end;
      end;
   end;
   else if (id eq "Y" or id eq "Y2") then do;                /* 6 */
      y1space = "datavalue"; y2space = "datavalue";
      x1space = "wallpercent"; x2space = "wallpercent";
      linecolor = "&majortickcolor";
      linethickness = &majorticktickness;
      if (primary) then do;
         yaxis="Y";
         do i = &start to &end by &by;
            y1 = i; x1 = &baseline;
            y2 = y1; x2 = &baseline + &length; output;
         end;
      end;
      else do;
         yaxis="Y2";
         do i = &start to &end by &by;
            y1 = i; x1 = 100 - (&baseline + &length);
            y2 = y1; x2 = 100 - &baseline; output;
         end;
      end;
      linecolor = "&minortickcolor";
      linethickness = &minorticktickness;
      do i = &start to &end - &by by &by;
         if (primary) then do;
            do z = 1 to &minortickcount;
               y1 = i + (gap * z); x1 = &baseline;
               y2 = y1; x2 = &baseline + (&length / 2); output;
            end;
         end;
         else do;
            do z = 1 to &minortickcount;
```

```
                y1 = i + (gap * z); x1 = 100 - &baseline;
                y2 = y1;
                x2 = 100 - (&baseline + (&length / 2)); output;
             end;
          end;
       end;
    end;
    else put "ERROR: Axis " id "is not valid.";

    keep id function x1space x2space y1space y2space x1 x2 y1 y2
       linecolor linethickness xaxis yaxis;
 run;

 %if %upcase(&append) eq Y %then %do;                          /* 7 */
     proc append base=&dsn data=&name;
     run;
     proc delete data=&name;
     run;
 %end;
%mend genTickAnnoData;
```

1. Use the %MACRO statement to begin the macro definition. Specify the macro name and the input parameters. See "About the %GENTICKANNODATA Macro" on page 464 for a description of the input parameters.
2. If the data is to be appended to existing data, create a name for a temporary data set. Otherwise, use the name provided, which replaces the data set if it already exists.
3. Open the DATA step, and specify the data set name.
4. Determine whether the axis is primary or secondary.
5. Create the annotation data for the X or X2 axis, if requested.
6. Create the annotation data for the Y or Y2 axis, if requested.
7. If requested, append the data to the existing data set, and then delete the temporary data set.

Using Macro %GENTICKANNODATA in a GTL Program

The following program shows how to use the %GENTICKANNODATA macro to generate the annotation data for the custom tick marks shown in Figure 23.1 on page 463.

```
%let _xstart=50;          /* 1 */
%let _xend=150;
%let _xinc=10;
%let _xmtc=4;
%let _ystart=50;
%let _yend=75;
%let _yinc=5;
%let _ymtc=4;
```

```
%genTickAnnoData(axis=x2, start=&_xstart, end=&_xend,       /* 2 */
   by=&_xinc,
   majortickcolor=GraphDataDefault:contrastColor,
   minortickcolor=cxa0a0a0,minortickcount=&_xmtc,
   majortickthickness=2);

%genTickAnnoData(axis=y, append=y, start=&_ystart,          /* 3 */
   end=&_yend, by=&_yinc,
   majortickcolor=GraphDataDefault:contrastColor,
   minortickcolor=cxa0a0a0, minortickcount=&_ymtc,
   length=1.3, majortickthickness=2);

proc template;
   define statgraph scatter1;                               /* 4 */
      nmvar _xstart _xend _xinc _xmtc _ystart _yend
         _yinc _ymtc;
      begingraph;
         entrytitle "Scatter Plot with Custom Inside Tick Marks";
         layout overlay /
            x2axisopts=(display=(label line tickvalues)
               offsetmin=0.04 offsetmax=0.04
               griddisplay=on gridattrs=(color=cxa0a0a0 pattern=dot)
               linearopts=(minorgrid=true
                  minorgridattrs=(color=lightgray pattern=dot)
                  tickvaluesequence=(start=_xstart end=_xend
                     increment=_xinc)
                  minortickcount=_xmtc tickvaluepriority=true))
            yaxisopts=(display=(label line tickvalues)
               offsetmin=0.06 offsetmax=0.06
               griddisplay=on gridattrs=(color=cxa0a0a0 pattern=dot)
               linearopts=(minorgrid=true
                  minorgridattrs=(color=lightgray pattern=dot)
                  tickvaluesequence=(start=_ystart end=_yend
                     increment=_yinc)
                  minortickcount=_ymtc tickvaluepriority=true));
            scatterplot x=weight y=height / name="scatter"
               group=sex xaxis=x2;
            discretelegend "scatter";
            annotate;
         endlayout;
      endgraph;
   end;
run;

proc sgrender data=sashelp.class template=scatter1          /* 5 */
   sganno=tickanno;
run;
```

1. Create macro variables for the axis values. Using macro variables helps ensure that the axis data specifications are consistent within the program. It also makes it easy to change the values at a later time.
2. Generate the tick annotation data for the X2 axis. If data set TICKANNO already exists, replace it with the data for the X2 tick marks.
3. Generate the tick annotation data for the Y axis. Append the data to data set TICKANNO.

4 Create the graph template. The NMVAR statement defines symbols that reference the corresponding macro variables. The values are resolved at run time, which means you do not have to recompile the template when the macro variable values are changed. These symbols are used in the X2AXISOPTS= and YAXISOPTS= options in the LAYOUT OVERLAY statement. The ANNOTATE statement draws the annotations.

5 Render the graph. Specify the plot data set and the annotation data set.

PART 7

Creating Interactive Graphs

24

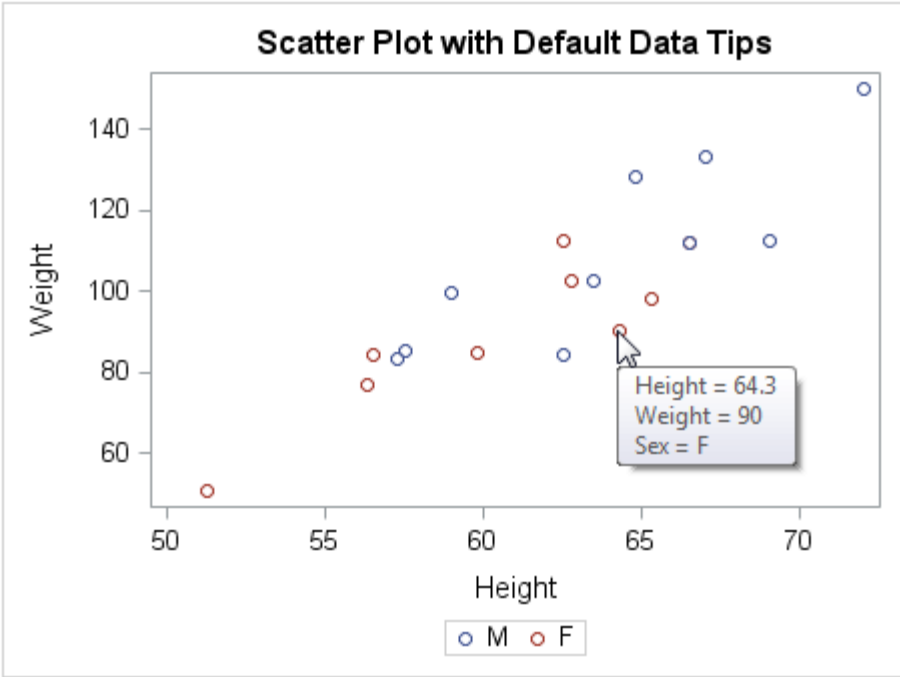
Adding Data Tips to Your Graph

Creating a Graph with Data Tips in an HTML Page

Data tips can be displayed by graphs that are included in HTML pages. When data tips are provided, you can position your pointer 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 as shown in the following example.

```
proc template;
  define statgraph defaultdatatips;
    begingraph;
      entrytitle "Scatter Plot with Default Data Tips";
      layout overlay;
        scatterplot x=height y=weight / group=sex name="s";
        discretelegend "s";
      endlayout;
    endgraph;
   end;
run;

ods graphics / reset width=5in imagemap=on;
proc sgrender data=sashelp.class template=defaultdatatips;
run;
ods graphics / reset;
```

Here is an example of a default data tip when the mouse pointer is held over a data point.



Prior to SAS 9.4M5, image-map support in the ODS HTML and ODS HTML5 destinations is restricted to only bitmapped images. Neither supports image maps for SVG images. Starting with SAS 9.4M5, the ODS HTML and ODS HTML5 destinations support image maps for SVG images as well. However, image-map support in the ODS HTML5 destination is restricted to only in-line SVG images.

ODS Graphics disables data tips when a preset upper threshold is reached. Prior to SAS 9.4M3, the threshold is based on the number of observations in the graph data. When the threshold is reached, data tips are disabled for the entire graph. Starting with SAS 9.4M3, the data-tip upper threshold is based on the number of data tips in each plot. The threshold is enforced on a per-plot basis. When the number of data tips in a plot reaches the upper data-tip threshold, data tips are disabled for that plot only. Data tips remain enabled for the remaining plots in the graph.

The data-tip upper threshold is controlled by the ODS GRAPHICS statement option TIPMAX=. The default is 500. When the data-tip upper threshold is reached, a note is written to the SAS log indicating that the data tips are disabled. The note provides information about how to use the TIPMAX= option in an ODS GRAPHICS statement to raise the threshold sufficiently to re-enable the data tips. For more information about the ODS GRAPHICS statement TIPMAX= option, see "ODS GRAPHICS" in *SAS Graph Template Language: Reference*.

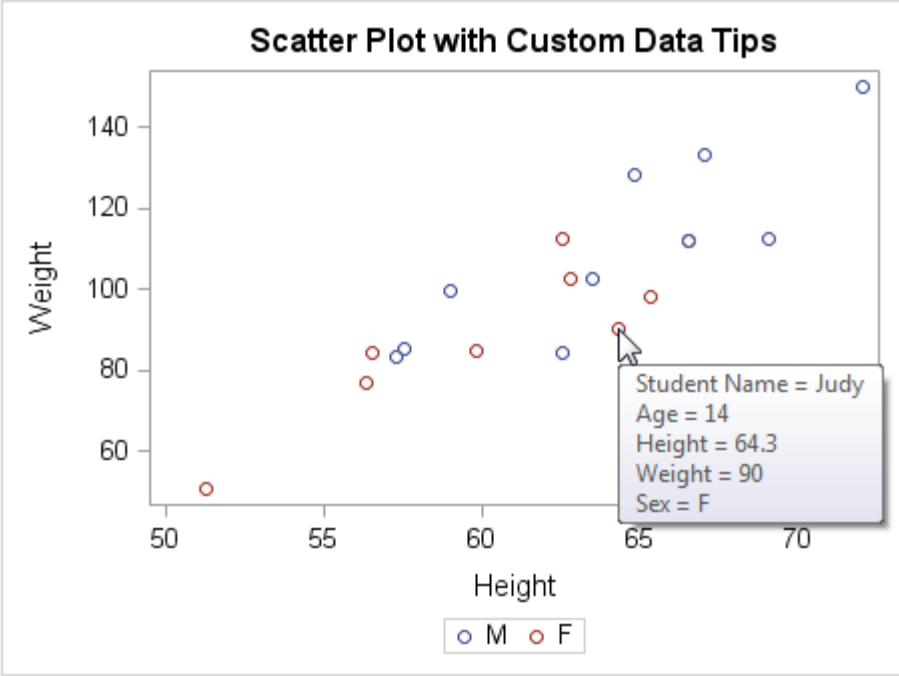
Creating a Graph with Custom Data Tips in an HTML Page

GTL supports plot statement syntax that enables you to suppress or customize the default data tip information. The following layout block creates custom data tips.

```
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;
```

Here is an example of a custom data tip when the mouse pointer is held over a data point.



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, the Name and Age column values are shown in the data 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 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 the following:

```
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 followingTIPLABEL= option:

```
tiplabel=(tip1="Student Name" group="Group")
```

For any role, you can assign a format to the display of tip values.

25

Adding Drill-Down Links to Your Graph

About Drill-Down Graphs

Drill-down graphs provide a convenient means for your users to explore complex data. In a drill-down graph, certain elements of the graph contain active links. When a user clicks a linked element, the linked resource appears in a new browser window by default. The resource can be another web page or an image file that contains a supporting graph. For several types of plots, you can use the URL= option in the plot statement to add drill-down links to your graphs for HTML presentations. The following plot statements support the URL= option:

BARCHART	MOSAICPLOTPARM	SERIESPLOT
BARCHARTPARM	NEEDLEPLOT	STEPPLOT
BUBBLEPLOT	PIECHART	TEXTPLOT
HEATMAPPARM	POLYGONPLOT	WATERFALLCHART
LINECHART	SCATTERPLOT	

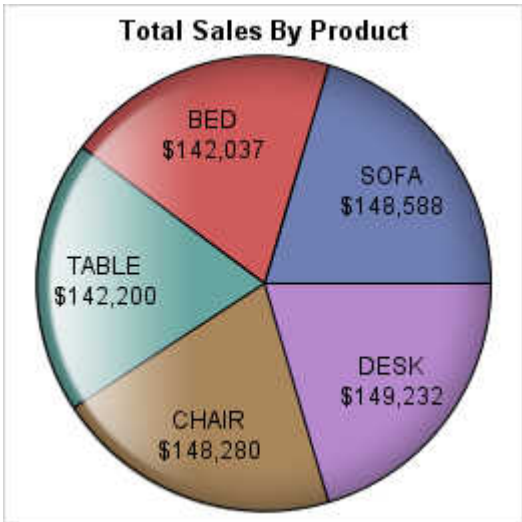
Starting with SAS 9.4M1, the following support is available:

- The following draw statements support the URL= option:

BEGINPOLYGON	DRAWLINE	DRAWIMAGE
BEGINPOLYLINE	DRAWARROW	DRAWOVAL
DRAWTEXT	DRAWRECTANGLE	

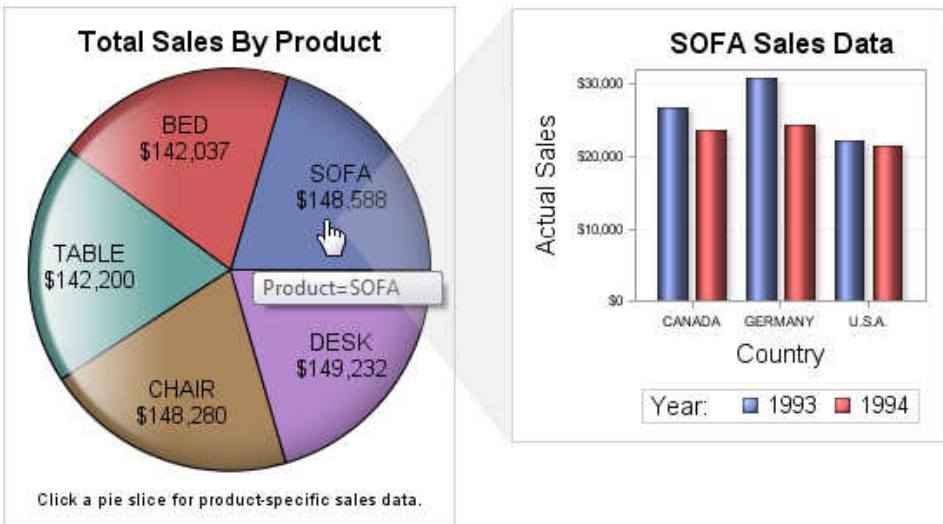
- Many of the ODS Graphics annotation functions support URLs. For information about the ODS Graphics annotation functions, see *SAS ODS Graphics: Procedures Guide*.

For an example of a drill-down graph, consider the pie chart shown in the following figure.



To provide your users with sales information for each product, you can create a drill-down pie chart in which each pie slice is linked to a bar chart that shows sales data for that product. You can also add a footnote that prompts the user to click a pie slice for more information. In this drill-down pie chart, if the user clicks the SOFA pie slice, a bar chart showing the sales data for SOFA appears in a new browser window, as shown in the following figure.

Figure 25.1 *Drill-Down Pie Chart*



When the mouse pointer is positioned on a pie slice, the data tip appears, and the mouse pointer changes to indicate an active link.

Drill-down links are generated only when they are enabled by ODS GRAPHICS statement option IMAGEMAP. Image maps are not generated by default. To generate drill-down links, you must include option IMAGEMAP or IMAGEMAP=ON in an ODS GRAPHICS statement. See "ODS GRAPHICS" in *SAS Graph Template Language: Reference*.

Prior to SAS 9.4M5, image-map support in the ODS HTML and ODS HTML5 destinations is restricted to only bitmapped images. Neither supports image maps for SVG images. Starting with SAS 9.4M5, the ODS HTML and ODS HTML5 destinations support image maps for SVG images as well. However, image-map support in the ODS HTML5 destination is restricted to only in-line SVG images.

Create a Drill-Down Graph

In order to create a drill-down graph, you must do the following:

- Add a column that contains the URL of each link to the data set for the drill-down graph.
- Use the URL= option in the plot statement to specify the data column that contains the link URLs.
- Enable image mapping in ODS Graphics by using the IMAGEMAP=ON option in the ODS GRAPHICS statement.
- Write the graph output to the ODS HTML or ODS HTML5 destination.

For example, to create the drill-down pie chart in Figure 25.1 on page 478, you must complete the following steps:

1. Add the URLs for the supporting bar charts to a character column in the data set for your main graph.

   Note: You must set the column length appropriately for the expected URL values. If a URL value exceeds the column length, the URL is truncated and the link does not work.

2. Create the template for the supporting bar charts.
3. Generate the supporting bar charts.
4. Create the template for the main pie chart. In the plot statement for the pie chart, include the URL=*url-column-name* option to specify the name of the column in the data set that contains the drill-down link URLs.
5. Use the following statement to enable image mapping in ODS Graphics:

   ```
   ods graphics / imagemap=on;
   ```

   Note: Drill-down links are displayed in a new browser window by default. To specify a different target, such as _blank or _self, add the DRILLTARGET= option to your ODS GRAPHICS statement. You must specify _blank or _self in lowercase. See "ODS GRAPHICS" in *SAS Graph Template Language: Reference*.

6. Generate the pie chart by using the ODS HTML destination.
7. Use the following statement to disable image mapping in ODS Graphics:

   ```
   ods graphics / imagemap=off;
   ```

Here is the SAS code for this example.

```
/* Specify the ODS output path */
filename outp "output-path";

/* 1. Add a URL column for the drill-down links to the
      SASHELP.PRDSALE data set. */
data sales;
  length url $30;
  set sashelp.prdsale;
  format actual dollar12.0;
  select (product);
    when ("SOFA")  url="sofa.html";
    when ("BED")   url="bed.html";
    when ("TABLE") url="table.html";
    when ("CHAIR") url="chair.html";
    when ("DESK")  url="desk.html";
    otherwise url=" ";
  end;
run;

/* 2. Create a template for the supporting graphs. */
proc template;
  define statgraph drilldown;
  begingraph;
    dynamic product;
    entrytitle product " Sales Data";
    layout overlay /
      yaxisopts=(griddisplay=on gridattrs=(color=lightgray
pattern=dot));
      barchart category=country response=actual /
        name="productsales"
        group=year
        groupdisplay=cluster
        barwidth=0.75
        dataskin=sheen;
      discretelegend "productsales" / title="Year:";
    endlayout;
  endgraph;
end;
run;

/* 3. Generate a supporting graph for each product. Because there are
      several products, create a macro that generates a graph
      for a specific product. */
%macro genchart(product=);
  /* Specify the image output filename. */
  ods graphics / imagename="&product";

  /* Generate the graph using ODS HTML. */
  ods _all_ close;
  ods html path=outp file="&product..html";
  proc sgrender data=sales template=drilldown;
    where product = "&product";
    dynamic product="&product"; /* Pass product to the template. */
  run;
  ods html close;
```

```
%mend genchart;

/* Use the macro to generate the supporting graphs. */
%genchart(product=SOFA);
%genchart(product=DESK);
%genchart(product=CHAIR);
%genchart(product=TABLE);
%genchart(product=BED);

/* 4. Create a template for the drill-down graph. */
proc template;
  define statgraph basechart;
  begingraph;
    entrytitle "Total Sales By Product";
    entryfootnote textattrs=(size=7pt) "Click a pie slice for
      product-specific sales data.";
    layout region;
      piechart category=product response=actual /
        datalabelcontent=(category response)
        datalabellocation=inside
        url=url
        tip=(category)
        dataskin=gloss;
    endlayout;
  endgraph;
end;
run;

/* 5. Enable image mapping in the HTML output and specify
      a base image name. */
ods graphics / reset imagemap=on imagename="prodsales"
  antialiasmax=2000 tipmax=2000;

/* 6. Generate the drill-down graph using ODS HTML. */
ods html path=outp file="sales.html";
proc sgrender data=sales template=basechart;
run;
ods html close;

/* 7. Disable image mapping and open an output destination. */
ods graphics / reset imagemap=off;
ods html; /* Not required in SAS Studio */
```

You can create multiple levels of drill-down graphs in your presentation. That is, your supporting graphs can also be drill-down graphs in as many levels as needed.

26

Creating Animated Graphs

About ODS Graphics Animation Support

An animated graph displays a series of charts automatically when the graph is viewed in a web browser or other viewer that supports animation. The animation plays as a sequence of graphs in a slide-show fashion with a delay between each graph. The sequence can play only one time, loop a fixed number of times and then stop, or loop indefinitely.

The ODS Graphics GIF and SVG universal printers support animation. The GIF format provides basic animation with no user interaction. The SVG format provides animation plus user-interaction features such as zoom, pause, replay, and so on. System options enable you to control the animation output and various properties of the animation. The following table lists the system options and the applicable universal printers for each.

Table 26.1 *System Options That Control Animated Graphs*

Option Name	Valid Printers	Description
ANIMATION=START | STOP	GIF and SVG	Starts or stops the creation of an animation file.
ANIMDURATION=MIN | *number*	GIF and SVG	Specifies the length of time, in seconds, that each frame in an animation is held in view. The default is MIN (0.1 seconds).

Option Name	Valid Printers	Description
ANIMLOOP=YES | NO | *number*	GIF and SVG	Specifies whether the animation loops. For GIF, `NO` specifies no looping, `YES` or `0` specifies continuous looping, and *number* specifies a fixed number of loops. The default is `YES`. For SVG, `NO` or a nonzero value specifies no looping. `YES` or `0` specifies continuous looping. The default is `YES`.
ANIMOVERLAY | NOANIMOVERLAY	GIF and SVG	Specifies whether animation frames are overlaid or played sequentially. The default is `ANIMOVERLAY`.
SVGAUTOPLAY | NOSVGAUTOPLAY	SVG	Specifies whether an SVG animation starts immediately in the web browser. The default is `SVGAUTOPLAY`.
SVGFADEIN=*number*	SVG	Specifies the number of seconds for an SVG frame to fade into view. The default is `0`.
SVGFADEMODE=OVERLAP | SEQUENTIAL	SVG	Specifies whether an SVG frame overlaps the previous frame or each frame is played sequentially when a frame is fading in and out. The default is `OVERLAP`.
SVGFADEOUT=*number*	SVG	Specifies the number of seconds that it takes for an SVG frame to fade out of view. The default is `0`.

For more information about these system options, see *SAS System Options: Reference*.

The animated graph output is contained in a GIF file or SVG file. An HTML file that displays the animation is not automatically created. To view the animation, you must manually incorporate the GIF file or SVG file into your own web page or document that supports animated images.

Create an Animated Graph

The basic steps for creating an animated graph in ODS Graphics are as follows:

1 Set the system options that you need in order to configure your animation. See Table 26.1 on page 483.

2 At the point in your SAS program where you want to start the animation output:

   a Set the ANIMATION=START system option to start the animation output.

   b Open the ODS PRINTER destination and specify the GIF printer or the SVG printer.

3 Run the ODS Graphics statements that are required to generate your sequence of graphs, such as SGRENDER, SGPLOT, and so on.

Note: To temporarily stop the animation output, set the ANIMATION=STOP system option. When you are ready to resume, set the ANIMATION=START system option.

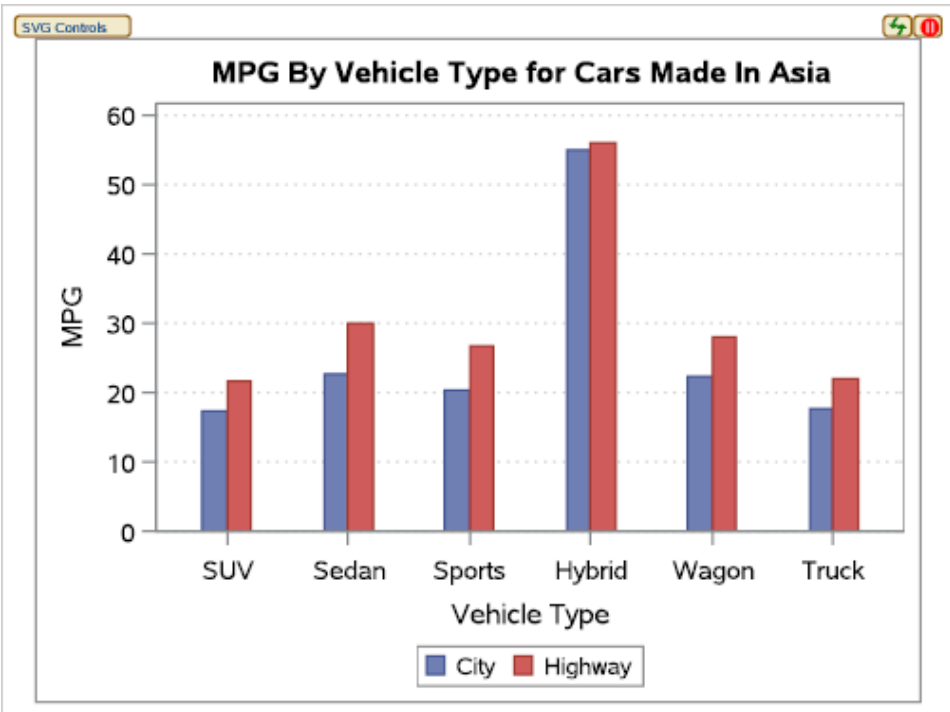
4 After you have generated all of your graphs:

   a Set the ANIMATION=STOP system option to stop the animation output.

   b Close the ODS PRINTER destination.

For more information about creating GIF and SVG animations, see "Creating Animated GIF Images and SVG Documents" in *SAS 9.4 Universal Printing*.

Here is an example that generates an animated graph in the SVG format. The animation shows bar charts of the average city and highway mileage for vehicles made in Asia, Europe, and the USA, in that order. The following figure shows the first graph in the sequence.



Each graph is displayed sequentially. A three-second delay is inserted between each graph. A one-second fade out and fade in is used to transition between graphs. The sequence loops continuously. A BY statement is used in the SGRENDER statement to automatically generate the sequence of graphs for Origin. Because the graphs are generated by Origin, the data must be sorted by Origin. The

special dynamic variable _BYVAL_ is used to display the BY variable value in the graph title. See "Special Dynamic Variables" on page 614.

Here is the SAS code for this example.

```
/* Sort data by origin */
proc sort data=sashelp.cars out=cars;
  by origin;
run;

/* Define the template for the graphs */
proc template;
  define statgraph mileage;
    dynamic _BYVAL_;
    begingraph;
      entrytitle "MPG By Vehicle Type for Cars Made In " _BYVAL_;
      layout overlay / cycleattrs=true
        xaxisopts=(label="Vehicle Type")
        yaxisopts=(label="MPG" griddisplay=on
          gridattrs=(color=lightgray pattern=dot)
          linearopts=(tickvaluesequence=(start=0 end=60
            increment=10) tickvaluepriority=true));
        barchart category=type response=mpg_city /
          name="City" legendlabel="City"
          discreteoffset=-0.1 barwidth=0.2 stat=mean;
        barchart category=type response=mpg_highway /
          name="Highway" legendlabel="Highway"
          discreteoffset=0.1 barwidth=0.2 stat=mean;
        discretelegend "City" "Highway";
      endlayout;
    endgraph;
  end;
run;

/* Create a file reference for the printer output */
filename prtout "anim.svg"; /* Specify the output filename */

/* Set the system animation options */
options printerpath=svg /* Specify the SVG universal printer */
  nonumber nodate /* Suppress the page number and date */
  animduration=3  /* Wait 3 seconds between graphs */
  animloop=yes    /* Play continuously */
  noanimoverlay   /* Display graphs sequentially */
  svgfadein=1     /* One-second fade-in for each graph */
  svgfadeout=1    /* One-second fade-out for each graph */
  nobyline;       /* Suppress the BY-line */

/* Close all currently open ODS destinations */
ods _all_ close;

/* Start the animation output */
options animate=start;

/* Clear the titles and footnotes */
title;
footnote;
```

```
/* Open the ODS PRINTER destination */
ods printer file=prtout style=htmlblue;

/* Generate the graphs */
proc sgrender data=cars template=mileage;
  by origin;
run;

/* Stop the animation output */
options animate=stop;

/* Close the ODS PRINTER destination */
ods printer close;

/* Open an ODS destination for subsequent programs */
ods html; /*Not required in SAS Studio */
```

PART 8

Graphical Output

27

Managing Your Graph's Appearance

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 27.1 *ods listing style=default;*

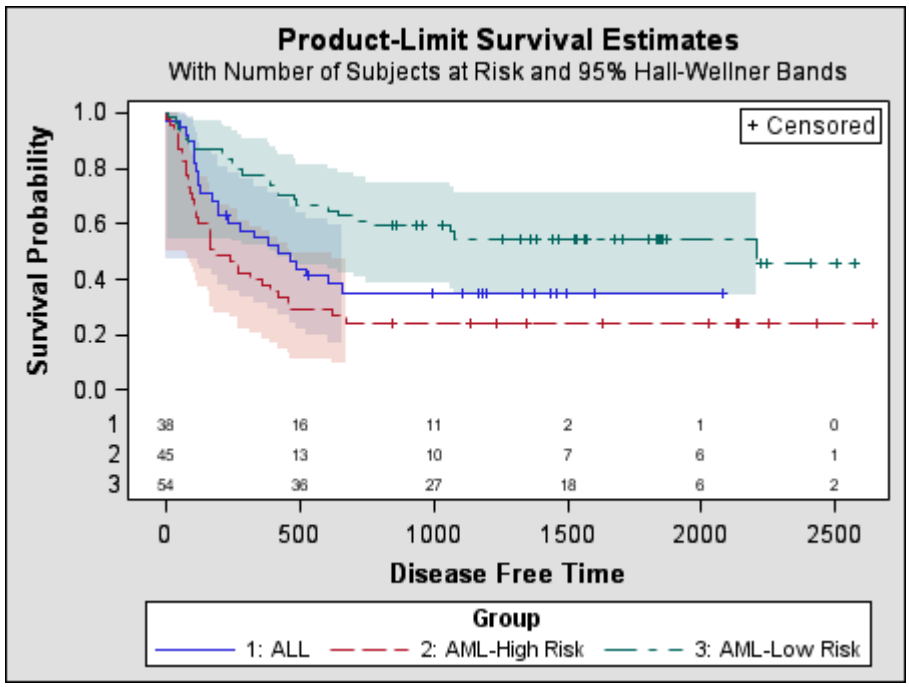


Figure 27.2 *ods listing style=meadow;*

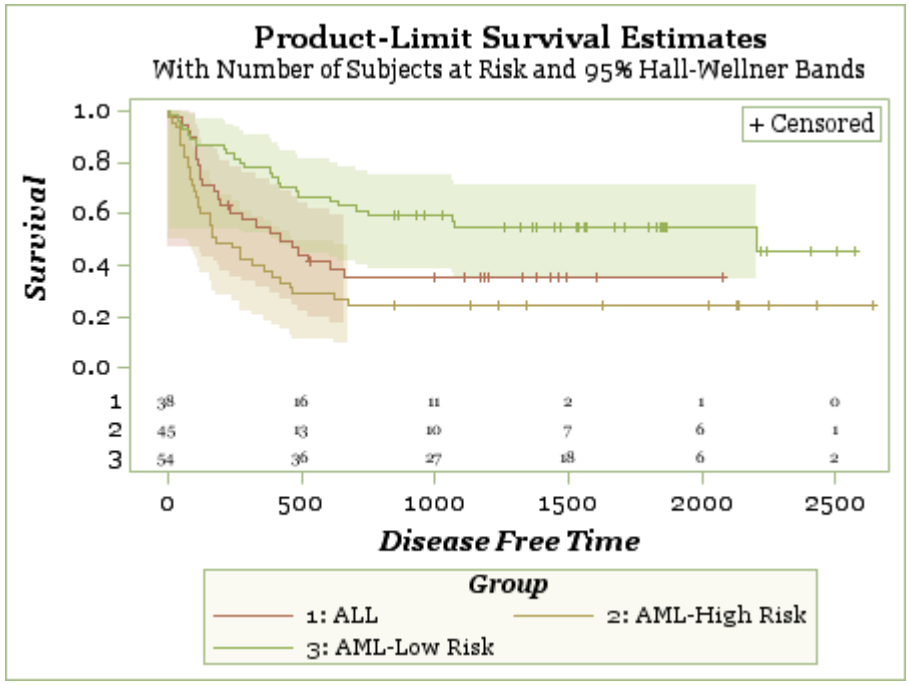
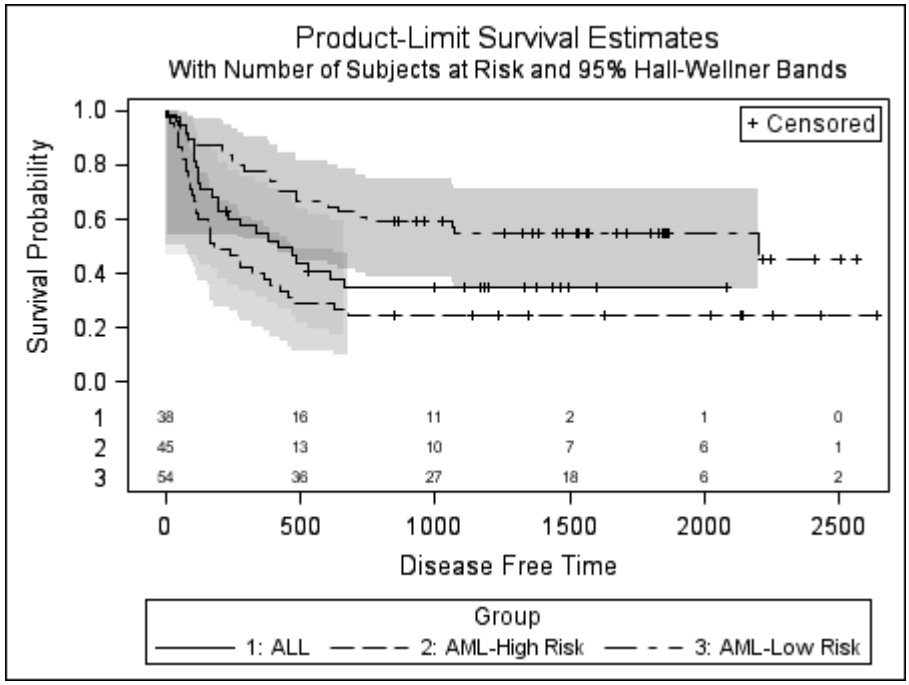


Figure 27.3 *ods listing style=journal;*



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 templates to create the look that you desire in your graphics output. The coding strategy that you use depends on how much style integration you need. If you want to change the appearance of all your plots or apply a custom style to them, you can define your own style. For details, see "Using ODS Styles to Control Graph Appearance" on page 495.

Methods for Changing the Appearance of Your Plots

If you want to change the appearance of your plots from what the supplied styles provide, there are several methods that you can use based on what you want to change. The following table summarizes the available methods.

Attributes to Change	Method	For More Information
The attributes of all plots that use a specific ODS style	Use one of the other supplied ODS styles or create a custom ODS style.	"Using ODS Styles to Control Graph Appearance" on page 495
The attributes of individual plots to make them independent of the current style	Use the appropriate attribute options in the plot statements.	"Options That Override Attributes for Individual Plots" on page 506
The attributes of all of the plots in a specific template to make them independent of the current style	Use the appropriate attribute options in the BEGINGRAPH statement for the template.	"Controlling the Appearance of Grouped Data for All Graphs in a Template" on page 519
The attributes of group values to make them independent of the data order	Use a discrete attribute map or the INDEX= option.	"Making the Appearance of Grouped Data Independent of Data Order" on page 532
The attributes of numeric values based on numeric ranges	Use a range attribute map.	"Using a Range Attribute Map" on page 543

Using ODS Styles to Control Graph Appearance

Evaluating Supplied Styles

Over fifty ODS styles are available for use with ODS Graphics. 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:

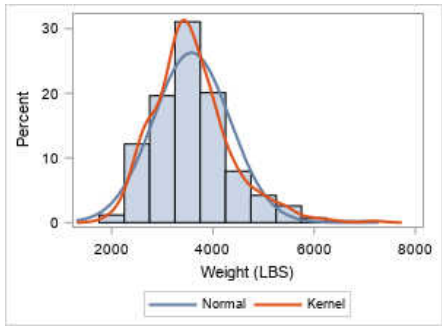
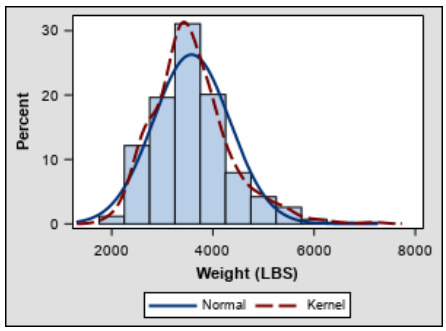
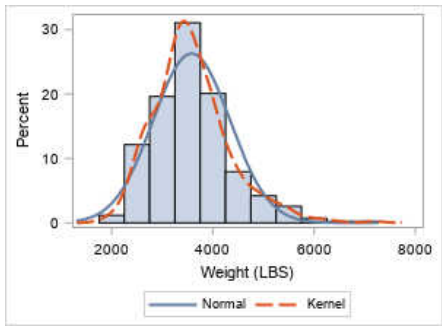
```
proc template;
  path sashelp.tmplmst;
  list styles;
run;
```

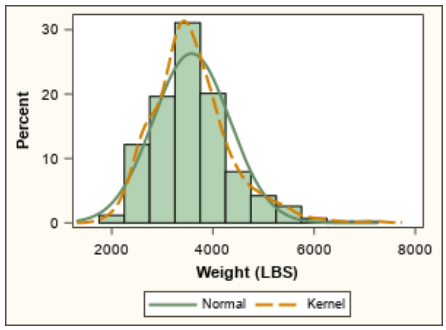
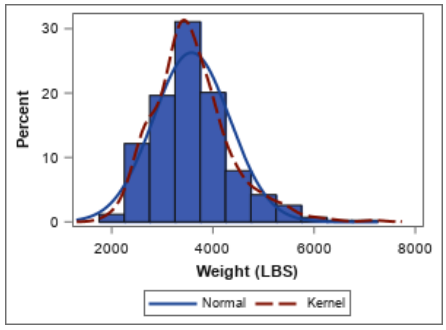
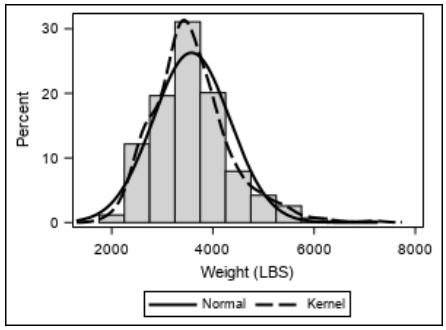
Note: In the SAS windowing environment, you can also browse the styles interactively using the Templates window. To do so, issue the ODSTEMPLATE command to open the Templates window, and then select **Styles** under the Sashelp.Tmplmst item store.

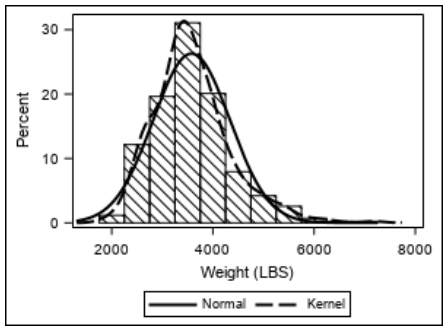
The following table lists the recommended styles and a brief description of each. The example output shown was generated by running the following code:

```
proc template;
   define statgraph histogram;
      begingraph;
         layout overlay;
            histogram weight / primary=true binaxis=false
               LegendLabel="Weight (LBS)";
            densityplot weight / lineattrs=GraphFit normal()
               legendlabel="Normal" name="density1";
            densityplot weight / lineattrs=GraphFit2 kernel()
               legendlabel="Kernel" name="density2";
            discretelegend "density1" "density2";
         endlayout;
      endgraph;
   end;
run;

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

Table 27.1 Recommended ODS Styles

Style	Example
HTMLBLUE [1] ■ white background ■ white wall ■ sans-serif fonts ■ table colors match the graph colors ■ a lighter color scheme for HTML content ■ group distinctions based primarily on color rather than markers, line styles, and fill patterns, when displayed ■ the default style for the ODS HTML destination	
DEFAULT [1] ■ gray background ■ white wall ■ sans-serif fonts ■ group distinctions based on color, line styles, and fill patterns, when displayed	
STATISTICAL [1] ■ white background ■ white wall ■ sans-serif fonts ■ contrasting color scheme of blues, reds, greens, and yellows for markers, lines, fill colors, and fill patterns ■ group distinctions based on color, line styles, and fill patterns, when displayed	

Style	Example
ANALYSIS [1] ■ light tan background ■ white wall ■ sans-serif fonts ■ muted color scheme of tans, greens, yellows, oranges, and browns for lines, markers, fill colors, and fill patterns ■ group distinctions based on color, line styles, and fill patterns, when displayed	
DAISY [1] ■ white background ■ white wall ■ sans-serif fonts ■ a higher contrast color scheme for lines, markers, and fill colors, and fill patterns ■ starting with SAS 9.4M6, group distinctions based on color, line styles, and fill patterns, when displayed prior to SAS 9.4M5, group distinctions based primarily on color rather than marker, line styles, and fill patterns, when displayed ■ recommended for creating accessible graphs (see Using ODS Styles to Create Accessible Output in *Creating Accessible SAS Output Using ODS and ODS Graphics*)	
JOURNAL and JOURNAL3[1, 2] ■ white background ■ white wall ■ sans-serif fonts ■ gray-scale color scheme for fill colors, and black-only color scheme for markers, lines, and fill patterns	

Style	Example
■ group distinctions based on color, line styles, and fill patterns, when displayed ■ for bar charts, gray-scale fill color scheme and back-only fill pattern color scheme by default (JOURNAL3 only)	
JOURNAL2 [2] ■ white background ■ white wall ■ sans-serif fonts ■ gray-scale color scheme for fill colors, and black-only color scheme for markers, lines, and fill patterns ■ group distinctions based on color, line styles, and fill patterns, when displayed ■ starting with SAS 9.4M5, for band plots, bar charts, box plots, ellipse plots, high-low plots, histograms, and polygon plots, black-only color scheme for fill patterns, by default ■ no solid filled areas—a minimal ink style ■ recommended for creating accessible graphs (see Using ODS Styles to Create Accessible Output in *Creating Accessible SAS Output Using ODS and ODS Graphics*)	

1 Supports fill patterns starting with SAS 9.4M5.

2 The JOURNAL2 and JOURNAL3 styles, by default, render grouped bars with fill patterns.

TIP If you want groups to be distinguished by color, marker, and line changes for all ODS styles that use color, specify ATTRPRIORITY=NONE in your BEGINGRAPH statement or in an ODS GRAPHICS statement. See "Attribute Rotation Patterns" on page 545.

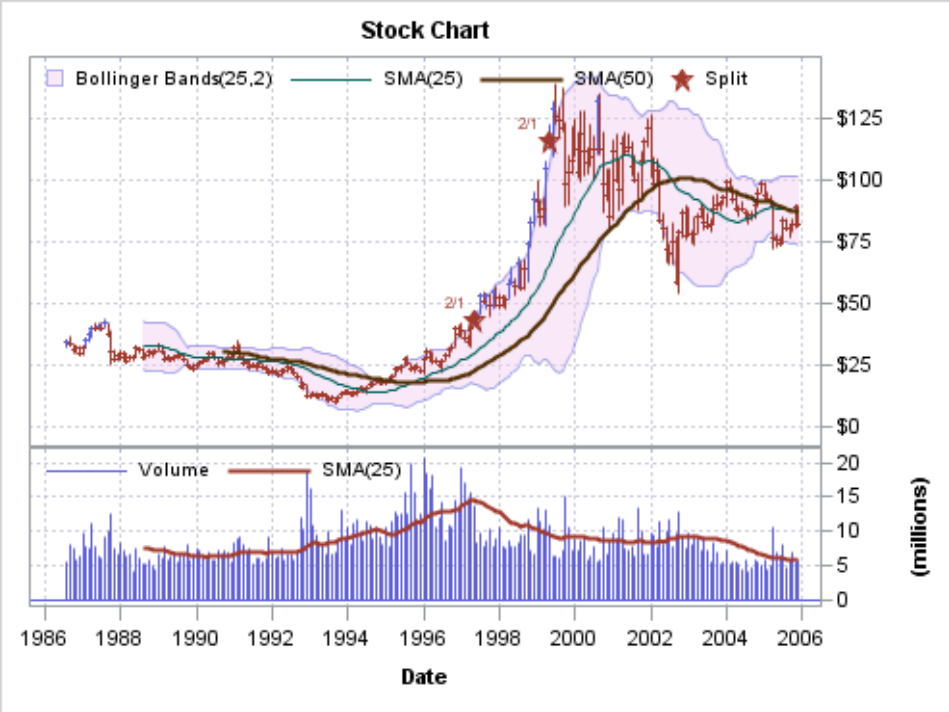
Creating Custom Styles

Reasons for Customizing an ODS Style

A custom style enables you to create graphs with specific visual characteristics that do not have to be hard coded into the GTL code 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, contour plots, and other plot types
- a combination of features that are related to a publication or corporate presentation scheme

By using a custom template, you can create graphs that have a unique appearance, such as the graph shown in the following figure.



A custom style template also enables you to modify the appearance of all graphs that use the template without having to change graph template code for each graph.

Style Elements in ODS Styles

Each ODS style consists of style elements that define the attributes of the style such as colors, fonts, plot markers, and so on. For a list of the style elements that affect ODS Graphics, see Appendix 2, "Graph Style Elements Used by ODS Graphics," on page 651. For additional information about ODS styles, see *SAS Output Delivery System: User's Guide*.

Font Specifications in ODS Styles

Two style elements govern all fonts in an ODS style: Fonts and GraphFonts. The Fonts element governs fonts in tables. The GraphFonts element governs fonts in graphs. Each element contains font attributes that specify the fonts for various text elements. To see the font definition of an ODS style, use the TEMPLATE procedure to write the style template code to the SAS log. Here is an example that displays the style template code for the STATISTICAL style.

```
proc template;
   source styles.statistical;
run;
```

Here is a partial listing of the STATISTICAL style template code that shows the Fonts and GraphFonts style elements and their attributes.

Output 27.1 *Partial Listing of the STATISTICAL Style 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);
   class GraphFonts /
      'NodeDetailFont' = ("<sans-serif>, <MTsans-serif>",7pt)
      'NodeInputLabelFont' = ("<sans-serif>, <MTsans-serif>",9pt)
      'NodeLabelFont' = ("<sans-serif>, <MTsans-serif>",9pt)
      'NodeTitleFont' = ("<sans-serif>, <MTsans-serif>",9pt)
      'GraphDataFont' = ("<sans-serif>, <MTsans-serif>",7pt)
      'GraphUnicodeFont' = ("<MTsans-serif-unicode>",9pt)
      'GraphValueFont' = ("<sans-serif>, <MTsans-serif>",9pt)
      'GraphLabel2Font' = ("<sans-serif>, <MTsans-serif>",10pt)
      'GraphLabelFont' = ("<sans-serif>, <MTsans-serif>",10pt)
      'GraphFootnoteFont' = ("<sans-serif>, <MTsans-serif>",10pt)
      'GraphTitleFont' = ("<sans-serif>, <MTsans-serif>",11pt,bold)
      'GraphTitle1Font' = ("<sans-serif>, <MTsans-serif>",14pt,bold)
      'GraphAnnoFont' = ("<sans-serif>, <MTsans-serif>",10pt);

…
```

Note: The node font attributes in the GraphFonts style element apply to SAS 9.4M1 and to later releases.

The TitleFont attribute in the Fonts style element Output 27.1 on page 501, for example, specifies the fonts for table titles. In style templates, the name of one or more font families 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`>, the font family `sans-serif` is defined in the SAS Registry. You can use the REGISTRY procedure to print the font information in the SAS Registry to the SAS log. See *Base SAS Procedures Guide*.

Note: In the SAS windowing environment, you can use the Registry Editor window to browse the font information in the SAS Registry. To do so, issue the REGEDIT command to open the Registry Editor window, expand **ODS**, and then click **FONTS**.

The registry definition of MTsans-serif and MTserif refer to TrueType fonts that are shipped with SAS and that are similar to Arial and Times New Roman. These MT (monotype) fonts can be used on any computer on which SAS is installed. A specific font family such as Verdana could be used instead.

A font specification in an ODS style can specify multiple fonts so that a reasonable substitution can be made when a font cannot be located on the current computer. The fonts are normally listed in a most-specific to most generic order. The TitleFont

attribute shown in Output 27.1 on page 501, for example, specifies a list of fonts in the following order: <sans-serif>, <MTsans-serif>, Helvetica, and then Helv. For table title text, SAS searches the computer for a font in the TitleFont font list in the order listed. The first font in the list that is found is used.

Several of the graph font style elements affect different features of a graph. The following table shows some features, but not all, that are affected by the graph fonts.

Style Element	Graph Text That It Affects
GraphTitleFont	Used for all titles of the graph. Typically, this is the largest font.
GraphFootnoteFont	Used for all footnotes. Typically, this font is smaller than the title font. Sometimes footnotes are italicized.
GraphLabelFont	Used for axis labels and legend titles. Generally, this font is smaller than the title font.
GraphValueFont	Used for axis tick values and legend entries. Generally, this font is smaller than the label font.
GraphDataFont	Used for text where minimum size is necessary (such as for point labels).
GraphUnicodeFont	Used for adding special glyphs (for example, α , ±, €) to text in the graph (see Chapter 19, "Adding Titles, Footnotes, and Text Entries to Your Graph," on page 317).
GraphAnnoFont	Default font for text added as an annotation in the ODS Graphics Editor.

Example: Customize the Fonts in the STATISTICAL Style

The recommended styles listed in Table 27.1 on page 496 use sans-serif fonts. This example shows you how to create a custom style that uses serif fonts instead of sans-serif fonts. Although you can create a style template from scratch, it is simpler to identify a style that is close to what you want and make limited changes to it. This example uses the STATISTICAL style as the starting point (parent) for the custom style. (See Output 27.1 on page 501.)

To create a custom style from the STATISTICAL style:

1. Name your modified style template `SerifStatistical`.
2. Specify STYLES.STATISTICAL as the parent style.

Note: By assigning the parent to STYLES.STATISTICAL, you need to change only the Fonts and GraphFonts style elements. The rest of the style elements are inherited from the STATISTICAL style.

3 In the Fonts element, make the following changes:

- Change all occurrences of <sans-serif> to <serif>.
- Change all occurrences of <MTsans-serif> to <MTserif>.
- Change Helvetica and Helv (sans-serif fonts) to Times (a serif font).

These changes affect the table fonts.

4 In the GraphFonts element, repeat the changes in Step 3 on page 503.

> **TIP** When you change fonts in a style, be sure to make consistent changes to the Fonts and GraphFonts elements.

These changes affect the graph fonts. The completed SerifStatistical template is shown in Example Code 27.1 on page 503.

5 Submit your template code. The following note appears in the SAS log.

```
NOTE: STYLE 'Styles.SerifStatistical' has been saved to:
SASUSER.TEMPLAT
```

Example Code 27.1 *Completed SerifStatistical Style Template*

```
proc template;
   define style Styles.SerifStatistical;
     parent = styles.statistical;
     style fonts from 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 from GraphFonts /
       'NodeDetailFont'=("<serif>, <MTserif>",7pt)
       'NodeInputLabelFont'=("<serif>, <MTserif>",9pt)
       'NodeLabelFont'=("<serif>, <MTserif>",9pt)
       'NodeTitleFont'=("<serif>, <MTserif>",9pt)
       '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: The node font attributes in the GraphFonts style element apply to SAS 9.4M1 and to later releases.

Testing Your Modified Style Template

After you submit your modified template, you can run a test program that uses your modified style to verify that your changes are correct. Here is a SAS program that tests the style template that was created in "Example: Customize the Fonts in the STATISTICAL Style" on page 502.

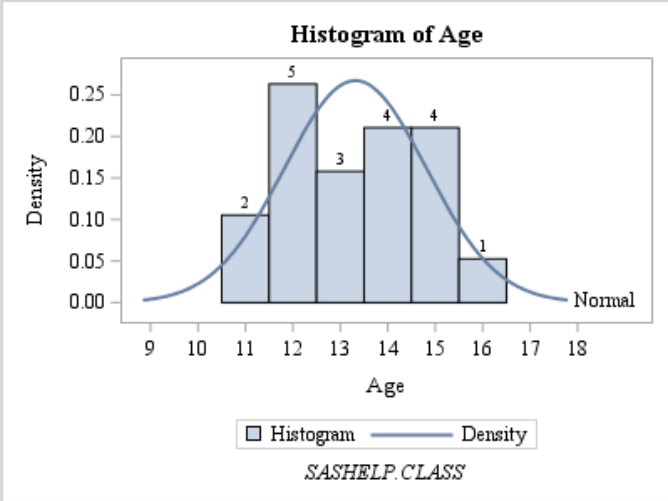
```
/* Open an HTML destination using the modified style */
ods html close;
ods html style=SerifStatistical;

/* Print the first 5 observations in SASHELP.CLASS */
proc print data=sashelp.class(obs=5);
run;

/* Create a histogram from variable Age */
proc template;
  define statgraph histogram;
    begingraph;
      entrytitle "Histogram of Age";
      entryfootnote "SASHELP.CLASS";
      layout overlay /
        xaxisopts=(label="Age");
        histogram age / name="Histogram"
           scale=density binwidth=1 datalabeltype=count;
        densityplot age / name="Density" normal()
           curvelabel="Normal";
        discretelegend "Histogram" "Density";
      endlayout;
    endgraph;
  end;
run;

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

ods html close;
```

The following figure shows the output.

The SAS System

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5

The SAS System



Notice that serif fonts are used in the table and graph text, which is correct.

Making Your Modified Style Template Available to Others

When you submit your template code, the template is stored by default in the first location in your ODS search path that has Write permission. You can run the following statement to see your current ODS search path.

```
ods path show;
```

Here is an example of the output.

```
Current ODS PATH list is:
1. SASUSER.TEMPLAT(UPDATE)
2. SASHELP.TMPLMST(READ)
```

The example output shows that the first location, SASUSER.TEMPLAT, has Write permission (UPDATE). Your default Sasuser.Templat location is available to you only, which is appropriate for templates that are intended for your personal use. If you want to make your template available to others, do the following:

1. Select a directory that is accessible to other users. This example uses the path **u:\ODS_templates**.
2. Use the ODS PATH statement to modify the ODS search path to include the path that you selected in Step 1 on page 506:

   ```
   libname common "u:\ODS_templates";
   ods path common.dept(update)
     sasuser.templat(update)
     sashelp.tmplmst(read);
   ```

   The first item store (Common.Dept) can be updated and will contain the new template (Styles.SerifStatistical). You must include Sashelp.Tmplmst in the path because the inherited parent style (Styles.Default) is in Sashelp.
3. Submit your template code again. A note similar to the following appears in the SAS log:

```
NOTE: STYLE 'Styles.SerifStatistical' has been saved to:
COMMON.DEPT
```

For other users to access your style template, they must include the following code at the start of the SAS programs:

```
libname common "path-to-your-style-template" access=readonly;
ods path sasuser.templat(update)
  common.dept(read)
  sashelp.tmplmst(read);
```

The code creates a library for the path in which you stored your modified style template, and then adds that library to the ODS search path after Sasuser.Templat.

Using Options to Override Style Attributes

Options That Override Attributes for Individual Plots

Overview of Graphical Properties

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 option in the layout statement:

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

The syntax for setting appearance options is similar. For example, in statements such as SERIESPLOT, DENSITYPLOT, REFERENCELINE, and DROPLINE, you use the LINEATTRS= option to specify the appearance of the line:

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

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

In BANDPLOT and MODELBAND statements, you use the following option to specify the appearance of the outline:

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

In a BOXPLOT statement, you use the following options to specify the appearance of the whiskers and median:

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

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

In BARCHART and BARCHARTPARM statements, you use the following options to specify the appearance of the bar fill color and pattern:

FILLATTRS=(COLOR=*color* TRANSPARENCY=*number*)

FILLPATTERNATTRS=(COLOR=*color* PATTERN=*pattern*)

The list of properties in each of these options is sometimes called an "attribute bundle."

Some GTL statements provide other options that override the visual attributes of graphical features for an individual graph. For more information, see *SAS Graph Template Language: Reference*. The remainder of this section describes the most commonly used options.

LINEATTRS Option

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

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

For information about the attributes that you can specify with *line-options*, see “Line Options” on page 666.

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?

Here is the GraphReference style element definition in the DEFAULT style template.

```
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 "Available Line Patterns" on page 670 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.

When selecting a different style element for the LINEATTRS= option, make sure that the style element sets 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 type of assignment, you really do not 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 are identical in terms of line properties.

You might want the reference lines to be somewhat like a style element, but different. This requires an override. Here are some examples of style overrides:

- The following example makes the reference line look like a grid line (color and pattern), but it is thicker (assuming most styles define grid lines as 1px):

  ```
  LINEATTRS=GraphGridLines(THICKNESS=2px)
  ```

- The following example makes the reference line look like an axis line (color and thickness), but it uses the DASH line pattern:

  ```
  LINEATTRS=GraphAxisLines(PATTERN=DASH)
  ```

- The following example makes the reference line look like a reference line (pattern and thickness), but it has the color of axis lines:

  ```
  LINEATTRS=GraphReference(COLOR=GraphAxisLines:ContrastColor)
  ```

- The following example is a shorter form of the previous example:

```
LINEATTRS=(COLOR=GraphAxisLines:ContrastColor)
```

Anytime you do not supply a style element or do not override all the suboptions, the suboptions not overridden come from the default style references.

- The following example shows how you can hardcode visual properties:

```
LINEATTRS=(COLOR=BLUE)
```

This technique is a straightforward way of getting what you want. The results might look good with the current ODS style, but it might not look as good when a different style is used.

For a complete list of all of the style elements and attributes and their default values, see Appendix 2, “Graph Style Elements Used by ODS Graphics,” on page 651.

MARKERATTRS Option

Much of what is said about line properties in “LINEATTRS Option” on page 507 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*)

For information about the attributes that you can specify with *marker-options*, see “Marker Options” on page 668.

The following table shows how the 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		TRANSPARENCY	transparency specification, which ranges from 0 (opaque) to 1 (entirely transparent)
none		WEIGHT	NORMAL or BOLD

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 19, "Adding Titles, Footnotes, and Text Entries to Your Graph," on page 317.

Options That Override Style Attributes for All of the Plots in a Template

GTL provides options that you can use to override style attributes for all plots in a template. These options are specified in the BEGINGRAPH statement for the template. The following table lists these options.

Option	Description
ATTRPRIORITY=AUTO | COLOR | NONE	Specifies a priority for cycling the group attributes. When COLOR is in effect, a color-priority rotation pattern is used instead of the default GraphData1–GraphDataN rotation pattern. See "The Color-Priority Attribute Rotation Pattern" on page 549.
DATACOLORS=(*color-list*)	Specifies the list of fill colors that replace the graph data colors from the GraphData1–GraphDataN style elements.

Option	Description
DATACONTRASTCOLORS=(*color-list*)	Specifies the list of contrast colors that replace the graph data contrast colors from the GraphData1–GraphDataN style elements.
DATALINEPATTERNS=(*line-pattern-list*)	Specifies the list of line patterns that replace the graph data line patterns from the GraphData1–GraphDataN style elements.
DATASKIN=NONE | CRISP | GLOSS | MATTE | PRESSED | SHEEN	Enhances the visual appearance of all plots in the template that support data skins. See "Using Data Skins" on page 554.
DATASYMBOLS=(*marker-symbol-list*)	Specifies a list of marker symbols that replace the graph data marker symbols from the marker symbols that are defined in the GraphData1–GraphDataN style elements.

For more information about these BEGINGRAPH statement options, see "BEGINGRAPH" in *SAS Graph Template Language: Reference*.

Controlling the Appearance of Non-grouped Data

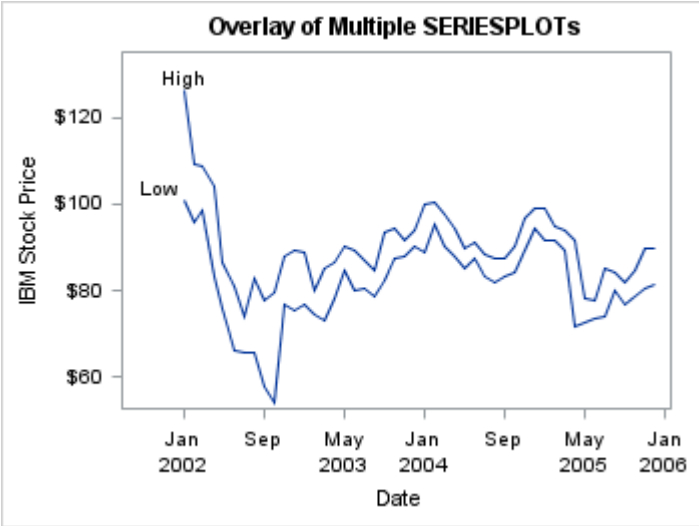
When you use statements such as SERIESPLOT, BANDPLOT, NEEDLEPLOT, ELLIPSE, STEPPLOT, FRINGEPLOT, LINEPARM, and VECTORPLOT to draw plots containing lines, the same style element, GraphDataDefault, is used for all line and marker properties. You can think of these plots as "non-specialized," and they all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

The following example produces series lines that 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;
```

Here is the output.

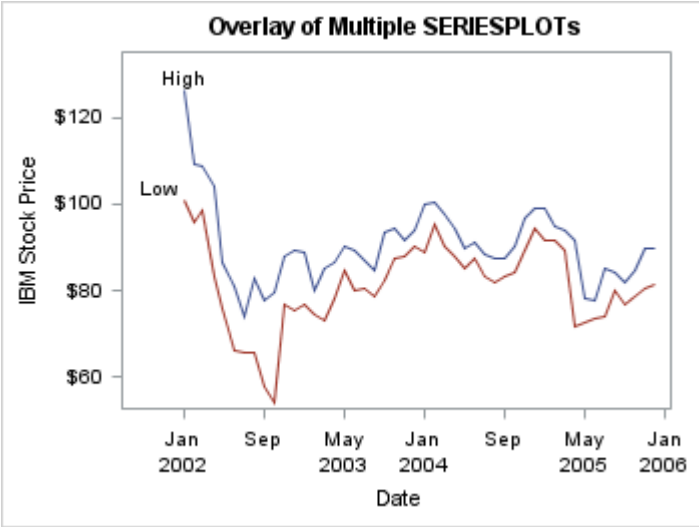


To ensure that the series lines differ in appearance, you can use any style element with line properties. A set of carefully constructed style elements named GraphData1–GraphDataN (where N=12 for most styles, some styles might have fewer) are normally used for this purpose. These elements all use different marker symbols, line pattern, fill colors (COLOR=) and line and marker colors (CONTRASTCOLOR=). All line and 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 layout block, 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;
```

Here is example output.

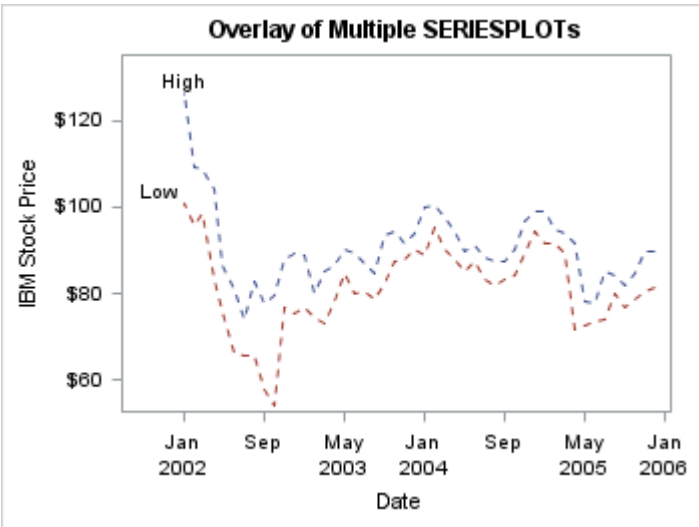


This same graph could also be achieved by specifying CYCLEATTRS=TRUE on the LAYOUT OVERLAY statement and omitting the LINEATTRS= options on the plot statements. In that case, the visual attributes for each plot in the overlay are rotated as described in "Attribute Rotation Patterns" on page 545. The GraphDataN style elements provide 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 that you want to hold constant as shown in the following layout block.

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

Here is example output.



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 are differences in appearance.

Controlling the Appearance of Grouped Data

Plots That Support Grouped Data

The GROUP= *column* option is used to plot data when a classification or grouping variable is available. Plots that support the GROUP= option include the following:

AXISTABLE	HEATMAPPARM	PIECHART
BANDPLOT	HIGHLOWPLOT	POLYGONPLOT
BARCHART	LINECHART	REGRESSIONPLOT
BARCHARTPARM	LINEPARM	SCATTERPLOT
BOXPLOT	LOESSPLOT	SCATTERPLOTMATRIX
BOXPLOTPARM	MODELBAND	SERIESPLOT
BUBBLEPLOT	MOSAICPLOTPARM	STEPPLOT
ELLIPSEPARM	NEEDLEPLOT	VECTORPLOT
FRINGEPLOT	PBSPLINEPLOT	WATERFALLCHART

Starting with SAS 9.4M2, the DENSITYPLOT, ELLIPSE, and HISTOGRAM statements also support the GROUP= option.

Using the Default Appearance for Grouped Data

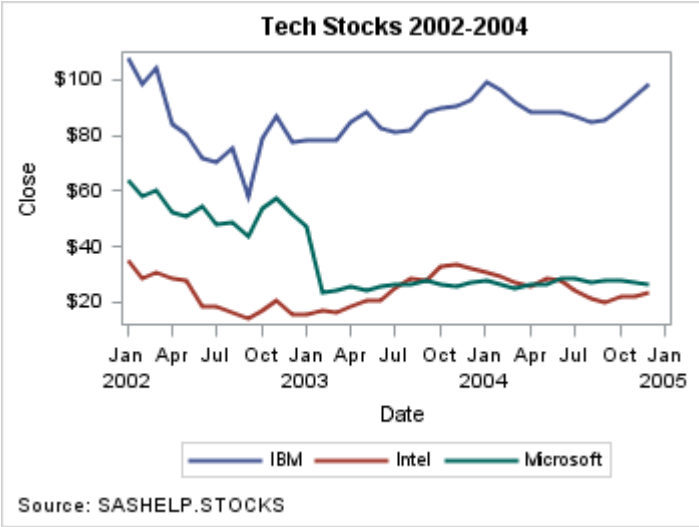
By default, the GROUP= option automatically uses the style elements GraphData1–GraphDataN for the presentation of each unique group value. Consider the following template for a series plot that displays grouped data.

Example Code 27.2 *Template for a Grouped Series Plot*

```
proc template;
  define statgraph groupedseries;
    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=groupedseries;
    where date between "1jan02"d and "31dec04"d;
  run;
```

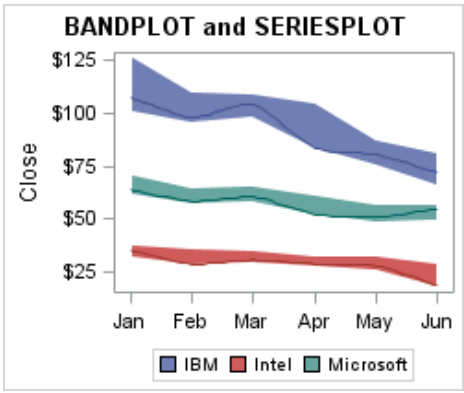
Executing this template with the ODS style HTMLBLUE generates the following output.

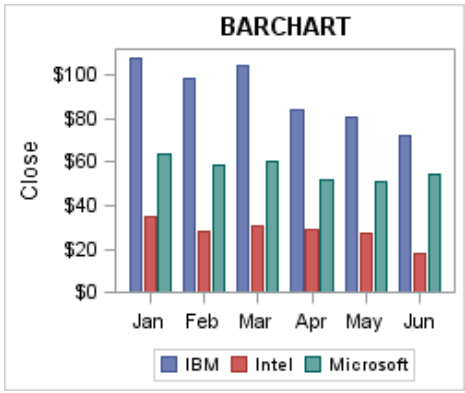


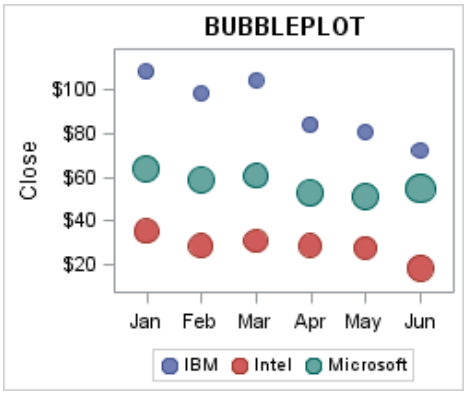
Attributes such as line color and pattern are used to display the group values. The attributes are defined by the GraphData1–GraphDataN style elements for the style that is in effect. Each group value is assigned attributes from a GraphData1–GraphDataN style element sequentially (1 to N). When a SAS data set is plotted, the group-value attributes are assigned in data order. When a CAS in-memory table is plotted, the group-value attributes are assigned in ascending order of the group column character or unformatted numeric values. For more information about attribute rotation, see “Attribute Rotation Patterns” on page 545.

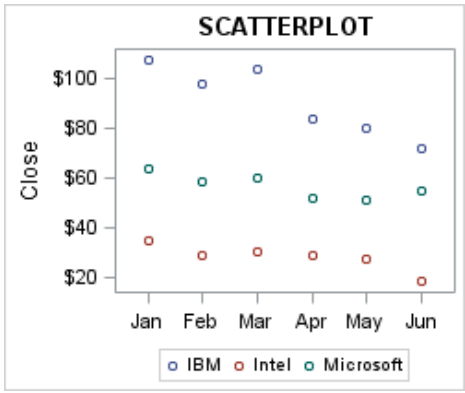
In the previous example, there are three unique values of column Stock in the Sashelp.Stocks data set: IBM, Intel, and Microsoft. The line colors and line patterns from the GraphData1–GraphData3 style elements of the HTMLBLUE style are used for each of the three group values. Because the HTMLBLUE style defines the AttrPriority="COLOR" style attribute, the color-priority attribute rotation pattern is used. The ContrastColor attribute specifies the line color, and the LineType attribute specifies the line pattern. See Graph Style Elements Used by ODS Graphics for information about the GTL style elements and attributes. Attribute See “Attribute Rotation Patterns” on page 545 for information about how the style attributes are rotated for group values.

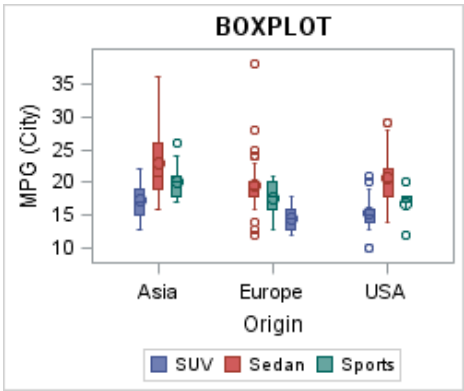
The colors and patterns are assigned to the values in the order in which they occur in the Sashelp.Stocks data set. In this case, GraphData1 is assigned to IBM, GraphData2 is assigned to Intel, and GraphData3 is assigned to Microsoft. Other attributes such as fill color, fill pattern, and marker color that are used in other plot types are assigned in a similar manner. Here are some additional examples of the default grouped data appearance for other plot types. All of the plots in these examples use the HTMLBLUE style.

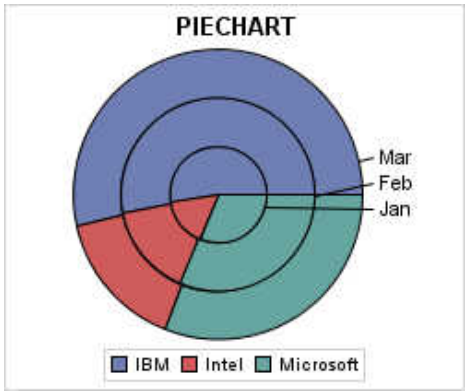












To specify different colors and patterns that you can specify a different ODS style or you can create a custom style. For information about creating custom styles, see "Using Custom Styles to Control the Appearance of Grouped Data" on page 517. For many plots and charts, you can also use attribute maps to override certain style attributes for specific group values. For information about attribute maps, see "Using a Discrete Attribute Map to Control the Appearance of Grouped Data" on page 518.

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 template 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.

Example Code 27.3 *Custom Style STOCKS*

```
proc template;
  define style Styles.stocks;
    parent=styles.listing;
    style GraphData1 from GraphData1 /
      ContrastColor=blue
      Color=blue
      MarkerSymbol="CircleFilled"
      Linestyle=1;
    style GraphData2 from GraphData2 /
      ContrastColor=brown
      Color=brown
      MarkerSymbol="TriangleFilled"
      Linestyle=1;
    style GraphData3 from GraphData3 /
      ContrastColor=orange
      Color=orange
      MarkerSymbol="SquareFilled"
      Linestyle=1;
  end;
run;
```

In this style template, 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 “Available Line Patterns” on page 670 for the complete set of line patterns.

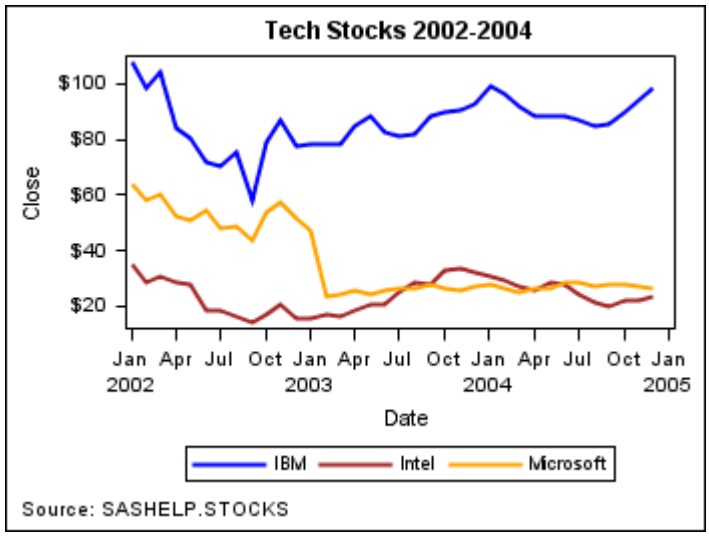
CONTRASTCOLOR is the attribute applied to grouped lines and markers. COLOR is the attribute applied to grouped filled areas, such as grouped bar charts or grouped ellipses. MARKERSYMBOL defines the same values that can be specified with the MARKERATTRS=(SYMBOL=*keyword*) option in GTL. See “SYMBOL=*style-reference* | *marker-name* ” on page 668 for the complete set of marker symbol names.

After the STOCKS style is defined, it must be requested on the ODS destination statement. No modification of the compiled template is necessary. Here is an example that uses the STOCKS style with the template in Example Code 27.2 on page 514.

```
/* Specify a path for the ODS output */
filename odsout "output-path";
ods _all_ close;
ods graphics / reset imagename="stocks";
```

```
ods html file="stocks.html" path=odsout style=stocks;

proc sgrender data=sashelp.stocks
  template=groupedseries;
  where date between "1jan02"d and "31dec04"d;
run;
ods html close;
ods html; /* Not required in SAS Studio */
```

Here is the output.



One issue that 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–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.

Using a Discrete Attribute Map to Control the Appearance of Grouped Data

You can use a discrete attribute map to specify the appearance of grouped data. For more information, see "Using a Discrete Attribute Map" on page 538.

Controlling the Appearance of Grouped Data for All Graphs in a Template

Options That Override Style Attributes for All of the Plots in a Template

You can use options in the BEGINGRAPH statement to override various style attributes for all of the plots that are defined in a template. The following table lists the options that you can use.

BEGINGRAPH Option	Description
ATTRPRIORITY=	Specifies a priority for the cycling of the attributes from the GraphData1–GraphDataN style elements. See "Attribute Rotation Patterns" on page 545.
DATACOLORS=	Specifies the list of fill colors that replace the graph data colors from the GraphData1–GraphDataN style elements.
DATACONTRASTCOLORS=	Specifies the list of contrast colors that replace the graph data contrast colors from the GraphData1–GraphDataN style elements.
DATASYMBOLS=	Specifies the list of marker symbols that replace the graph data marker symbols from the marker symbols that are defined in the GraphData1–GraphDataN style elements.
DATALINEPATTERNS=	Specifies the list of line patterns that replace the graph data line patterns from the GraphData1–GraphDataN style elements.

These options apply to all of the plots that are defined in the template. They affect the attribute assignments when the following criteria are true:

- The GROUP= option is used on a plot statement.
- Attribute options such as LINEATTRS= or MARKERATTRS= reference a graphData1–GraphDataN style element.
- The CYCLEATTRS=TRUE option is in effect in an overlay-type layout.

You can override the attributes that are defined by these options on a per-plot basis by using attribute options or a reference to an attribute map on the plot statement.

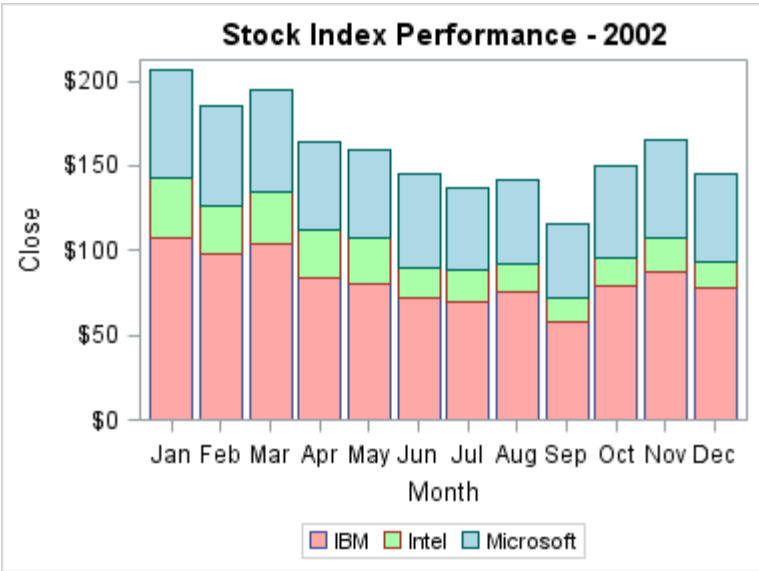
Overriding the Fill Colors

You can use the DATACOLORS=(*color-list*) option to override the fill colors that are defined in the current style's GraphData1–GraphDataN style elements. When you specify the DATACOLORS= option, the fill colors rotate through the colors that are specified in the color list rather than through the colors that are specified in the current style's GraphData1–GraphDataN style elements. This rotation pattern enables you to use the same colors in your plots regardless of the style that is in effect.

Here is an example that specifies the fill colors very light red, very light green, and light blue as the colors for the three group values of a grouped bar chart.

```
proc template;
  define statgraph groupedbar;
    dynamic year;
    begingraph / datacolors=(verylightred verylightgreen lightblue);
      entrytitle "Stock Index Performance - " year;
      layout overlay / xaxisopts=(label="Month");
        barchart category=date response=close / group=stock
name="barchart";
        discretelegend "barchart";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.stocks template=groupedbar;
  where date between "1jan02"d and "31dec02"d;
  format date MONNAME3.;
  dynamic year="2002";
run;
```

The following figure shows the result.



If the number of unique colors needed exceeds the number of colors that are specified in the color list, two new sets of colors are computed based on the original colors. The first set is one shade lighter that the original colors, and the second set is one shade darker. In the previous example, this pattern provides a maximum of nine unique colors. Beyond nine values, the color pattern repeats, starting with the 10th value.

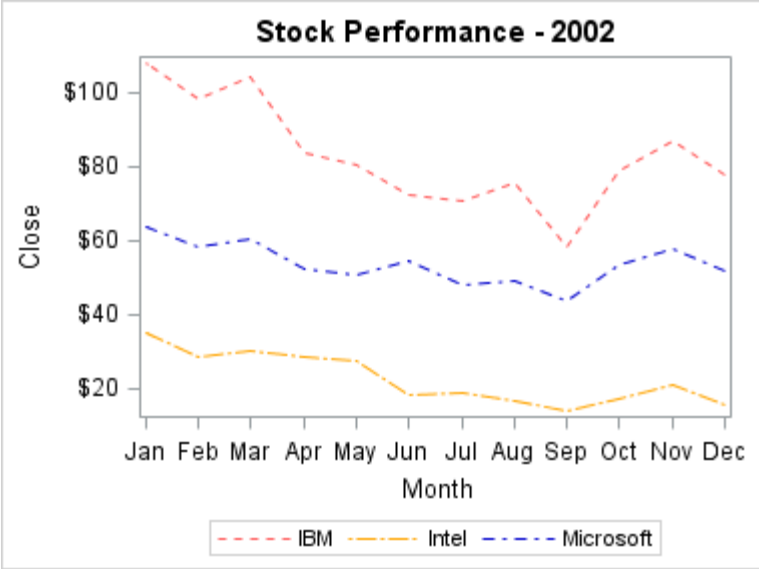
Overriding the Line Patterns and Line Colors

You can use the DATACONTRASTCOLORS= and DATALINEPATTERNS= options to override the line patterns and line colors that are defined in the current style's GraphData1–GraphDataN style elements. In this way, you can use the same colors and line patterns in your plots regardless of the style that is in effect. Here is an example that specifies the contrast colors light red, orange, and medium blue as the colors for the three group values of a grouped series plot. It also specifies the line patterns 2, 9, and 41 as the line patterns for the group values. (See "Available Line Patterns" on page 670.) This example uses the HTMLBLUE style. In order to use a different line pattern for each group value, the code must also include the ATTRPRIORITY=NONE option to ensure that color does not take priority in the attribute rotation.

Here is the SAS code for this example.

```
proc template;
  define statgraph groupedseries2;
    dynamic year;
    begingraph / attrpriority=none
                 datacontrastcolors=(lightred orange mediumblue)
                 datalinepatterns=(2 9 41);
      entrytitle "Stock Performance - " year;
      layout overlay / xaxisopts=(type=discrete label="Month");
        seriesplot x=date y=close / group=stock name="series";
        discretelegend "series";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.stocks template=groupedseries2;
  where date between "1jan02"d and "31dec02"d;
  format date MONNAME3.;
  dynamic year="2002";
run;
```

The following figure shows the result.



The colors and line patterns remain constant regardless of the style that is in effect.

Overriding the Marker Symbols and Marker Colors

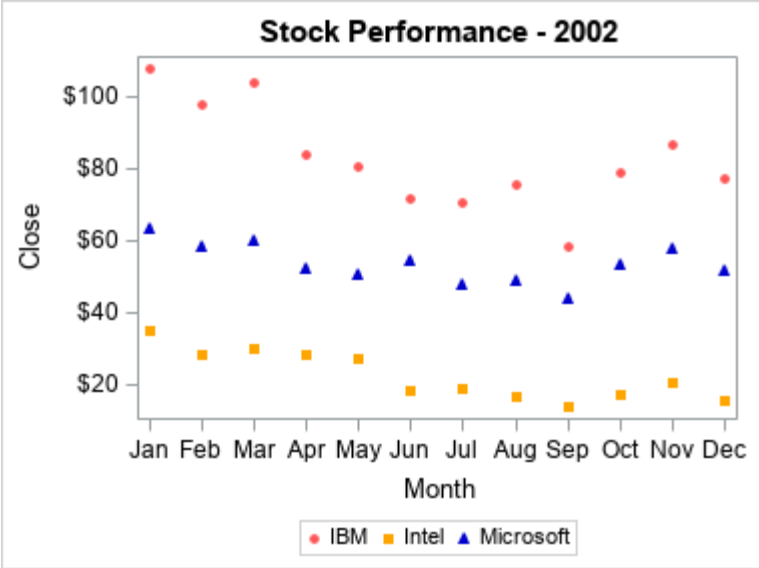
You can use the DATACONTRASTCOLORS= and DATASYMBOL= options to override the marker symbols and marker colors that are defined in the current style's GraphData1–GraphDataN style elements. In this way, you can use the same marker symbols in your plots regardless of the style that is in effect. Here is an example that specifies the marker symbols circle, square, and triangle for the three group values of a grouped scatter plot. This example uses the HTMLBLUE style. In order to use a different marker symbol for each group value, the code must also include the ATTRPRIORITY=NONE option to ensure that color does not take priority in the attribute rotation.

Here is the SAS code for this example.

```
proc template;
  define statgraph groupedscatter;
    dynamic year;
    begingraph / attrpriority=none
                 datasymbols=(circlefilled squarefilled trianglefilled)
                 datacontrastcolors=(lightred orange mediumblue);
      entrytitle "Stock Performance - " year;
      layout overlay / xaxisopts=(type=discrete label="Month");
        scatterplot x=date y=close / group=stock name="scatterplot";
        discretelegend "scatterplot";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.stocks template=groupedscatter;
  where date between "1jan02"d and "31dec02"d;
  format date MONNAME3.;
```

```
    dynamic year="2002";
  run;
```

The following figure shows the result.



The marker symbols remain constant regardless of the style that is in effect.

Changing the Grouped Data Display

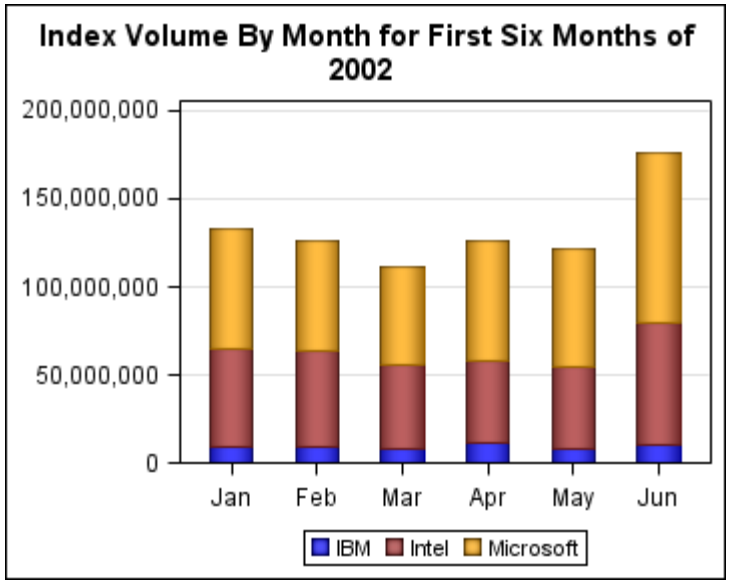
With many plots, you can use the GROUPDISPLAY= option to change how group values are displayed. The following table summarizes the values that you can use with this option and the plot statements that are applicable for each.

Table 27.2 *GROUPDISPLAY Option Values and the Applicable Plots for Each*

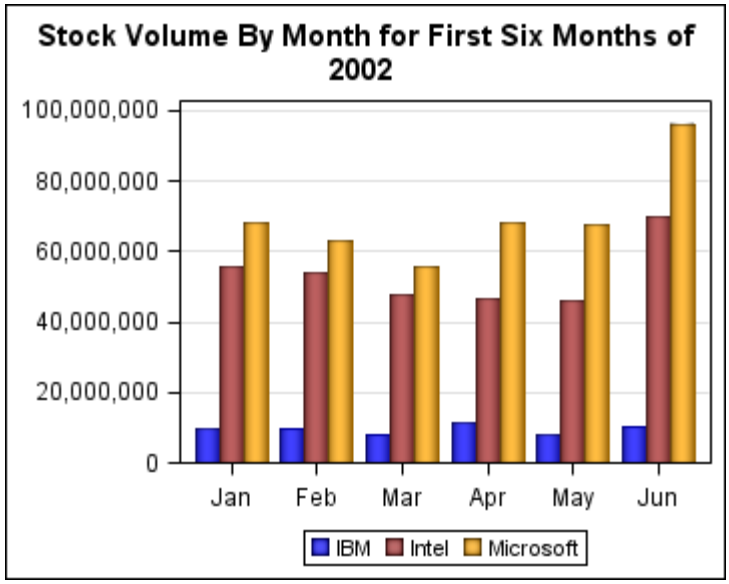
Value	Applicable Plot Statements	Description
STACK	BARCHART BARCHARPARM	Stacks each group value on a single bar at the category value on the axis.
CLUSTER	BARCHART BARCHARPARM BOXPLOT BOXPLOTPARM NEEDLEPLOT SCATTERPLOT SERIESPLOT STEPPLOT	Displays the group values side-by-side in a cluster that is centered on the category value on the axis. **Note:** For NEEDLEPLOT, SCATTERPLOT, SERIESPLOT, and STEPPLOT, the X axis must be categorical. **Note:** For SCATTERPLOT, SERIESPLOT, and STEPPLOT, you can use the CLUSTERAXIS= option to specify cluster grouping on the Y axis, when applicable.

Value	Applicable Plot Statements	Description
OVERLAY	BOXPLOT BOXPLOTPARM NEEDLEPLOT SCATTERPLOT SERIESPLOT STEPPLOT	Overlays the group values at the category value on the axis.

For bar charts, by default, group values are stacked on each category bar. The following figure shows an example using the STOCKS style that was created in Example Code 27.3 on page 517.



When GROUPDISPLAY=CLUSTER, the group values are shown as a cluster of bars, one bar for each group value in the category value, centered over the category value as shown in the following figure.



Here is the code that generated the previous plot.

```
/* Create a variable for the desired year. */
%let year=2002;

/* Specify a path for the ODS output */
filename outp "output-path";

/* Create a data set of the first six months of the year. */
data stocks;
  set sashelp.stocks;
  where year(date) eq &year and month(date) le 6;
  month=month(date);
run;

/* Format the numeric months into 3-character month names. */
proc format;
  value month3char
    1="Jan" 2="Feb" 3="Mar" 4="Apr" 5="May" 6="Jun";
run;

/* Create the template. */
proc template;
  define statgraph stocksgraph;
    begingraph;
      dynamic year;
      entrytitle "Stock Volume By Month for First Six Months of " year;
      layout overlay /
        yaxisopts=(griddisplay=on display=(line ticks tickvalues))
        xaxisopts=(display=(line ticks tickvalues));
        barchart category=month response=volume /
          name="total"
          dataskin=pressed
          group=stock
          groupdisplay=cluster;
        discretelegend "total";
      endlayout;
    endgraph;
```

```
  end;
run;

/* Generate the bar chart using the bar template. */
ods graphics on / reset outputfmt=static imagename="stocks_&year";;
ods _all_ close;
ods html file="stocks_&year..html" path=outp style=stocks;
proc sgrender data=stocks template=stocksgraph;
  dynamic year=&year;
  format month month3char.;
run;
ods html close;
ods html; /* Not required in SAS Studio */
```

The STOCKS ODS style defines the graph appearance. (See Example Code 27.3 on page 517.) The width of each cluster is directly based on the number of category values on the axis. By default, the cluster width is 85% of the midpoint spacing. You can use the CLUSTERWIDTH= option to adjust the cluster width.

Within each cluster, by default, each bar occupies 100% of its available space. When a cluster contains the maximum number of bars, no gap exits between the adjacent bars in the cluster. You can use the BARWIDTH= option to add space between the bars in the cluster.

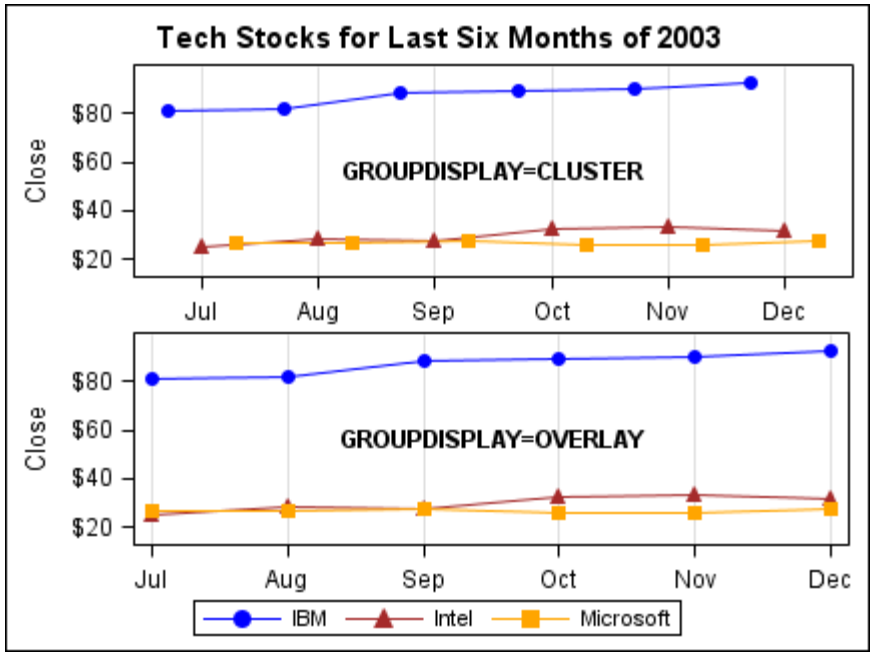
For the remaining plot types (see Table 27.2 on page 523), the behavior of the GROUPDISPLAY=CLUSTER option is similar to that of bar charts. That is, the group values are clustered at the category value on the axis. You can also use the CLUSTERWIDTH= option to vary the width of the clusters. For these plots, the clusters include the following:

- plot markers for series plots and scatter plots
- step transitions and plot markers for step plots
- boxes for box plots

This is useful if you want to overlay a grouped SERIESPLOT onto a grouped clustered BARCHART, for example.

When GROUPDISPLAY=OVERLAY is used, each group value for a category is positioned at the category value on the axis. If one or more values appear in the same position, the symbol for the last value overlays the symbol for the previous value. The following figure shows series plot cluster and overlay group displays together so that you can compare the two.

Note: Style STOCKS was used to generate the series plots. See Example Code 27.3 on page 517.



Notice in the cluster display that the three-symbol cluster for each category is centered on the category value, while in the overlay display, all of the symbols are aligned on the category value. Also notice that in the overlay display some of the symbols overwrite others that appear in the same location. In the case of a scatter plot, the plot symbols behave in the same manner.

Including Missing Group Values

By default, missing group values are excluded from the plot. If you want to include a group for the missing values, use the INCLUDEMISSINGGROUP=TRUE option in your plot statement. If you include a discrete legend, the missing group is also added to the legend. For numeric values, the label for the missing group in the legend is a dot or the character that is specified by the SAS MISSING option. For character values, the label for the missing group is a blank. You can use a FORMAT statement to assign a more meaningful label to the missing group category. For example, consider the following survey data.

Obs	location	purchase_ method	rating
1	Atlanta	In Person	8.4
2	Atlanta	In Person	7.4
3	Seattle	In Person	6.4
4	Vermont	Web	9.0
5	Vermont		5.0
6	Vermont	In Person	9.8
7	Seattle	In Person	8.8
8	Seattle	Web	9.5
9	Atlanta	Web	8.5

Notice that the purchase method value is missing in observation 5. Here is a template that generates a vertical bar chart from this data that is grouped by the Purchase_Method column.

```
proc template;
  define statgraph survey;
    begingraph;
      entrytitle "Customer Survey Results";
      layout overlay / xaxisopts=(label="Store Location")
        yaxisopts=(label="Satisfaction Rating");
        barchart category=location response=rating / name="barchart"
          stat=mean group=purchase_method groupdisplay=cluster
          barwidth=0.9 dataskin=sheen includemissinggroup=true;
        discretelegend "barchart" / title="Purchase Method";
      endlayout;
    endgraph;
  end;
run;
```

To include a group for missing Purchase_Method column values, the INCLUDEMISSINGGROUP=TRUE option is specified in the BARCHART statement. The FORMAT procedure is used to create a format that assigns a meaningful label to the missing group.

```
/* Create a format for the missing order-type values */
proc format;
  value $ordertypefmt " "="Not Specified";
run;
```

The FORMAT procedure VALUE statement defines format $ORDERTYPEFMT, which assigns the label Not Specified to any missing character value. Notice that a missing character value is specified as a blank space enclosed in quotation marks in the VALUE statement. It might seem appropriate to use empty quotation marks (" or "") to specify a missing character value. However, doing so produces unexpected results. To specify a missing character value, enclose a single space in quotation marks (' ' or " ").

Format $ORDERTYPEFMT is then applied to the Purchase_Method column in the PROC SGRENDER statement.

```
/* Generate the chart */
proc sgrender data=surveydata template=survey;
  format purchase_method  $ordertypefmt.;
run;
ods graphics off;
```

Here is the output.

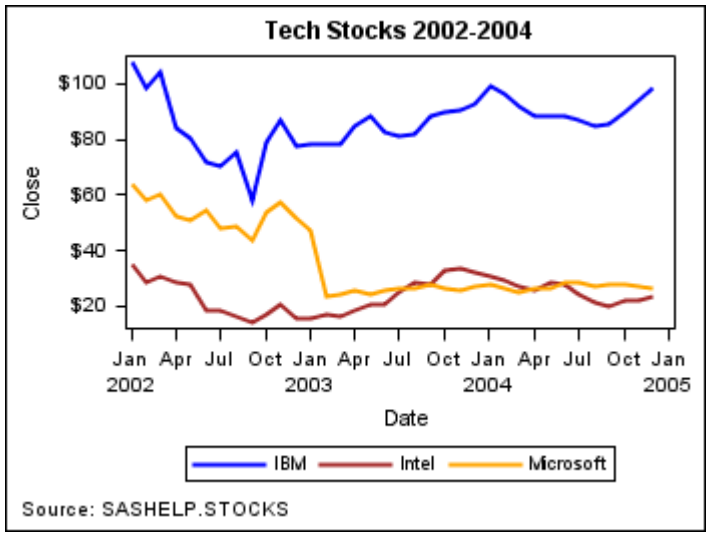
By default, the visual attributes of the missing group are determined by the GraphMissing style element. Because a user-defined format is applied to the group value in this example, the visual attributes of the missing group are determined by the GraphData3 style element of the default style. To change the visual attributes of the missing group value, you can use the INDEX= option in the BARCHART statement to specify a different GraphDataN style element, use a different ODS style, or you can use an attribute map.

Changing the Grouped Data Order

When unique group values are gathered, they are internally recorded in the order in which they appear in the data. They are not subsequently sorted. As a result, the group values appear in the plot in the order in which they occur in the data. Here is an example using the GROUPEDSERIES template was created in Example Code 27.2 on page 514 and the custom style STOCKS that was created in Example Code 27.3 on page 517.

```
/* Specify a path for the ODS output */
filename odsout "output-path";
ods _all_ close;
ods graphics / reset imagename="stocks";
ods html file="stocks.html" path=odsout style=stocks;

proc sgrender data= sashelp.stocks template=groupedseries;
  where date between "1jan02"d and "31dec04"d;
run;
ods html close;
ods html; /* Not required in SAS Studio */
```

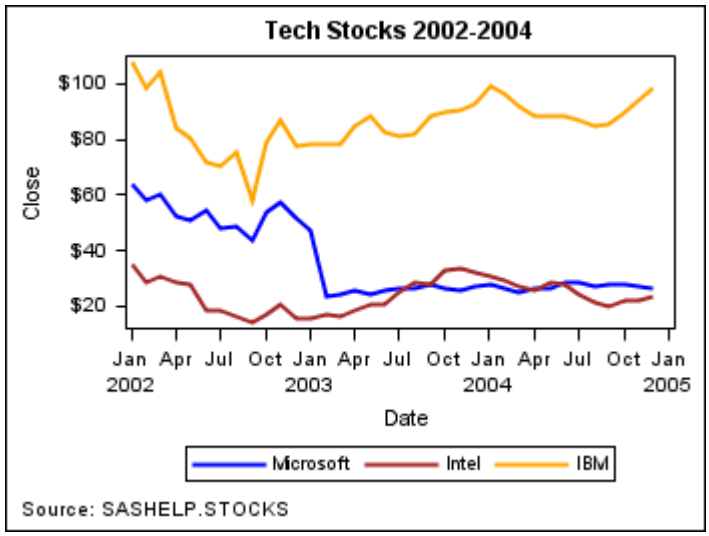
Here is the output.



In this example, the groups are ordered in the order in which they appear in the Sashelp.Stocks data set. Assume that you want to arrange the groups in ascending order. One way to do this is to use the SORT procedure to create a sorted data set to use with your plot. Here is an example.

```
/* Specify a path for the ODS output */
filename odsout "output-path";

proc sort data=sashelp.stocks out=stocks;
   by descending stock;
run;
ods _all_ close;
ods graphics / reset imagename="stocks";
ods html filename="stocks.html" path=odsout style=stocks;

proc sgrender data= stocks template=groupedseries;
 where date between
       "1jan02"d and "31dec04"d;
run;
ods html close;
ods html; /* Not required in SAS Studio */
```

Here is the result.



Changing the order of the data changes the order in which the group values appear on the plot. It also changes their association with the GraphData1–GraphDataN style elements, which might change their appearance. This is apparent in the previous example.

Another way to arrange your data is to use the GROUPORDER= option. For many plots, you can instead use the GROUPORDER= option to change the order of the groups in your plot without having to create a sorted data set or change the order of the data in the original data set.

Note: The GROUPORDER= option is ignored if the GROUP= option is not used.

The GROUPORDER= option determines the default order of groups in the legend and the order of the groups within each category. It does not change the association of the GraphData1–GraphDataN style attributes with the group values. You can set GROUPORDER= to one of the values that is shown in the following table.

Value	Description
DATA	Displays the group values in the order in which they appear in the plot data (default)
ASCENDING	Displays the group values in ascending order
DESCENDING	Displays the group values in descending order

The attributes of the missing group value are determined by the GraphMissing style element except when the MISSING= system option is used to assign a missing character other than "." or when a user-defined format is applied to the missing group value. In those cases, the attributes of the missing group value are determined by a GraphData1-GraphdDataN style element based on data order instead of the GraphMissing style element.

Note: Using the GROUPORDER=ASCENDING or GROUPORDER=DESCENDING option performs a linguistic sort on the group items and has the same effect as sorting the input data. However, the data is not changed.

Making the Appearance of Grouped Data Independent of Data Order

About Grouped Data Appearance and Data Order

When a SAS data set is plotted, group values are assigned a GraphDataN style element in data order. When a CAS in-memory table is plotted, the group values are assigned a GraphDataN style element in ascending order of the group column character or of the unformatted numeric values. When the input data source is modified, sorted, or filtered, the association of each group value with a GraphData1–GraphDataN style element might change. If you do not care which line pattern, marker symbols, or colors are associated with particular group values, changing associations might not be a problem. However, there might be cases in which you want the appearance of your plots to be consistent. For example, if you create several stock market plots grouped by STOCK, you might want a consistent set of visual properties for each stock name across all of the plots, regardless of the input data order.

There are two methods that you can use to achieve visual consistency regardless of the data order: you can use a discrete attribute map to assign attributes to specific group values or you can use the INDEX= option to map group values to specific graphData1–graphDataN style elements. This section shows you how to use both methods.

Using a Discrete Attribute Map to Achieve Data-Independent Appearance for Grouped Plots

You can use a discrete attribute map to associate visual attributes with specific group values, which enables you to make plots that are consistent regardless of the data order. Here is an example of a stock plot that is visually consistent regardless of the order of the data. This example plots the closing stock price for IBM, Microsoft, and Intel. In this example, each plot always uses the attributes shown in the following table.

Stock	Line Pattern	Marker	Line and Marker Color
IBM	solid	filled circle	blue
Intel	solid	filled square	orange
Microsoft	solid	filled triangle	dark red

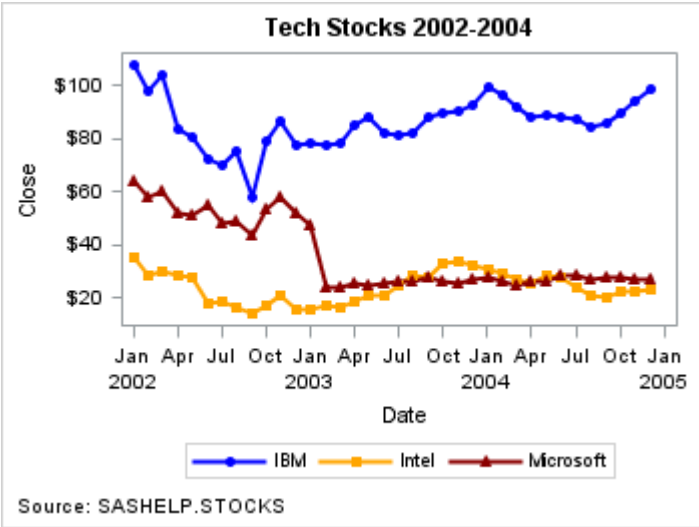
To enforce this consistency, you must create a discrete attribute map that maps the desired attributes to the stock values. Here is the code for this example.

```
/* Define the graph template. */
proc template;
  define statgraph groupindex;
    begingraph;
      /* Create an attribute map for this graph. */
      discreteattrmap name="stockname" / ignorecase=true;
        value "IBM" /
          markerattrs=GraphData1(color=blue  symbol=circlefilled)
          lineattrs=GraphData1(color=blue pattern=solid);
        value "Intel" /
          markerattrs=GraphData2(color=orange symbol=squarefilled)
          lineattrs=GraphData2(color=orange pattern=solid);
        value "Microsoft" /
          markerattrs=GraphData3(color=darkred symbol=trianglefilled)
          lineattrs=GraphData3(color=darkred pattern=solid);
      enddiscreteattrmap;

      /* Create the attribute map variable. */
      discreteattrvar attrvar=stockmarkers var=stock
        attrmap="stockname";

      /* Define the graph. */
      entrytitle "Tech Stocks 2002-2004";
      entryfootnote halign=left "Source: SASHELP.STOCKS";
      layout overlay;
        seriesplot x=date y=close / group=stockmarkers
          name="series" lineattrs=(thickness=2) display=(markers);
        discretelegend "series";
      endlayout;
    endgraph;
  end;
run;

/* Render the graph. */
proc sgrender data= sashelp.stocks template=groupindex;
  where date between "1jan02"d and "31dec04"D;
run;
```

Note: Custom style STOCKS was created in Example Code 27.3 on page 517.

Here is the output.

Figure 27.4 Graph Using a Discrete Attribute Map



To verify that the plot attributes are consistent regardless of the data order, create a temporary data set from the Sashelp.Stocks data set and sort it by date and stock name as shown in the following code.

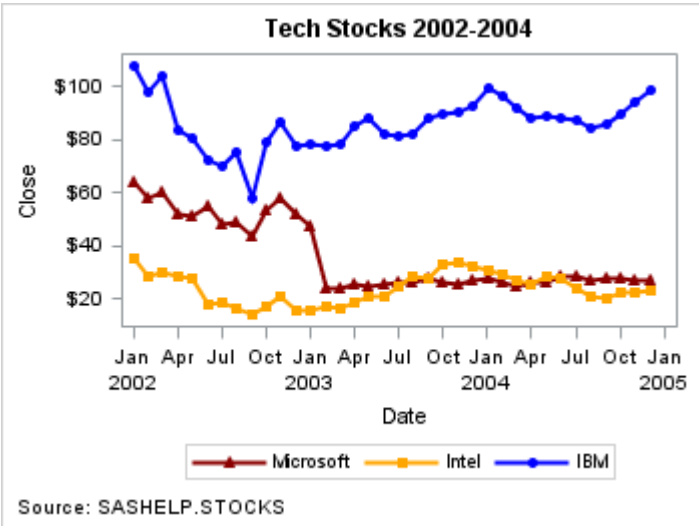
```
/* Sort the SASHELP.STOCKS data by date and stock name. */
proc sort data=sashelp.stocks out=work.stocks;
   by date descending stock;
run;
```

Next, generate the graph again using the sorted data set.

```
/* Render the graph. */
proc sgrender data= stocks template=groupindex;
 where date between "1jan02"d and "31dec04"D;
run;
```

Here is the result.

Figure 27.5 Graph with the Data in Descending Order



Notice that the colors, line patterns, and markers remain the same for each stock even though the data order has changed. For more information about discrete attribute maps, see "Using a Discrete Attribute Map" on page 538.

Using the INDEX= Option to Achieve Data-Independent Appearance for Grouped Plots

You can use the INDEX= option to specify a column in your data set that maps graphData1–graphDataN style elements to specific group values. This enables you to make plots that are visually consistent regardless of the data order. Here is the example in "Using a Discrete Attribute Map to Achieve Data-Independent Appearance for Grouped Plots" on page 532 modified to use the INDEX= option instead of a discrete attribute map to achieve the same result.

To enforce this consistency, you must do the following:

- Add numeric column IDX to the data set that maps the stock values to a graphDataN style element as follows:

  1 IBM

  2 Intel

  3 Microsoft

- Specify the INDEX=IDX option in the SERIESPLOT statement. This maps graphData1 to IBM, graphData2 to Intel, and graphData3 to Microsoft regardless of the data order.

Here is the code for this example.

```
/* Specify a path for the ODS output */
filename odsout "output-path";

/* Add the IDX column to the data set */
data stocks;
  set sashelp.stocks;
  if (stock="IBM") then idx=1;
  else if (stock="Microsoft") then idx=2;
    else if (stock="Intel") then idx=3;
run;

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

```
  endgraph;
end;
run;

/* Render the graph using the custom style. */
ods _all_ close;
ods graphics / reset imagename="stocks";
ods html file="stocks.html" path=odsout style=stocks;
proc sgrender data= stocks template=groupindex;
  where date between "1jan02"d and "31dec04"D;
run;
ods html close;
ods html; /* Not required in SAS Studio */
```

Note: Custom style STOCKS was created in Example Code 27.3 on page 517.

The output is the same as that shown in Figure 27.4 on page 534.

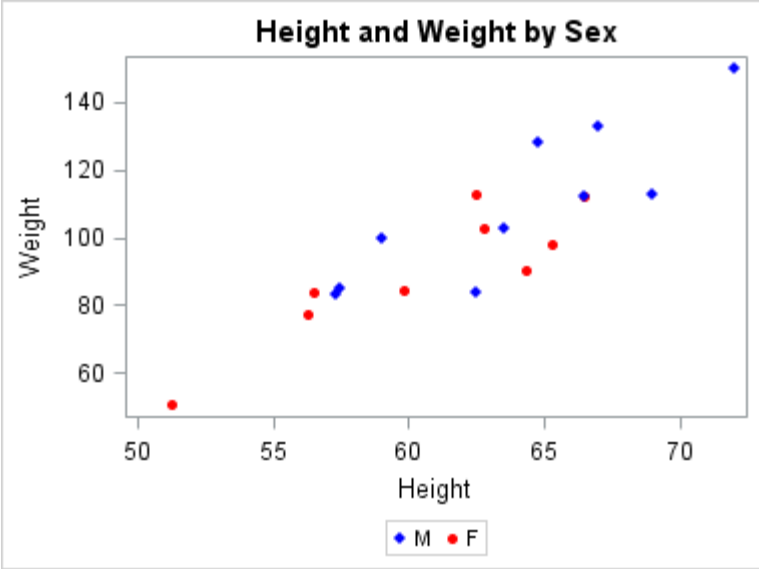
As described in "Using a Discrete Attribute Map to Achieve Data-Independent Appearance for Grouped Plots" on page 532, if you sort the data and generate the graph again, the visual attributes of the graph remain the same as that shown in Figure 27.5 on page 534.

Using Attribute Maps

About Attribute Maps

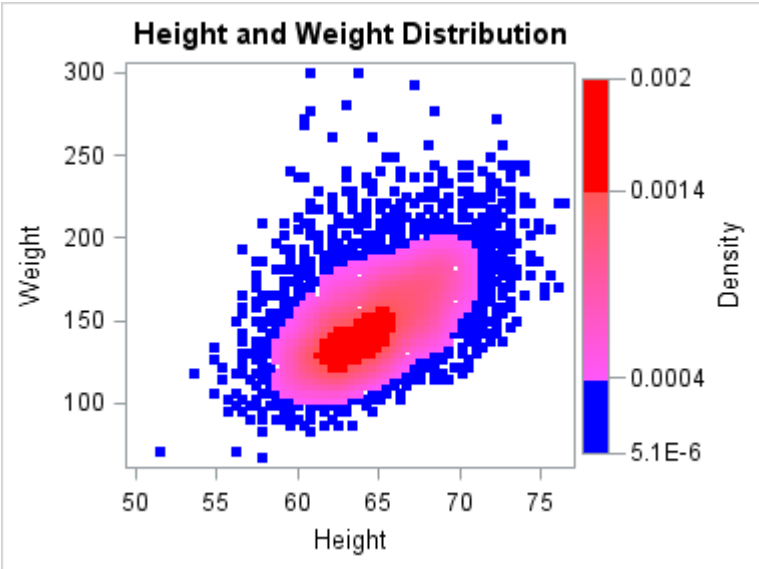
By default, many of the graphical attributes of a plot vary with the plot data. For example, when plots display grouped values, the graphical attributes for each group value are selected by default from the GraphData1–GraphDataN style elements in data order. Changes in the data order can significantly change the appearance of the plot. When plots display a color gradient, the colors assigned to the classification variable are derived by default from a color ramp based on the actual range of the data. The color assigned to each value can vary with the range of the classification values. Attribute maps enable you to assign the same graphical properties to specific values or ranges of values regardless of data order or the data range. They are useful when you want your graphs to have consistent visual properties when the data varies.

GTL supports two types of attribute maps: discrete attribute maps and range attribute maps. A discrete attribute map maps discrete values to graphical properties. For example, consider the following plot of height and weight grouped by sex. You can use a discrete attribute map to assign a filled blue diamond to M and a red filled circle to F, as shown in the following figure.



Regardless of data order, the same plot markers and colors are applied to the group values.

A range attribute map maps numeric values or ranges of numeric values to graphical properties. For example, consider the following plot of height and weight distribution. You can use a range attribute map to assign colors based on specific ranges of density, as shown in the following figure.



Regardless of the actual data ranges, the same colors are applied to the data that falls within the specified ranges.

Using a Discrete Attribute Map

How a Discrete Attribute Map Is Defined and Used

To define and use a discrete attribute map, you must do the following:

- Define the attribute map in a DISCRETEATTRMAP block in your template or in a SAS data set.
- Associate the attribute map with a classification variable in the plot data.
- Reference the attribute map where needed in your plot statements.

For more information about how to create a discrete attribute map and how to reference it in your plot statements, see "Key Concepts for Using Attribute Maps" in *SAS Graph Template Language: Reference*.

Defining Your Discrete Attribute Map in Your Template

To define and use your attribute map in your template, use a DISCRETEATTRMAP block to specify the mapping information. Use a DISCRETEATTRVAR statement to create an attribute-map variable that you can use to reference the attribute map in your plot statements. Place the DISCRETEATTRMAP block and the DISCRETEATTRVAR statement in the global definition area of the BEGINGRAPH block. Do not embed the DISCRETEATTRMAP block or the DISCRETEATTRVAR statement in any other GTL block. They must be children of the BEGINGRAPH statement.

Here is an example that creates and applies a discrete attribute map to the values in column Stock of the plot data set.

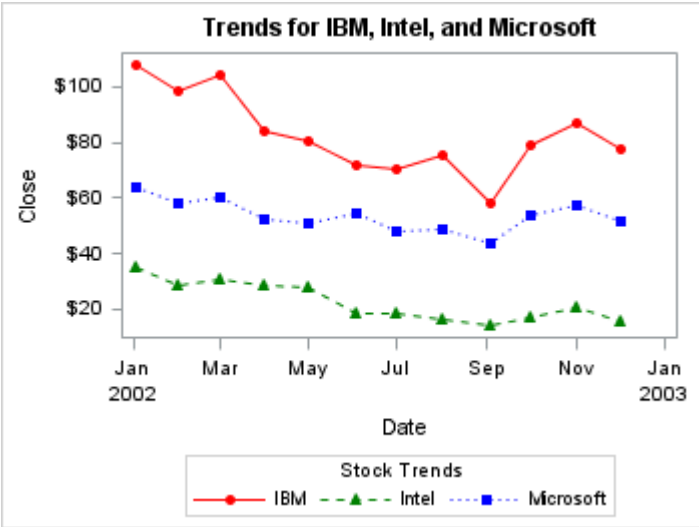
```
/* Create a stock data set for the year 2002 */
proc sort data=sashelp.stocks out=stocks;
  by stock date;
  where date between '01JAN02'd and '30DEC02'd;
run;

/* Create a template for IBM, Microsoft, and Intel stocks */
proc template;
define statgraph stockchart;
begingraph;
  entrytitle "Trends for IBM, Intel, and Microsoft";
  discreteattrmap name="stockname" / ignorecase=true;
    value "IBM" /
      markerattrs=GraphData1(color=red  symbol=circlefilled)
      lineattrs=GraphData1(color=red pattern=solid);
```

```
      value "Intel" /
        markerattrs=GraphData2(color=green symbol=trianglefilled)
        lineattrs=GraphData2(color=green pattern=shortdash);
      value "Microsoft" /
        markerattrs=GraphData3(color=blue symbol=squarefilled)
        lineattrs=GraphData3(color=blue pattern=dot);
    enddiscreteattrmap;
    discreteattrvar attrvar=stockmarkers var=stock
      attrmap="stockname";
    layout overlay;
      seriesplot x=date y=close /
        group=stockmarkers
        display=(markers)
        name="trends";
      discretelegend "trends" / title="Stock Trends";
    endlayout;

  endgraph;
  end;
  run;

  /* Plot the stock trends  */
  proc sgrender data=stocks template=stockchart;
  run;
  quit;
```

This example overrides the attributes in the default style with different line and marker attributes for the IBM, Intel, and Microsoft stock plot lines. In the example code, notice that the NAME="stockname" option in the DISCRETEATTRMAP statement provides a name for the discrete attribute map. Also notice that the ATTRVAR=stockmarkers option in the DISCRETEATTRVAR statement provides a name for the attribute-map-to-data-set-column association. The ATTRMAP="stockname" and the VAR=stock options in the DISCRETEATTRVAR statement associate the attribute map stockname with the data set column Stock respectively to create an attribute variable. In the SERIESPLOT statement, the GROUP=stockmarkers option applies the attribute map to the specified group values.

The following figure shows the resulting output.

Figure 27.6 *An Attribute Map Used in a Plot of Stock Trends*



In this example, the markers, line colors, and line patterns in the plot remain the same regardless of the data, data order, or ODS style that is in effect. For example, if you run this example with a different ODS style and without any data for IBM, the appearance of the Intel and Microsoft plot lines and markers remain the same. The legend displays the items that appear in the data using the attributes that are defined in the discrete attribute map.

TIP To display all of the items that are defined in the discrete attribute map in the legend regardless of whether they appear in the data, specify the DISCRETELEGENDENTRYPOLICY=ATTRMAP option in the DISCRETEATTRMAP statement. See "DISCRETEATTRMAP" in *SAS Graph Template Language: Reference* for more information.

Defining Your Discrete Attribute Map in a SAS Data Set

Starting with SAS 9.4M1, you can define your discrete attribute map in a SAS data set rather than in your template. An attribute map cannot be defined in a CAS in-memory table. You must define it in a SAS data set. Using a SAS data set enables you to change the attribute mapping without having to modify your template. The data set replaces the DISCRETEATTRMAP block and VALUE statements in the template code. The SAS data set contains one observation for each discrete value that you want to map.

Each observation contains a required ID and VALUE column, and one or more graphical property columns. The ID column specifies the name of the attribute map. The Value column specifies the discrete values that are to be mapped to the specified graphical properties. The graphical property columns define the graphical properties that are mapped to each value. The data set can contain observations for more than one attribute map.

For more information about how to define a discrete attribute map in a SAS data set, see "Specifying the Attribute Mapping Information in a SAS Data Set" in *SAS Graph Template Language: Reference*.

To associate the attribute map that is defined in the data set with a classification variable in the plot data, use one of the following methods in the template:

- Use a DISCRETEATTRVAR statement to create an attribute map variable for the attribute map, and then specify the attribute map variable as needed in your plot statements.
- Specify the classification variable by name in your plot statements as needed. In the SGRENDER statement, include a DATTRVAR statement that associates the classification variable in the plot data with the attribute map.

With both methods, you must also include the DATTRMAP= option in the SGRENDER statement to specify the name of the attribute map data set.

Note: For information about the SGRENDER DATTRMAP= option and the DATTRVAR statement, see *SAS ODS Graphics: Procedures Guide*

In the example shown in "Defining Your Discrete Attribute Map in Your Template" on page 538, you can create the following data set to replace the DISCRETEATTRMAP block in the template code:

```
/* Create the attribute map data set */
data attrmap;
  length ID VALUE MARKERCOLOR MARKERSYMBOL LINECOLOR LINEPATTERN $15;
  input ID$ VALUE$ MARKERCOLOR$ MARKERSYMBOL$ LINECOLOR$ LINEPATTERN$;
datalines;
stockname IBM       red   circlefilled   red   solid
stockname Intel     green trianglefilled green shortdash
stockname Microsoft blue  squarefilled   blue  dot
;
run;
```

The ID column specifies STOCKNAME as the name of the attribute map. The Value column specifies the mapped values, and the remaining columns specify the graphical properties that are mapped to each value. Here is the example code, modified to use the Attrmap data set rather than the DISCRETEATTRMAP block .

```
/* Create a stock data set for the year 2002 */
proc sort data=sashelp.stocks out=stocks;
  by stock date;
  where date between '01JAN02'd and '30DEC02'd;
run;

/* Create a template for IBM, Microsoft, and Intel stocks */
proc template;
define statgraph stockchart;
begingraph;
  entrytitle "Trends for IBM, Intel, and Microsoft";
  discreteattrvar attrvar=stockmarkers var=stock
    attrmap="stockname";
  layout overlay;
    seriesplot x=date y=close /
      group=stockmarkers
      display=(markers)
      name="trends";
```

```
    discretelegend "trends" / title="Stock Trends";
  endlayout;
endgraph;
end;
run;

/* Plot the stock trends */
proc sgrender data=stocks dattrmap=attrmap template=stockchart;
run;
quit;
```

The DISCRETEATTRVAR statement creates the attribute map variable STOCKMARKERS, which is specified by the GROUP= option in the SCATTERPLOT statement. The ATTRMAP= option in the DISCRETEATTRVAR statement specifies the name of the attribute map, as specified in the ID column of the attribute map data set. The DATTRMAP= option in the SGRENDER statement specifies the name of the attribute map data set. The output is shown in Figure 27.6 on page 540. To associate the Stock variable with a different attribute map, you must modify the DISCRETEATTRVAR statement ATTRMAP= option in the template code to specify the new attribute map name. Likewise, to associate the STOCKNAME attribute map with a different variable, you must modify the DISCRETEATTRVAR statement VAR= option to specify the new variable name.

Here is the template, modified to use a DATTRVAR statement in the SGRENDER statement instead of using the DISCRETEATTRVAR statement in the template. The modification associates the attribute map with column Stock in the plot data.

```
/* Create a template for IBM, Microsoft, and Intel stocks */
proc template;
define statgraph stockchart;
begingraph;
  entrytitle "Trends for IBM, Intel, and Microsoft";
  layout overlay;
    seriesplot x=date y=close /
      group=stock
      display=(markers)
      name="trends";
    discretelegend "trends" / title="Stock Trends";
  endlayout;
endgraph;
end;
run;

/* Plot the stock trends  */
proc sgrender data=stocks dattrmap=attrmap template=stockchart;
   dattrvar stock="stockname";
run;
quit;
```

In this case, the Stock column is specified by name in the GROUP= option in the SERIESPLOT statement. In the SGRENDER statement, the DATTRMAP= option specifies the attribute map data set name, and the DATTRVAR statement associates the column Stock with the attribute map that is specified in the ID column of the attribute map data set. To associate the Stock variable with a different attribute map or to associate the STOCKNAME attribute map with a different variable, you need only modify the DATTRVAR statement. You do not need to modify the template code.

In some cases, you might need to use a WHERE statement in your SGRENDER procedure step to subset your data. Be aware that a WHERE statement in the SGRENDER procedure step applies to the data set specified in the DATA= option only. It does not apply to the attribute-map data set specified in the DATTRMAP= option. If you need to subset the data in your attribute map data set using a WHERE clause, use a WHERE statement in a DATA step to create a new attribute map data set or use the data set WHERE option in the SGRENDER procedure DATTRMAP= option. Here is an example of the latter:

```
dattrmap=attrmap(where=(value in ("Microsoft", "Intel")))
```

Using a Range Attribute Map

To define and use a range attribute map, you must do the following:

- Define the attribute map in a RANGEATTRMAP block in the global definition area of your template between the BEGINGRAPH statement and the first layout statement. Do not embed the RANGEATTRMAP block in any other GTL block. It must be a child of the BEGINGRAPH statement.
- Use a RANGEATTRVAR statement to associate the attribute map with a response variable in the plot data. Place the RANGEATTRVAR statement in the global definition area of your template. Do not embed the RANGEATTRVAR statement in any other GTL block. It must be a child of the BEGINGRAPH statement.
- Reference the attribute map where needed in your plot statements.

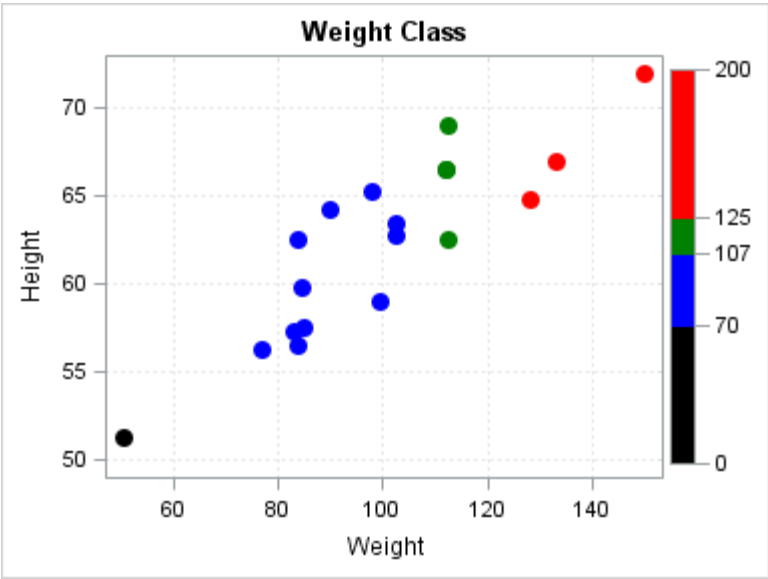
For more information about how to create a range attribute map, see "Defining a Range Attribute Map" in *SAS Graph Template Language: Reference*.

Here is an example of a template that creates and applies a range attribute map to the Weight column of a SCATTERPLOT statement data set in order to color the markers in the resulting plot by weight range. It also creates a continuous legend.

```
proc template;
  define statgraph attrrange;
    begingraph;
      /* Create the range attribute map. */
      rangeattrmap name="scale";
        range 0-70 /
          rangealtcolor=black; /* 0 to 70 inclusive */
        range 70<-107 /
          rangealtcolor=blue; /* 70 exclusive to 107 inclusive */
        range 107<-125 /
          rangealtcolor=green; /* 107 exclusive to 125 inclusive */
        range 125<-200 /
          rangealtcolor=red; /* 125 exclusive to 200 inclusive */
      endrangeattrmap;

      /* Create the range attribute map variable. */
      rangeattrvar attrvar=weightrange var=weight attrmap="scale";

      /* Create the graph. */
      entrytitle "Weight Class";
      layout overlay /
```

```
        xaxisopts=(griddisplay=on gridattrs=(color=lightgray
pattern=dot))
        yaxisopts=(griddisplay=on gridattrs=(color=lightgray
pattern=dot));
        scatterplot x=weight y=height / markercolorgradient=weightrange
          markerattrs=(symbol=circlefilled size=10) name='wgtclass';

        /* Add a continuous legend. */
        continuouslegend 'wgtclass';
      endlayout;
    endgraph;
  end;
run;

/* Render the graph. */
ods graphics / width=4in height=3in;
proc sgrender data=sashelp.class template=attrrange;
run;
```

The following figure shows the resulting output.



In the example code, notice that the NAME="scale" option in the RANGEATTRMAP statement provides a name for the range attribute map. A RANGE statement specifies a color for each of four ranges. Because the SCATTERPLOT statement uses the ContrastColor style attribute for the marker colors in a grouped plot, the RANGEALTCOLOR= option is used in each RANGE statement to define the range color. The ATTRVAR=weightrange option in the RANGEATTRVAR statement provides a name for the attribute-map-to-data-set-column association. The ATTRMAP="scale" and the VAR=weight options in the RANGEATTRVAR statement associate the attribute map scale with the data set column Weight respectively. In the SCATTERPLOT statement, the MARKERCOLORGRADIENT=weightrange option applies the range attribute map to the Weight column values and colors the plot markers according to the ranges that are specified in the range attribute map. To add a legend that displays the marker colors for the weight ranges, you can include a CONTINUOUSLEGEND statement. For information about continuous legends, see "Adding a Continuous Legend" on page 374.

Attribute Rotation Patterns

About the Attribute Rotation Patterns

By default, attributes such as colors, marker symbols, fill patterns, and line patterns are rotated for group values in a grouped plot or for each plot in an overlay when CYCLEATTRS=TRUE. The attributes are derived from the GraphData1–GraphDataN style elements for the style that is in effect. There are two distinct attribute rotation patterns: the default pattern and the color-priority pattern. These patterns are described in "The Default Attribute Rotation Pattern" on page 546 and "The Color-Priority Attribute Rotation Pattern" on page 549.

The rotation pattern that is used is determined by the AttrPriority attribute in the current style, by the ATTRPRIORITY= option in the ODS GRAPHICS statement, or by the ATTRPRIORITY= option in the BEGINGRAPH statement. The ODS GRAPHICS statement ATTRPRIORITY= option value is AUTO by default, which enables the AttrPriority attribute in the current style to determine the rotation pattern. In that case, if the current style's AttrPriority attribute specifies COLOR, the color-priority rotation pattern is used. If the current style does not specify the AttrPriority attribute or specifies AttrPriority="NONE", the default rotation pattern is used. The ODS GRAPHICS statement ATTRPRIORITY= option can override the current AttrPriority style attribute setting with NONE or COLOR. (See "ODS GRAPHICS" in *SAS Graph Template Language: Reference*.) The BEGINGRAPH statement ATTRPRIORITY= option overrides the ODS GRAPHICS statement ATTRPRIORITY= option for all plots in a BEGINGRAPH block. (See ATTRPRIORITY=.)

> **TIP** If you want groups to be distinguished by color, marker, and line changes for all ODS styles that use color, specify ATTRPRIORITY=NONE in your BEGINGRAPH statement or in an ODS GRAPHICS statement.

When you plot a SAS data set with either the default rotation pattern or the color-priority pattern, the group-value attributes are assigned in data order. Any change in the data order can affect the appearance of grouped plots. When you plot a CAS in-memory table, the group-value attributes are always assigned in ascending order of the group column character or of the unformatted numeric values.

> **TIP** To make the appearance of your grouped plots independent of data order, use a discrete attribute map or the INDEX= option (where supported) to control the appearance of your group values. See "Making the Appearance of Grouped Data Independent of Data Order" on page 532.

The Default Attribute Rotation Pattern

The default rotation iterates through the lists of colors, marker symbols, fill patterns (when supported), and line patterns as they are defined in the GraphData1–GraphDataN style elements.

Note: Prior to SAS 9.4M5, fill patterns are supported only by selected ODS styles such as JOURNAL2 and MONOCHROMEPRINTER. Starting with SAS 9.4M5, the DEFAULT ODS style and all of the styles that are derived from it support fill patterns. For more information, see *SAS Output Delivery System: User's Guide*.

When all of the values in the list for contrast colors, marker symbols, fill patterns, and line patterns have been used, the values iterate through the list again. When all of the values for the area fill colors have been used, two new sets of colors are automatically generated, which are based on the original colors. The first set is one shade lighter than the original colors, and the second set is one shade darker.

The number of unique attribute combinations that are available depends on the ODS style that you are using. The DEFAULT style, for example, defines the attributes that are shown in the following table.

Table 27.3 *Colors, Marker Symbols, and Line Patterns Defined in the DEFAULT Style*

Style Element	Color[1]	Contrast Color[1]	Marker Symbol[2]	Line Pattern[3]	Fill Pattern [4]
GraphData1	cx7C95CA	cx2A25D9	CIRCLE	1	L1
GraphData2	cxDE7E6F	cxB2182B	PLUS	4	X1
GraphData3	cx66A5A0	cx01665E	X	8	R1
GraphData4	cxA9865B	cx543005	TRIANGLE	5	L2
GraphData5	cxB689CD	cx9D3CDB	SQUARE	14	X2
GraphData6	cxBABC5C	cx7F8E1F	ASTERISK	26	R2
GraphData7	cx94BDE1	cx2597FA	DIAMOND	15	L3
GraphData8	cxCD7BA1	cxB26084		20	X3
GraphData9	cxCF974B	cxD17800		41	R3
GraphData10	cx87C873	cx47A82A		42	L4
GraphData11	cxB7AEF1	cxB38EF3		2	R4

Style Element	Color[1]	Contrast Color[1]	Marker Symbol[2]	Line Pattern[3]	Fill Pattern [4]
GraphData12	cxDDD17E	cxF9DA04			

1 The colors are defined by GDATA1 through GDATA12, and the contrast colors are defined by GCDATA1 through GCDATA12 in the style template.

2 See the SYMBOL= option in "Marker Options" on page 668.

3 See "Available Line Patterns" on page 670.

4 Fill patterns are available with the DEFAULT ODS style starting with SAS 9.4M5. For an example of each of the fill patterns, see Table A3.1 on page 666.

These attribute definitions yield the following results:

- 36 unique area fill colors (12 base colors + 12 lighter colors + 12 darker colors)
- 7 unique marker symbols, and 84 unique marker symbol and marker color combinations (7 symbols x 12 contrast colors or fill colors)
- 11 unique line patterns, and 132 unique line color and line pattern combinations (11 patterns x 12 contrast colors)
- 11 unique area fill patterns, and 396 unique area fill pattern and fill color combinations (11 fill patterns x 36 fill colors)

Note: Fill patterns are available with the DEFAULT ODS style starting with SAS 9.4M5.

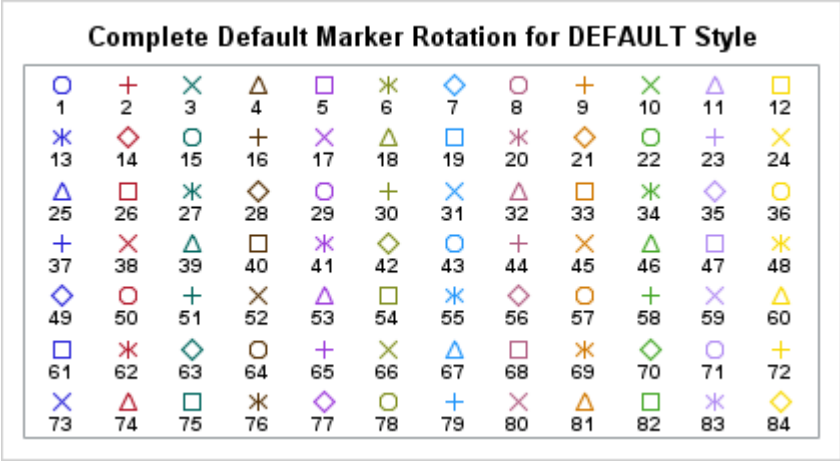
When area fill colors are rotated, the colors first iterate through the 36 unique colors (the 12 base colors, the 12 lighter colors, and finally the 12 darker colors). The entire pattern repeats starting with the 37th iteration. The following figure shows one complete fill color rotation cycle for the DEFAULT style.

When marker symbol attributes are rotated, the marker symbols first iterate through the seven symbols and the first seven contrast colors (unfilled markers). On the eighth iteration, the first marker symbol (CIRCLE in this case) is repeated with the eighth contrast color. The next four symbols are then repeated with the remaining three contrast colors. On the 13th iteration, the first contrast color is repeated with the fifth marker symbol (SQUARE in this case). This pattern continues for the remaining group values. The entire pattern repeats, starting with the 85th iteration.

Note: For filled markers and outlined filled markers, the marker fill colors rotate through the fill colors that are defined in the GraphData1–GraphDataN style

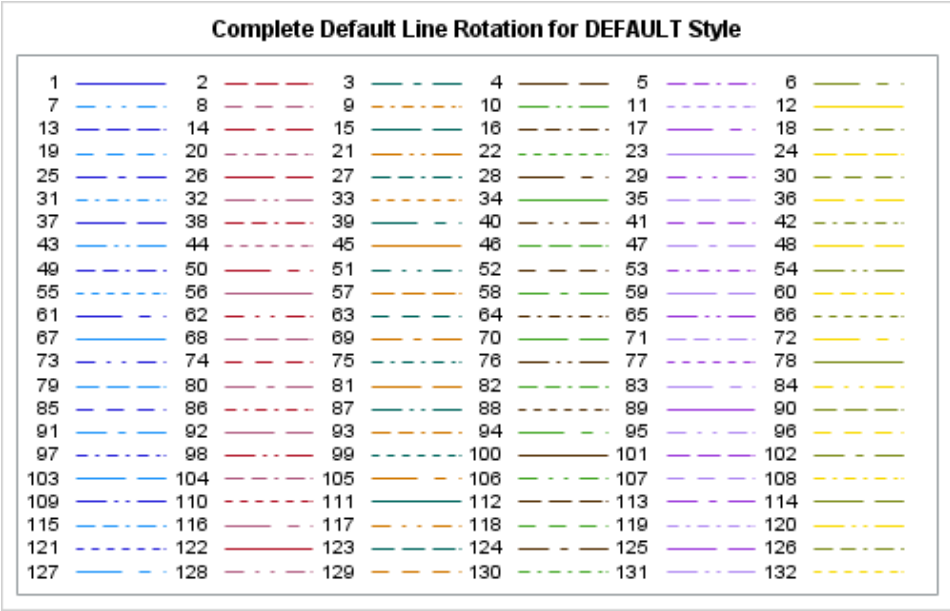
elements. Unlike area fill colors, new lighter and darker colors are not generated for the marker fills. The marker fill colors simply repeat.

The following figure shows the complete standard rotation of marker symbols and symbol colors for the DEFAULT style.



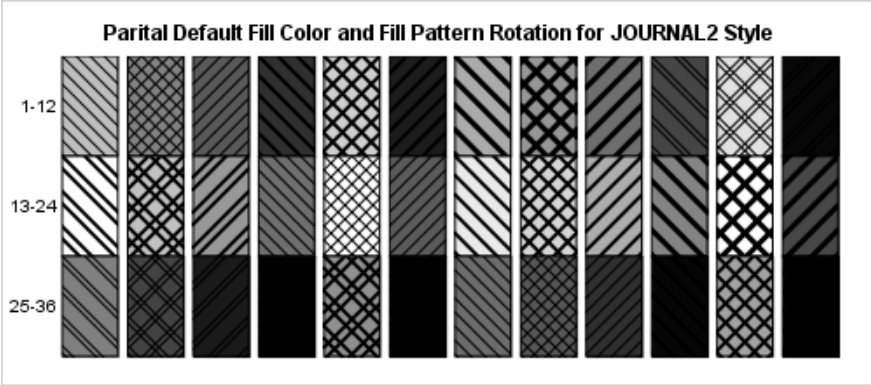
When line attributes are rotated, the line patterns first iterate through the 11 line patterns with the first 11 contrast colors. On the 12th iteration, the first line pattern (1) is repeated with the 12th contrast color (see "Available Line Patterns" on page 670). On the 13th iteration, the first contrast color is repeated with the second line pattern (4). This pattern continues for the remaining group values. The entire pattern repeats, starting with the 133rd iteration.

The following figure shows the complete standard rotation of line styles and line colors for the DEFAULT style.

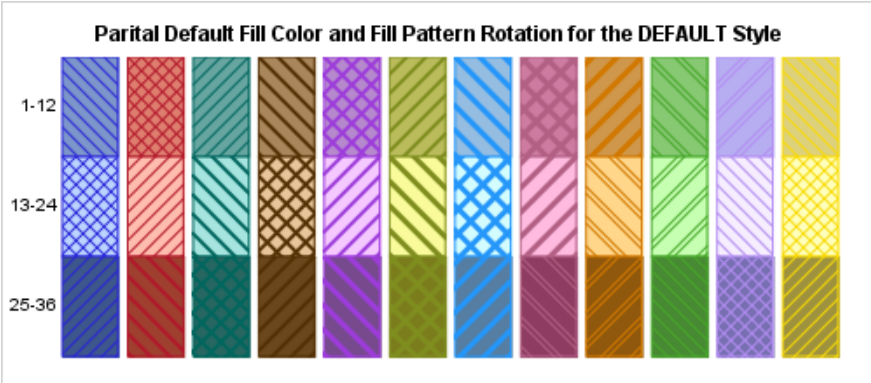


Prior to SAS 9.4M5, some ODS styles such as JOURNAL2 provide fill pattern as an additional attribute. Starting with SAS 9.4M5, the DEFAULT ODS style and all of the styles that are derived from it provide fill pattern as an additional attribute. By default, the fill patterns rotate as they are defined in the GraphData1–GraphDataN style elements. The JOURNAL2 style, for example, provides 36 gray-scale fill colors

and 15 unique fill patterns, which yield 540 unique fill color and fill pattern combinations. The following figure shows the default fill color and fill pattern rotation for the first 36 iterations using the JOURNAL2 style.



Starting with SAS 9.4M5, the DEFAULT ODS style provides 36 fill colors and 11 unique fill patterns, which yield 396 unique fill color and fill pattern combinations. The following figure shows the default fill color and fill pattern rotation for the first 36 iterations using the DEFAULT style.



ODS styles that are derived from the DEFAULT style follow the same default fill-pattern rotation.

The Color-Priority Attribute Rotation Pattern

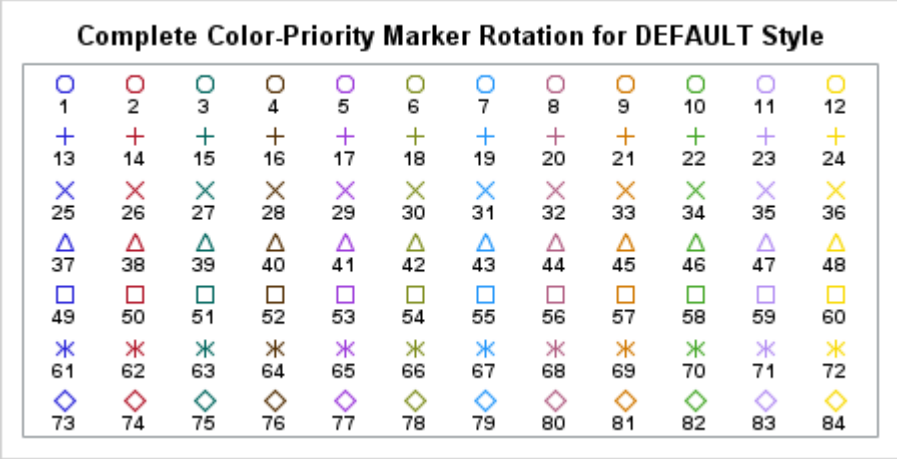
The color-priority rotation pattern alters the rotation of marker symbol and line attributes, and fill patterns (when supported) only. Area fill-color rotation follows the default rotation in the color-priority pattern. In the color-priority rotation pattern, the marker symbol, line pattern, or fill pattern is held constant while each color in the list is applied to the marker symbol, line, or area. The number of unique combinations is unchanged from the default rotation pattern. The examples in this section use the DEFAULT style to describe the color-priority rotation pattern so that you can compare the two patterns. See Table 27.3 on page 546.

Note: Because the DEFAULT style does not define the AttrPriority="COLOR" style attribute, in order to activate the color-priority rotation pattern, you must specify the ATTRPRIORITY=COLOR option in an ODS GRAPHICS statement in your SAS

program or in the BEGINGRAPH statement in your template. See ATTRPRIORITY=.

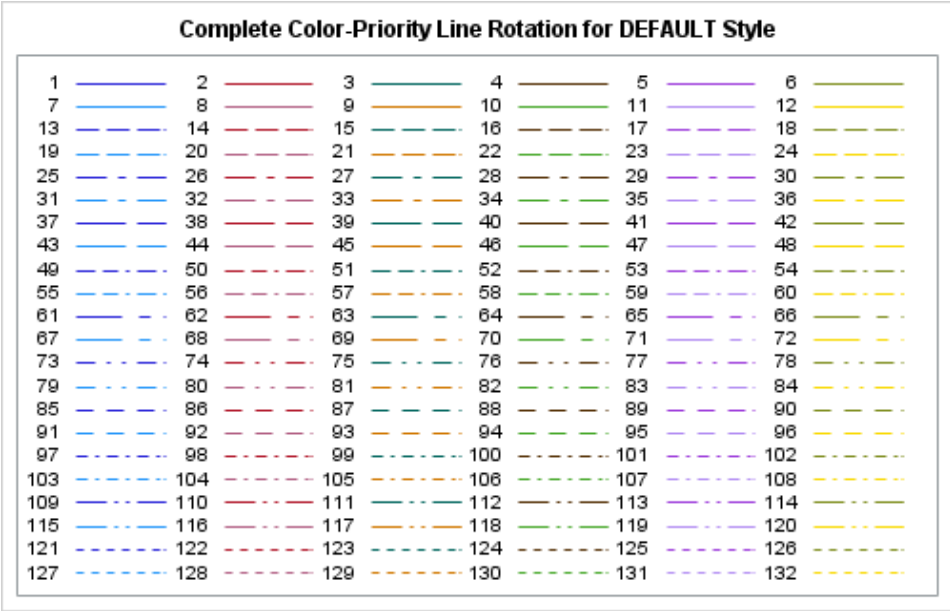
When marker attributes are rotated, the first symbol (CIRCLE in this case) is held constant while the colors iterate through the entire list of contrast colors (unfilled markers). On the 13th iteration, the marker symbol changes to the second symbol in the list (PLUS in this case), which is held constant while the colors iterate again. This pattern continues for the remaining group values. The entire pattern repeats, starting with the 85th iteration.

The following figure shows the complete color-priority rotation of marker symbols and symbol colors for the DEFAULT style.

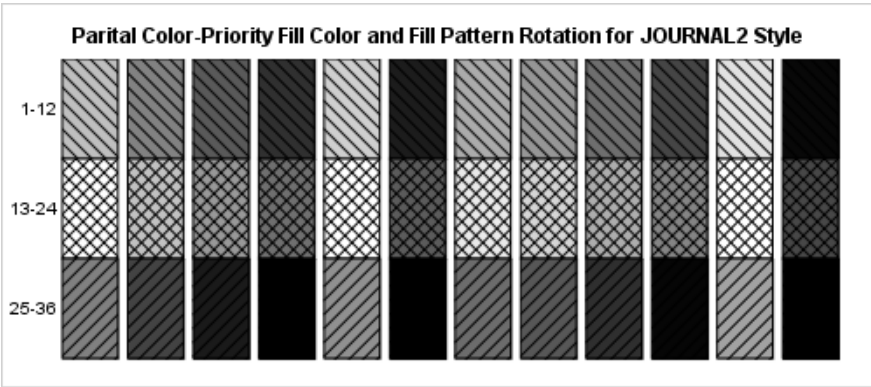


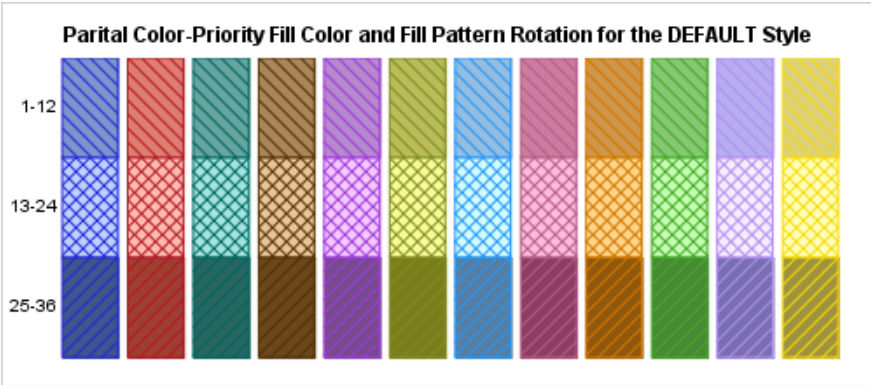
When line attributes are rotated, the first line pattern (1) is held constant while the colors iterate through the entire list of contrast colors (see "Available Line Patterns" on page 670). On the 13th iteration, the line pattern changes to the second pattern (4), which is held constant while the colors iterate again. This pattern continues for the remaining group values. The entire pattern repeats, starting with the 133rd iteration.

The following figure shows the complete color-priority rotation of line styles and line colors for the DEFAULT style.



When fill patterns are rotated, the first pattern is held constant while the fill colors rotate through the entire list of fill colors. When all of the fill colors have been used, the next fill pattern is held constant while the fill colors iterate again. This pattern continues for the remaining fill patterns. Prior to SAS 9.4M5, only ODS styles JOURNAL2, JOURNAL3, and MONOCHROMEPRINTER support fill patterns. Starting with SAS 9.4M5, the DEFAULT style and all of the styles that are derived from it also support fill patterns. The following figures show the color-priority fill color and fill pattern rotation for the first 36 iterations using the JOURNAL2 and DEFAULT styles.





ODS styles that are derived from the DEFAULT style follow the same color-priority fill-pattern rotation.

Using Transparency

GTL supports transparency for plot features such as filled areas, marker symbols, lines, and so on, for output formats that support transparency. Output formats that support transparency are PNG and vector graphics, with the following exceptions:

- The EMF output format does not support transparency when the graph is rendered as vector-graphics output. Transparency is supported when the graph is rendered as a PNG image.
- The ODS PS destination does not support transparency when the graph is rendered as a PNG image. Transparency is supported when the graph is rendered as vector-graphics output.

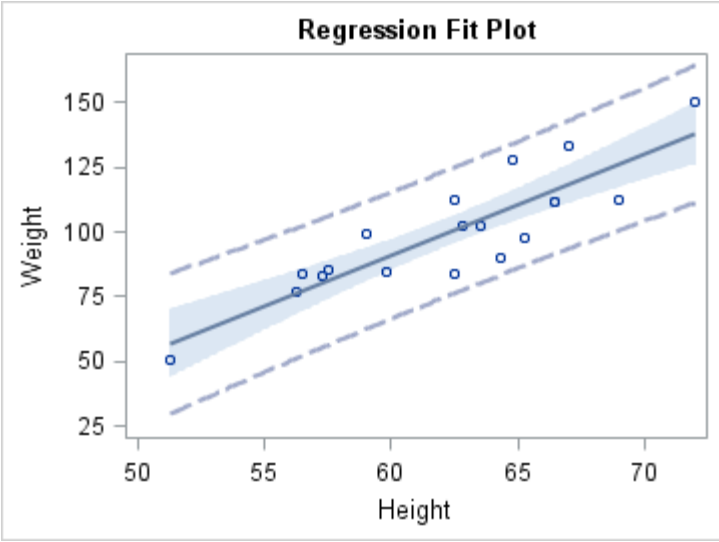
Transparency is useful in overlay situations where filled areas obscure underlying graph elements. In that case, you can make the filled areas semitransparent in order to allow the underlying elements to show through. You can also reduce the visual impact of certain elements in relation to others in order to draw attention to certain aspects of a graph. Many of the GTL plot statements provide the DATATRANSPARENCY= option, which enables you to adjust the transparency of the plot data elements. Many of the attribute options such as MARKERATTRS=, OUTLINEATTRS=, FILLATTRS= that are provided in GTL statements support a TRANSPARENCY= suboption. This suboption enables you to adjust the transparency of more specific elements. For information about how to adjust transparency by using these options, see *SAS Graph Template Language: Reference*.

Here is an example that specifies data transparency for model band plots that are overlaid with a scatter plot and a regression plot. Transparency is used to change the focus of the graph. Here is the SAS code.

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

/* Define the template for the graph */
proc template;
  define statgraph fit;
    begingraph;
      entrytitle "Regression Fit Plot";
      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;
    endgraph;
  end;
run;

/* Render the graph */
proc sgrender data=predict template=fit;
run;
```

Here is the output generated in the PNG format.



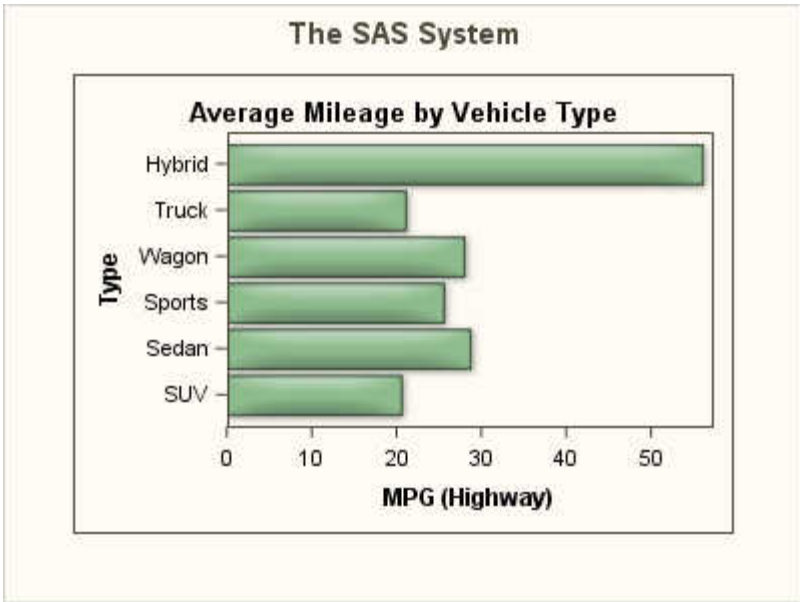
Setting the transparency of the model band data elements to 0.5 reduces their visual impact. Focus on the scatter plot markers and the regression line in the graph is increased.

Starting with SAS 9.4M3, you can use transparency to enable the background to show through a graph. To make the graph background transparent, use the OPAQUE=FALSE option in the BEGINGRAPH statement. To make the graph layout background transparent as well, use the WALLDISPLAY=NONE or WALLDISPLAY=(OUTLINE) option in the layout statement.

Here is an example.

```
proc template;
  define statgraph barchart;
    begingraph / opaque=false;
      entrytitle "Average Mileage by Vehicle Type";
      layout overlay / walldisplay=(outline);
        barchart category=type response=mpg_highway / name="barchart"
          fillattrs=(color=DarkSeaGreen)
          stat=mean orient=horizontal dataskin=matte;
      endlayout;
    endgraph;
  end;

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

Here is the output generated using the Analysis ODS style.



Using Data Skins

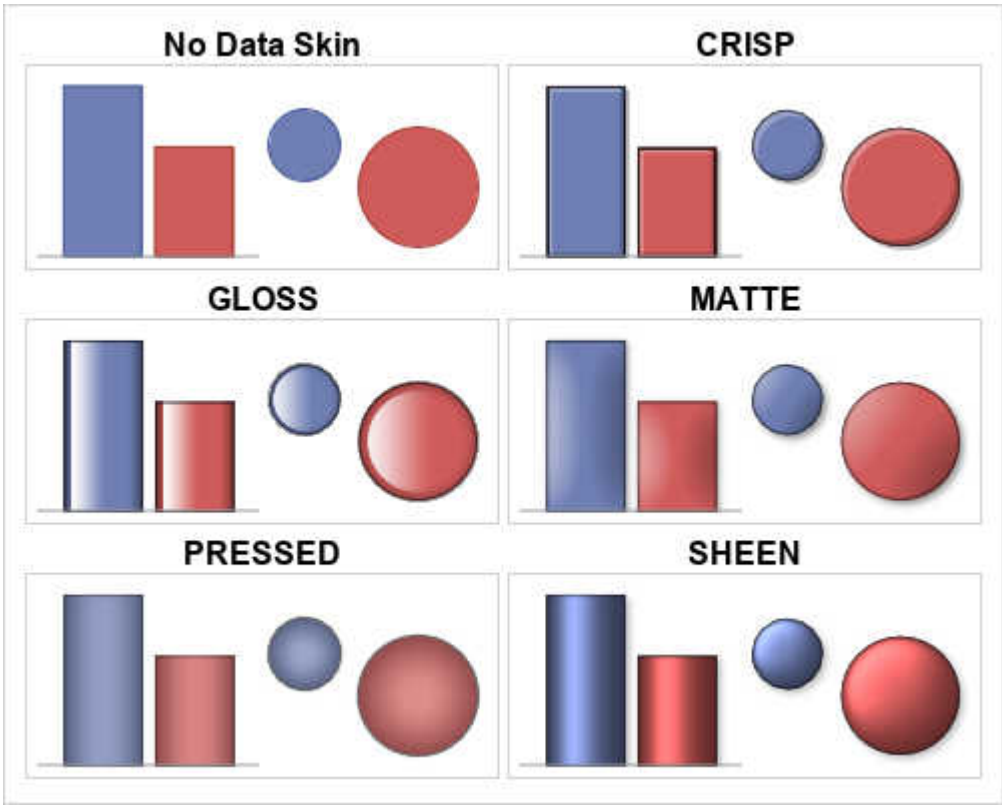
Data skins add a heightened visual effect to two-dimensional plots. Each skin uses shading, highlighting, and shadowing to give the appearance of contour and depth to certain elements of a graph, including the legend. For plots, the effect is generated by filters and is applied to filled areas, markers, and lines. When a data skin is applied to a filled area, it does not change the underlying fill color and pattern of the area. Typically, a data skin sets the area fill outline color to black. The outline color is controlled by the filters that generate the skin and is not controlled by the ODS style attributes or any custom outline attributes that are specified. For very small or very narrow filled areas, the data skin might not draw an outline around the filled area. For filled outlined markers, the outline color is determined by the ODS style attributes or by any custom marker attributes that are specified.

The effect that a data skin has on a filled area depends on the skin type, and on the size and color of the filled area. Because the ODS style determines the fill color by default, the effect can depend on the ODS style. Some skins have a greater effect than others. Most of the skins work best with lighter colors over a medium to large filled area. Over small filled areas and with some fill colors, the effect can be significantly reduced.

Note: Some ODS styles such as JOURNAL2 and MONOCHROMPRINTER use pattern fill for certain areas rather than color fill. For these styles, data skins have no effect on the pattern-filled areas.

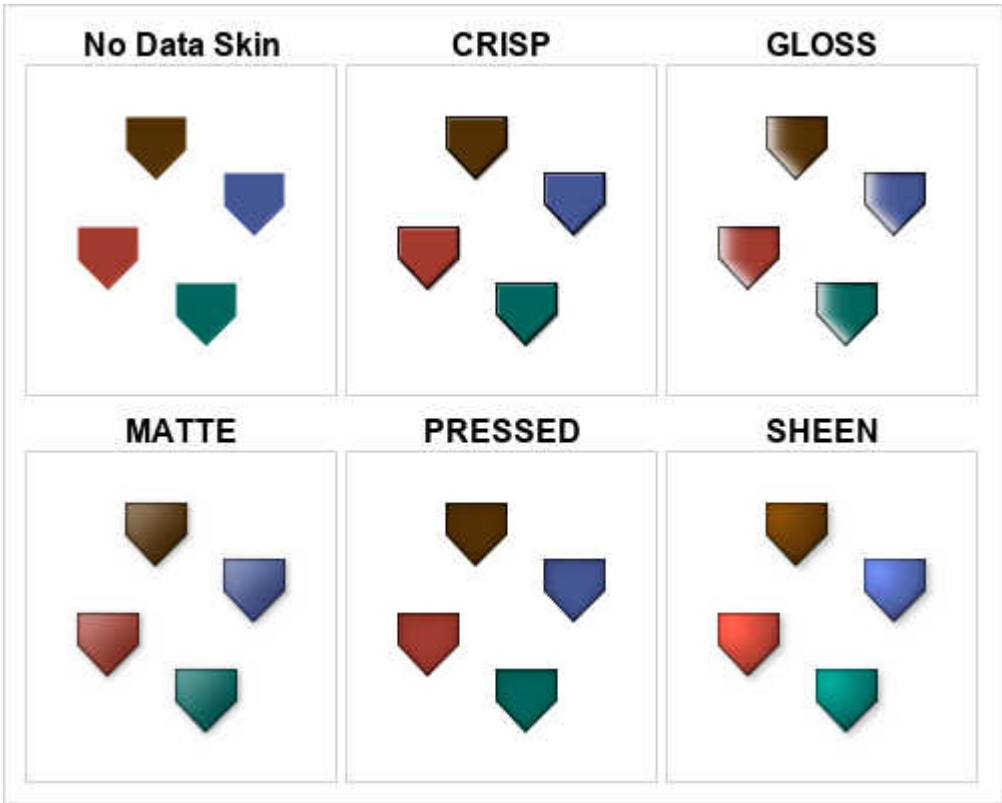
You can apply data skins to filled areas, markers, and lines in a plot. The data skins include CRISP, GLOSS, MATTE, PRESSED, and SHEEN. The following figure shows the effect of each data skin on filled bars and bubbles with the default HTMLBlue ODS style. A display with no data skin applied is included for comparison.

Figure 27.7 Data Skins Applied to Filled Bars and Bubbles



The next figure shows each of the data skins applied to large HOMEDOWNFILLED markers.

Figure 27.8 Data Skins Applied to HOMEDOWNFILLED Markers



The effect of a data skin on filled markers is more apparent when the markers are enlarged.

Except for the GLOSS data skin, the data skins also affect the appearance of plot lines and the outlines for unfilled markers and bubbles. They do not affect the outlines of unfilled bars, boxes, and so on. As with filled areas, the effect of data skins on lines varies with skin type and line color. It is also more apparent when the thickness of the lines is increased. The skins do not change the color of the lines. They add subtle effects such as drop shadows that enhance their appearance. The following figures show each of the data skins applied to plot lines, unfilled bubbles, and unfilled markers.

Figure 27.9 Data Skins Applied to Plot Lines

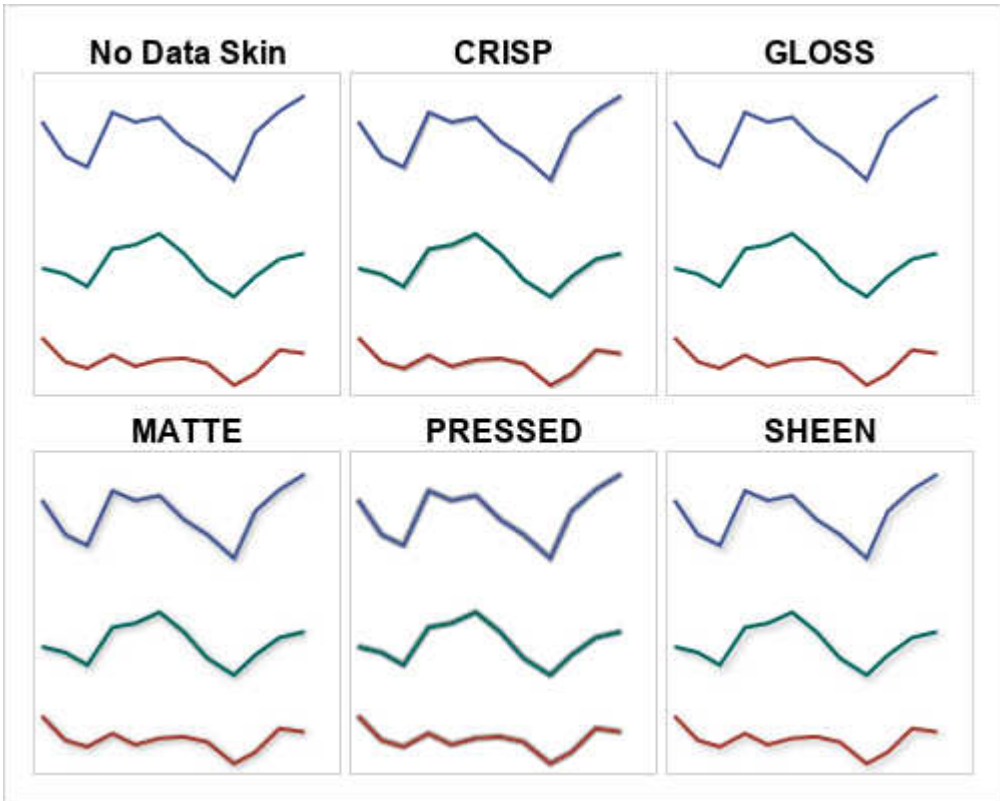


Figure 27.10 Data Skins Applied to Unfilled Bubbles

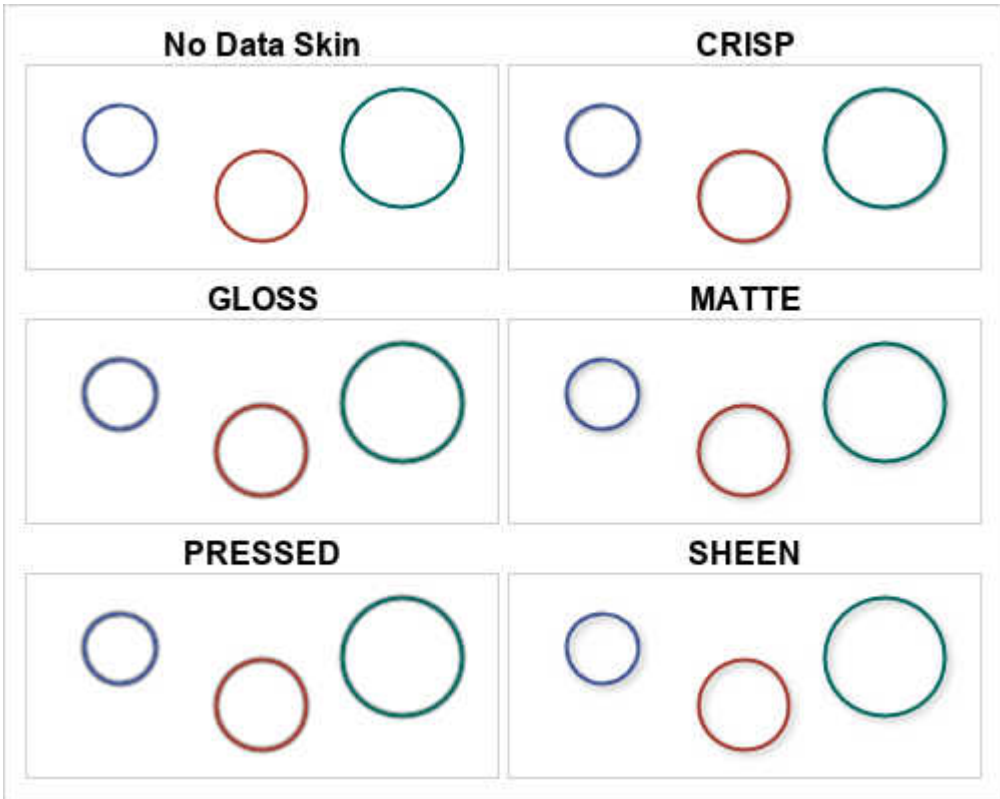


Figure 27.11 Data Skins Applied to HOMEDOWN Markers

You can specify data skins in the following GTL plot statements:

BARCHART	HISTOGRAM	SCATTERPLOT
BARCHARTPARM	HISTOGRAMPARM	SCATTERPLOTMATRIX
BOXPLOT	LINECHART	SERIESPLOT
BOXPLOTPARM	NEEDLEPLOT	STEPPLOT
BUBBLEPLOT	PIECHART	VECTORPLOT
DROPLINE	POLYGONPLOT	WATERFALLCHART
HIGHLOWPLOT	REFERENCELINE	

You cannot specify data skins for data-driven annotations or for graphical elements that are drawn using the GTL draw statements.

For all plots that support data skins, the GraphSkin:DataSkin style element in the active style specifies by default the data skin that is applied. For all of the plots in a template, you can use the DATASKIN= option in the BEGINGRAPH statement to override the data skin that is specified by the current style. For an individual plot, you can use the DATASKIN= option in the plot statement to override the data skin that is specified by the current style or by the BEGINGRAPH statement's DATASKIN= option. You can set the following values for the style GraphSkin:DataSkin element and the DATASKIN= options: NONE, SHEEN, GLOSS, PRESSED, CRISP, or MATTE.

In OVERLAY and PROTOTYPE layouts, the maximum number of skinned graphical elements is limited to 200 per plot for performance reasons. This limit does not apply to plots in other layout types. For graphs that contain multiple plots, this limit applies to each plot and not to the entire graph. A skinned graphical element can be a bar, bubble, marker, series line, and so on. It does not necessarily correlate with the number of observations in the plot data. If this limit is exceeded for a plot, the

specified data skin is not applied to that plot, and the following warning appears in the SAS log:

```
NOTE: Data skin has been disabled because the threshold has been
reached. You can set DATASKINMAX=nnn in the ODS GRAPHICS statement
to restore data skin.
```

In that case, you can use the DATASKINMAX= option in your ODS GRAPHICS statement to increase the threshold to the value specified in the note (*nnn*) or to a higher value.

Note: A plot that contains a large number of skinned graphical elements might take several minutes to render.

Using Anti-Aliasing

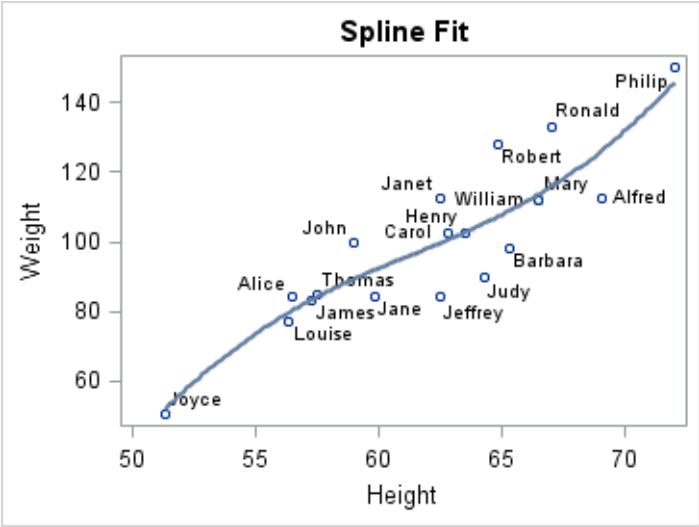
Anti-aliasing is a graphical rendering technique that improves the readability of text and the crispness of the graphical primitives, such as the markers and lines. By default, ODS Graphics uses anti-aliasing.

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.

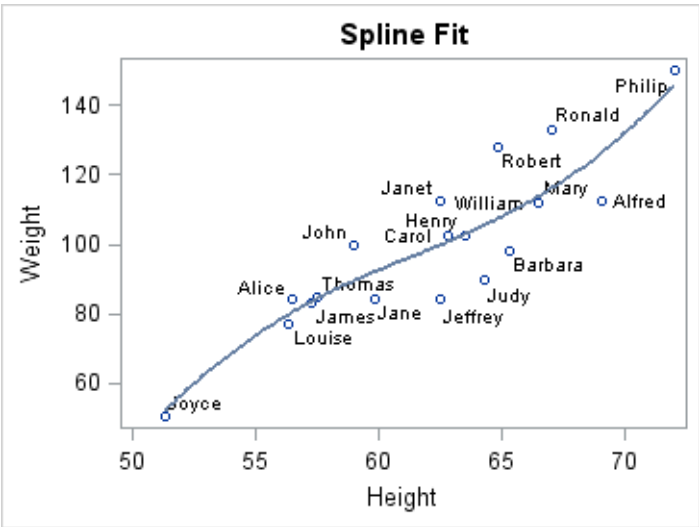
To see how much the graph quality is improved with anti-aliasing, you can turn this feature on and off with the ANTITALIAS= option in the ODS GRAPHICS statement.

```
proc template;
  define statgraph fitline;
    begingraph;
    entrytitle "Spline Fit";
      layout overlay;
        scatterplot x=height y=weight / primary=true datalabel=name;
        pbsplineplot x=height y=weight;
      endlayout;
    endgraph;
  end;
run;

ods graphics / antialias=on;
proc sgrender data=sashelp.class template=fitline;
run;
```

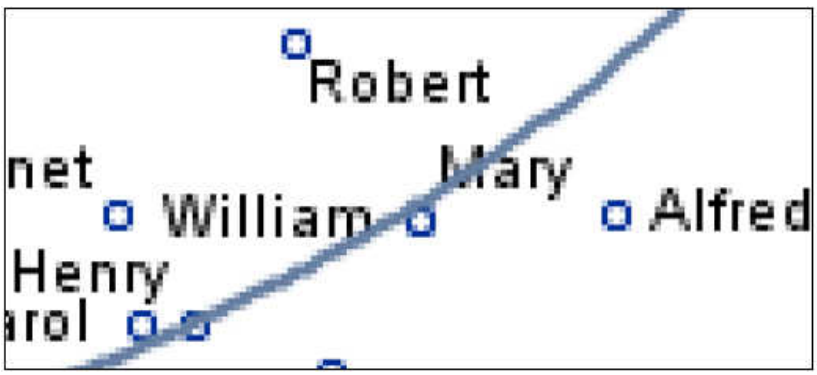
Here is the output.



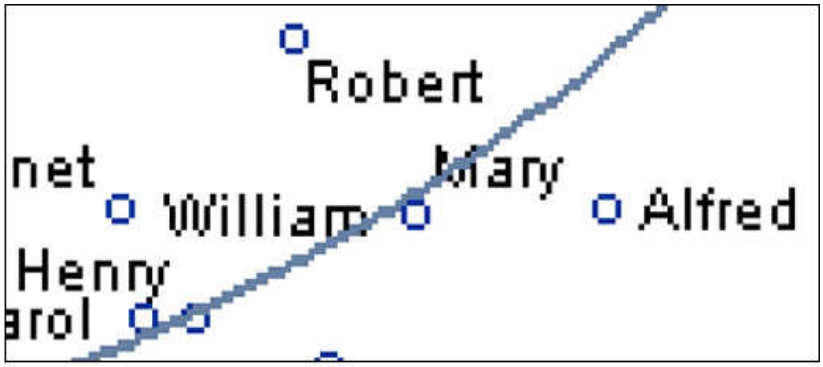
Here is the result when ANTIALIAS=OFF.



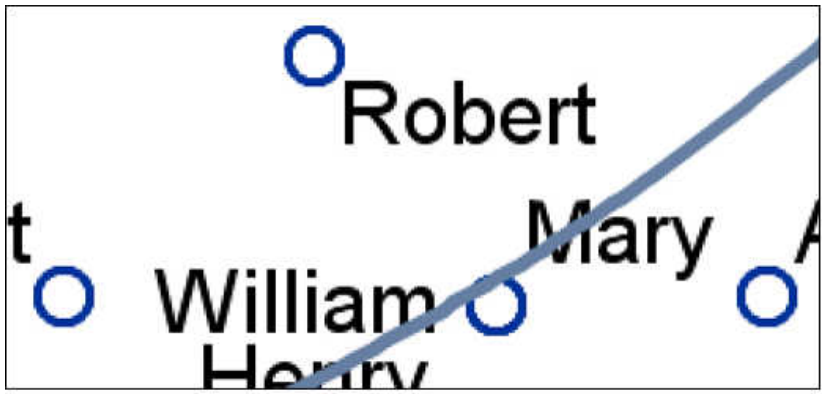
The following image shows a zoomed-in view of a portion of the anti-aliased image (100dpi). Notice that the text, markers, and line appear fuzzy because of the anti-aliasing algorithm.



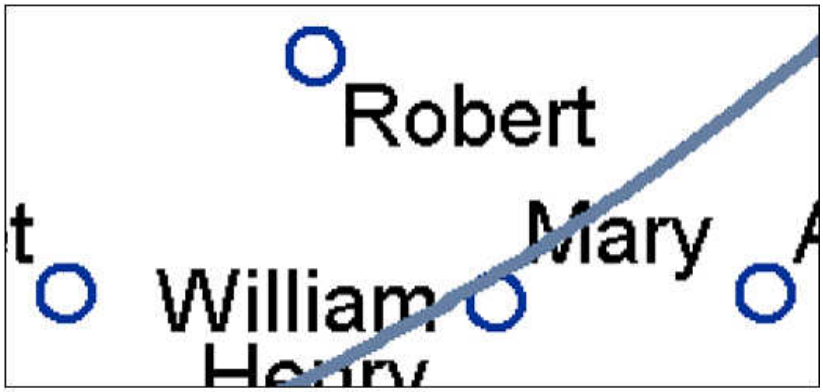
This next image shows a zoomed-in view of the image (100dpi) that has anti-aliasing turned off. Notice that the text, markers, and line are not fuzzy but have a jagged appearance.



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 3-D surface plots might require even more resources.

A higher DPI increases anti-aliasing resources. ODS Graphics disables the anti-aliasing feature when the resources required for anti-aliasing exceed a preset upper threshold. Prior to SAS 9.4M3, the threshold is based on the number of observations in the graph data. When the threshold is reached, anti-aliasing is disabled for the entire graph. Starting with SAS 9.4M3, the anti-aliasing upper threshold is based on the number of drawing elements in each plot. The threshold is enforced on a per-plot basis. When the number of drawing elements in a plot reaches the upper anti-aliasing threshold, anti-aliasing is disabled for that plot only. Anti-aliasing remains enabled for the remainder of the graph.

The anti-aliasing threshold is controlled by the ODS GRAPHICS statement ANTIALIASMAX= option. The default is 4000. When the anti-aliasing upper threshold is reached, a note is written to the SAS log indicating that anti-aliasing is disabled in all or part of the graph. The note also provides information about how to use the ANTIALIASMAX= option in an ODS GRAPHICS statement to raise the threshold sufficiently to re-enable anti-aliasing for all of the plots in the graph. For more information about the ODS GRAPHICS statement ANTIALIASMAX= option, see "ODS GRAPHICS" in *SAS Graph Template Language: Reference*.

Using Subpixel Rendering

You can specify subpixel rendering in order to generate smooth curves and more precise bar spacing in many plots. In SAS 9.4M2 and in earlier releases, you can use subpixel rendering for plots that are generated with the following statements:

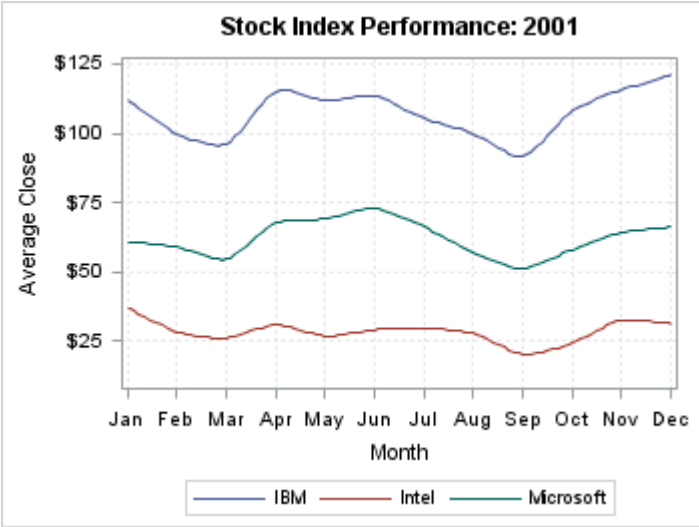
BANDPLOT	HIGHLOWPLOT	REGRESSIONPLOT
BARCHART	LINECHART	SERIESPLOT
BARCHARTPARM	LOESSPLOT	
DENSITYPLOT	PBSPLINEPLOT	

Starting with SAS 9.4M3, you can also use subpixel rendering for plots that are generated with the following statements:

BOXPLOT	HEATMAP	SCATTERPLOT
BOXPLOTPARM	HEATMAPPARM	SCATTERPLOTMATRIX
BUBBLEPLOT	HIGHLOWPLOT	WATERFALLCHART
CONTOURPLOTPARM	HISTOGRAMPARM	
DENSITYPLOT	POLYGONPLOT	

The SAS default rendering technology for ODS Graphics is used to render curved lines for GTL plots. For the Java technology, for example, subpixel rendering is disabled by default in SAS 9.4M2 and in earlier releases. Starting with SAS 9.4M3, subpixel rendering is always enabled for vector-graphics output. Prior to SAS 9.4M3, subpixel rendering is enabled by default for image output, unless the graph contains a scatter plot or a scatter-plot matrix. In those cases, subpixel rendering is disabled by default. Starting with SAS 9.4M3, subpixel rendering is enabled by default for all images.

When subpixel rendering is disabled, curved lines can appear slightly jagged. The following figure shows a series plot with smoothed lines (SMOOTHCONNECT=TRUE) that was rendered with Java technology, using the default line rendering.



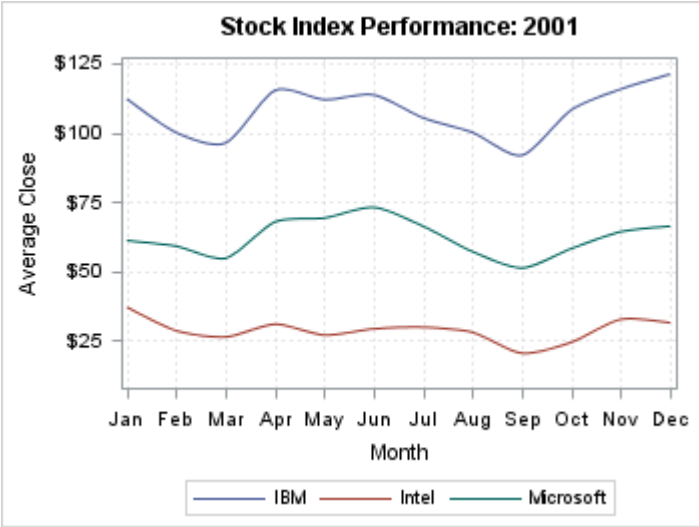
Notice that the line curves appear slightly jagged.

To enable subpixel rendering, include the SUBPIXEL=ON option in your BEGINGRAPH statement. Starting with SAS 9.4M3, you can also use the SUBPIXEL= option in an ODS GRAPHICS statement to control subpixel rendering for images. In order to use subpixel rendering, anti-aliasing must also be enabled. Anti-aliasing is enabled by default. If it is disabled, include the ANTIALIAS=ON option in your ODS GRAPHICS statement.

Anti-aliasing is disabled automatically when the resources required for anti-aliasing exceed a preset threshold. When anti-aliasing is disabled for all or part of a graph, subpixel rendering is disabled for the entire graph. The anti-aliasing threshold is controlled by the ODS GRAPHICS statement ANTIALIASMAX= option. When the anti-aliasing threshold is reached, a note is written to the SAS log stating that anti-aliasing is disabled. The note recommends a new value for the ANTIALIASMAX= option that you can use to re-enable anti-aliasing. When you re-enable anti-aliasing, you also re-enable subpixel rendering in this case.

For more information about the ODS GRAPHICS statement SUBPIXEL=, ANTIALIAS=, and ANTIALIASMAX= options, see “ODS GRAPHICS” in *SAS Graph Template Language: Reference*.

The next figure shows the same series plot generated with subpixel rendering enabled, anti-aliasing enabled, and smoothed lines.



Notice that the line curves are much smoother. Here is the SAS code for the previous graph.

```
/* Sort the SASHELP.STOCKS data by date. */
proc sort data=sashelp.stocks out=stocks;
  by date;
run;

/* Create the template for the graph. */
proc template;
  define statgraph subpixelon;
     begingraph / subpixel=on;
     entrytitle "Stock Index Performance: 2001";
     layout overlay /
       xaxisopts=(label="Month" griddisplay=on
         gridattrs=(pattern=dot color=lightgray))
       yaxisopts=(label="Average Close" type=linear
         linearopts=(viewmin=10) griddisplay=on
         gridattrs=(pattern=dot color=lightgray));
       seriesplot x=eval(put(date, monname3.)) y=close /
         name="stocks"
         group=stock
         smoothconnect=true;
       discretelegend "stocks";
     endlayout;
   endgraph;
 end;

 /* Render the graph for the year 2001. */
proc sgrender data=stocks template=subpixelon;
  where year(date) = 2001;
run;
```

To disable subpixel rendering for all images, specify SUBPIXEL=OFF in an ODS GRAPHICS statement. When subpixel rendering is disabled for all images, you can specify the SUBPIXEL=ON option in a template's BEGINGRAPH statement to re-enable subpixel rendering only for that template.

Recommendations

The issue of when to use hardcoded 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 with one style in mind and use hardcoded 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 will not be much conflict with blacks and grays coming from the style. You really do not care what the graph looks like with another style.

- You are creating a reusable graph template (without hardcoded 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 hardcoded overrides. This allows 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 “Using ODS Styles to Control Graph Appearance” on page 495 for more information.

28

Managing Your Graphics Output

Introduction to ODS Graphics Output

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:

Area	How It Controls 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 item store file that is stored in your SASUSER storage location. It contains the default settings for many SAS products and their features. You can use the REGISTRY procedure to browse or edit this hierarchical file. Here is an example of how to use the REGISTRY procedure to print the current ODS Graphics registry settings to the SAS log.

```
proc registry list startat="ods\ods graphics";
run;
```

Here is the output.

```
NOTE: Contents of SASHELP REGISTRY starting at subkey [ods\ods graphics]
[  ods\ods graphics]
  Default State="Off"
  Design Height="480PX"
  Design Width="640PX"
```

Note: In the SAS windowing environment, you can also use the Registry Editor window to browse the Registry interactively. To do so, issue the REGEDIT command to open the Registry Editor window, expand **ODS**, and then click **ODS GRAPHICS**.

The following table describes how the ODS GRAPHICS registry keys are used.

Registry Key	Use
Default State	Determines whether the ODS Graphics environment is active by default
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 are automatically produced every time you run an ODS Graphics-enabled procedure such as UNIVARIATE, ARIMA, or REG.

Note: The SAS procedures such as SGRENDER, SGPLOT, SGPANEL, and SGSCATTER produce template-based graphics only. 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 581.

ODS Destination Statement Options That Affect 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 28.1 *ODS Destination Options That Affect ODS Graphics*

ODS Destination	Options for ODS Graphics	Description
LISTING		Creates a standalone image. The default image format is PNG.

ODS Destination	Options for ODS Graphics	Description
		(See also "Using a Universal Printer with ODS PRINTER to Control the Image Format" on page 578.)
	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=96 is the default. **Note:** For TIFF output in SAS 9.4M2 and in earlier releases, the DPI property for the TIFF image might show 100 DPI instead of the IMAGE_DPI= setting. The actual image resolution is the IMAGE_DPI= setting. This issue is fixed in SAS 9.4M3.
	STYLE= *style-definition*	Specifies the style to use. STYLE=LISTING is the default.
PDF		Creates one or more embedded images in a PDF document. The default image format is SVG.
	DPI=*number*	Specifies the resolution in dots per inch for image output. DPI=150 is the default.
	STYLE= *style-definition*	Specifies the style to use. STYLE=PEARL is the default.
RTF		Creates one or more embedded images 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=200 is the default.
	STYLE= *style-definition*	Specifies the style to use. STYLE=RTF is the default.
HTML		Creates one or more standalone images and the HTML page. The images are referenced in the

ODS Destination	Options for ODS Graphics	Description
		HTML page. The default image format is PNG.
	FILE="*filename*.html"	Specifies the name of the HTML output file.
	PATH="*directory-spec*"	Specifies the directory where the HTML output and the images are created. Use the GPATH= option to specify a different directory for the images.
	GPATH="*directory-spec*"	Specifies the directory where the 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=96 is the default.
	STYLE= *style-definition*	Specifies the style to use. HTMLBLUE is the default style in the SAS windowing environment and in SAS Studio.

ODS GRAPHICS Statement Options

The ODS GRAPHICS statement is the primary statement that controls the run-time environment for producing template-based graphs. It is 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 statement does not affect GRSEG-based graphs.

All options for the ODS GRAPHICS statement are global to a SAS session, unless one of the following occurs:

- The graphics environment is disabled with the ODS GRAPHICS OFF statement
- The ODS Graphics options are reset to their default state with the ODS GRAPHICS RESET or ODS GRAPHICS RESET=*option* statement

The following table shows some of the available options. For a complete and more detailed explanation of all available options, see “ODS GRAPHICS” in *SAS Graph Template Language: Reference*.

Table 28.2 Partial Listing of ODS GRAPHICS Statement Options

Task	Option
Specify the threshold for allowing anti-aliasing.	ANTIALIASMAX= *positive-integer* The default is 4000.
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.
Increase the maximum number of discrete values that are allowed in a graph.	DISCRETEMAX=*positive-integer* The default is 1000. If your graph data contains more than 1000 discrete values, your graph is not drawn and a warning message is written to the SAS log stating that the DISCRETEMAX threshold has been reached. In that case, use the DISCRETEMAX= option to increase the maximum number of discrete values that are allowed. Starting with SAS 9.4M5, the log message includes a suggested value for DISCRETEMAX=.
Increase the maximum number of group values that are allowed in a graph.	GROUPMAX=*positive-integer* The default is 1000. If your graph data contains more than 1000 group values, the plot GROUP= option is ignored, and a warning message is written to the SAS log stating that the GROUPMAX threshold has been reached. In that case, use the GROUPMAX= option to increase the maximum number of group values that are allowed. Starting with SAS 9.4M5, the log message includes a suggested value for GROUPMAX=.
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. You should always provide a unit such as PX, IN, CM, or MM with the dimension value.
Specify the image format used to generate image files.	OUTPUTFMT= STATIC | *image-format* Supported formats include PNG, GIF, JPEG, WMF, TIFF, PDF, EMF, PS, EPS, EPSI, PCL, SVG, and others. The keyword STATIC is the default, which means to automatically select the best format, based on the output destination. **Note:** The PDF, PS, EPS, EPSI, EMF, and PCL formats support images in the vector graphics format.

Task	Option
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 legends are drawn.	LEGENDAREAMAX=**n** 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. Use the MAXLEGENDAREA= option in SAS releases prior to SAS 9.4M3. Use the LEGENDAREAMAX= option starting with SAS 9.4M3. **Note:** You can continue using the MAXLEGENDAREA= option in SAS 9.4M3 and in later releases. However, the LEGENDAREAMAX= option is preferred.
Reset one or more ODS GRAPHICS options to its default. RESET by itself is the same as RESET=ALL.	RESET | RESET= *option* Options include ALL, HEIGHT, WIDTH, INDEX, and so on. 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 resets to the base name without the increment number. This is useful if you run a PROC several times and are interested only in the images from the last run (the previous ones are 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 whether the plot markers are scaled with the graph size.	SCALEMARKERS=YES | NO | ON | OFF The default is ON. The scale factor is based on the height of the graph cells and the height of the graph.
Specify the maximum number of distinct hotspot areas that are allowed before data tips are disabled.	TIPMAX=*n* The default number is 500.
Specify the width of any graph.	WIDTH= *dimension* Supported dimension units include SPX (special pixels), PX (pixels), IN (inches), CM (centimeters), and

Task	Option
	MM (millimeters). This option overrides the design width specified by the template definition. The default unit is SPX. You should always provide a unit such as PX, IN, CM, or MM with the dimension value.

Controlling the Image Name and Image Format

Specifying and Resetting the Image Name

Specifying the Image Name

For ODS Graphics output, by default, the ODS object name is used as the “root” name for the image output file. The following example creates a GIF image named REGPLOT:

```
ods graphics / imagename="regplot" outputfmt=gif;
```

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. This numbering can be reset with the RESET=INDEX option.

Resetting the Image Name

The RESET=INDEX option enables you to reset the file name numbering sequence. This feature applies to SAS 9.4M3 and later releases.

For a usage example, suppose that you are developing a paper or a presentation and it takes several submissions to get the desired output. You can use the RESET or RESET=INDEX 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.

When specifying this option, you can also specify the value for the index counter. The value that you specify determines the suffix for the next subsequent image. For example:

```
ods graphics / reset=index(100) imagename="MyName";
```

The next graph that you produce is named MYNAME100.

This feature is useful for creating animated graphics. For example, for a sequence of 100 images, you might begin with the following statement:

```
ods graphics / reset=index(1) imagename="MyName";
```

In the example, your program produces 100 images named MYNAME1, MYNAME2, ..., MYNAME100. If you later add more images to the animation, you might submit the following:

```
ods graphics / reset=index(101) imagename="MyName";
```

The next generated image is named MYNAME101.

Specifying the Image Format

Each ODS destination uses a default format for its output. You can use the OUTPUTFMT= option in the ODS GRAPHICS statement to change the output format.

Note: Unless you have a special requirement for changing the image format, we recommend that you not change it. The default PNG or vector graphic format is far superior to other formats, such as GIF, in support for transparency and a large number of colors. Also, PNG and vector graphics images require much less disk storage space than JPEG or TIFF formats.

If you want to generate vector graphics images, you can use the following OUTPUTFMT= values for each destination:

Table 28.3 *Generating Vector Graphics Output with ODS*

ODS Destination	OUTPUTFMT=value
ODS EPUB	OUTPUTFMT=SVG
ODS destination for Excel	OUTPUTFMT=EMF
ODS HTML	OUTPUTFMT=SVG
ODS HTML5	Vector graphics images are generated by default
ODS LISTING	OUTPUTFMT=EMF OUTPUTFMT=PDF OUTPUTFMT=PS | EPS | EPSI OUTPUTFMT=SVG OUTPUTFMT=PCL
ODS PDF	Vector graphics images are generated by default

ODS Destination	OUTPUTFMT=value
ODS PCL	OUTPUTFMT=PCL (for PCL output)
ODS PS	OUTPUTFMT=PS (for PostScript output)
ODS destination for PowerPoint	OUTPUTFMT=EMF
ODS PRINTER	OUTPUTFMT=PCL (for PCL output) OUTPUTFMT=PDF (for PDF output) OUTPUTFMT=PS | EPS | EPSI (for PostScript output) OUTPUTFMT=SVG
ODS RTF	OUTPUTFMT=EMF
ODS Measured RTF	OUTPUTFMT=EMF

Note: For information about SVG and SAS Universal Printing, see "Creating SVG Documents Using Universal Printing" in *SAS 9.4 Universal Printing*.

When a vector graphics image cannot be generated for the format that you specify, a PNG image is generated instead and is embedded in the specified output file. The output file format and extension are not changed in that case. In the following cases, a vector graphics image cannot be generated:

- surface plots
- bivariate histograms
- graphs that use smooth gradient contours
- graphs that include continuous legends
- graphs that use data skins
- graphs that use transparency (EMF and PS ODS destinations only)
- graphs that contain one or more rotated images
- graphs that have a broken axis
- graphs that contain outline marker characters

Starting with SAS 9.4M2, additional cases for which vector graphics output cannot be generated for graphs are as follows:

- graphs that use gradient fill for bars in a bar chart, histogram, or waterfall chart
- graphs that use the back-light effect on text
- graphs that include a text plot that displays text with an outlined bounding box or text with a filled bounding-box background
- graphs that include images (PostScript output only)

Starting with SAS 9.4M3, vector graphics output can be generated in the EMF, PDF, and SVG output formats for the following cases:

- graphs that use data skins

  Note: For the EMF, PDF, and SVG formats, vector graphics output is not supported for graphs that use transparency and data skins. An image is generated in that case.

- graphs that include one or more rotated images
- graphs that use gradient fills (except PDF)
- graphs that use a continuous legend

  Note: For the PDF output format, vector graphics output is not supported for graphs that use a continuous legend and data transparency. An image is generated in that case.

Starting with SAS 9.4M5, vector graphics output can or cannot be generated in the output formats as indicated in the following table:

Table 28.4 *Support for Vector Graphics*

Case	EMF	PCL	PDF	PS	SVG
Surface plots	No	No	No	No	No
Gradient fill contour plots	No	No	No	No	No
3-D bivariate histograms	No	No	No	No	No
Continuous legends without data or fill transparency	Yes	No	Yes	No	Yes
Continuous legends with data transparency	Yes	No	No	No	Yes
Data skins	Yes	No	Yes	No	Yes
Data skins with transparency	No	No	No	No	Yes
Data transparency, without images	Yes	Yes	Yes	No	Yes
Transparent images[1]	Yes	No	Yes	No	Yes
Images with data transparency	No	No	No	No	Yes
Rotated images	Yes	No	Yes	No	Yes
Text plots with backlight	Yes	No	Yes	No	Yes
Text plots with fill and outline	Yes	No	Yes	Yes	Yes
Scatter plots with an outlined marker character	Yes	No	Yes	No	Yes
Broken axis	Yes	No	Yes	No	Yes

Case	EMF	PCL	PDF	PS	SVG
Fill patterns	Yes	No	No	No	Yes
Fill pattern legend chiclets	No	No	No	No	No

1 Starting with SAS 9.4M7, vector graphics are supported for transparent images in EMF, PDF, and SVG output. The image itself must be transparent. If you make an image transparent by specifying transparency in your program, a vector graphic is not created.

Note: The SGMAP procedure, which is new in SAS 9.4M5, does not support vector-based output.

Using a Universal Printer with ODS PRINTER to Control the Image Format

When you use the ODS PRINTER destination, you can use the PRINTER= option to specify a universal printer to generate the image. The universal printers can generate image files in formats other than PNG, such as SVG, PDF, or EMF. For more information about the ODS PRINTER destination, see *SAS Output Delivery System: User's Guide*.

To use a universal printer with the ODS PRINTER destination to generate an image file, do the following:

1. Create a template to generate your graph if you have not already done so.
2. Use a FILENAME statement to create a file reference for your image output file.
3. Close the currently open ODS destinations.

   ```
   ods _all_ close;
   ```

4. Open the ODS PRINTER destination and specify the SVG printer.

   ```
   ods printer printer=svg file=filref;
   ```

5. Use the SGRENDER procedure to generate your graph.
6. Close the ODS PRINTER destination.
7. Open an ODS destination for subsequent programs.

   Note: This step is not required in SAS Studio.

Here is an example that generates a graph in the SVG format using the ODS LISTING destination.

```
/* Specify the path and name for the image output file */
filename imgout "./barchart.svg";

/* Create the graph template */
proc template;
```

```
  define statgraph barchart;
    begingraph;
      dynamic printer dev imagedpi;
      entrytitle "Average Mileage by Vehicle Type";
      layout overlay;
        barchart category=type  response=mpg_highway /
          stat=mean orient=horizontal;
      endlayout;
    endgraph;
  end;
run;

/* Close the currently open ODS destinations */
ods _all_ close;

/* Open the ODS PRINTER destination */
ods printer printer=svg file=imgout;

/* Generate the graph */
proc sgrender data=sashelp.cars template=barchart;
run;

/* Close the ODS PRINTER destination */
ods printer close;

/* Open an ODS destination for subsequent programs. (Not
   required in SAS Studio.) */
ods html;
```

Controlling the Location of the Image Output

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\images";
```

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`**) suboption 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 _all_ close;
ods html style=statistical
  path="u:\public_html"
  gpath="u:\public_html" (url=none)
```

```
  file="report.html";

/* Create the graph template */
proc template;
  define statgraph barchart;
    begingraph;
      dynamic mpgtype;
      entrytitle "Average " mpgtype " MPG by Vehicle Type";
      layout overlay;
        /* Generate graph based on MPG type */
        if (upcase(mpgtype) = "CITY")
          barchart category=type  response=mpg_city /
            stat=mean orient=horizontal;
        else
          barchart category=type  response=mpg_highway /
            stat=mean orient=horizontal;
        endif;
      endlayout;
    endgraph;
  end;
run;

/* Generate city MPG chart */
proc sgrender data=sashelp.cars template=barchart
  des="Average City MPG by Type";
  dynamic mpgtype="City";
run;

/* Generate highway MPG chart */
proc sgrender data=sashelp.cars template=barchart
  des="Average Highway MPG by Type";
  dynamic mpgtype="Highway";
run;

ods html close;
ods html; /* Not required in SAS Studio */
```

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="Average City MPG by Type" src="graph.png" style=" height: 480px;
  width: 640px;" border="0" class="c">

<img alt="Average Highway MPG by Type" src="graph1.png" style=" height:
  480px; width: 640px;" border="0" class="c">
```

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 of Graph Size Control

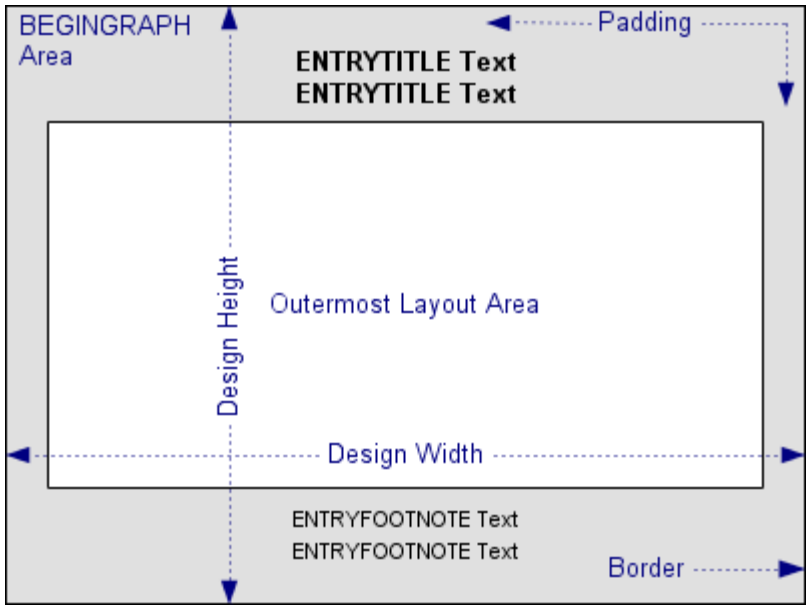
By default, the size of the graph that you create with ODS Graphics is governed by the following:

- 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), which creates a 480px by 480px graph.

```
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;

proc sgrender data=sashelp.cars template="squareplot";
  dynamic title="Square Plot" xvar="mpg_highway" yvar="mpg_city";
run;
```

If a 550px width or height is set in an ODS GRAPHICS statement before the template is executed with the SGRENDER procedure, a 550px by 550px graph is created, maintaining the 1:1 aspect ratio:

```
ods graphics / width=550px;
proc sgrender data=sashelp.cars template="squareplot";
  dynamic title="Square Plot" xvar="mpg_highway" yvar="mpg_city";
run;

/* Setting a 550px height would create the same size graph */
ods graphics / height=550px;
proc sgrender data=sashelp.cars template="squareplot";
  dynamic title="Square Plot" xvar="mpg_highway" yvar="mpg_city";
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 14, “Creating Gridded Graphs Using the GRIDDED Layout ,” on page 207, Chapter 15, “Creating Lattice Graphs Using the LATTICE Layout,” on page 221, and Chapter 16, “Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts,” on page 255.

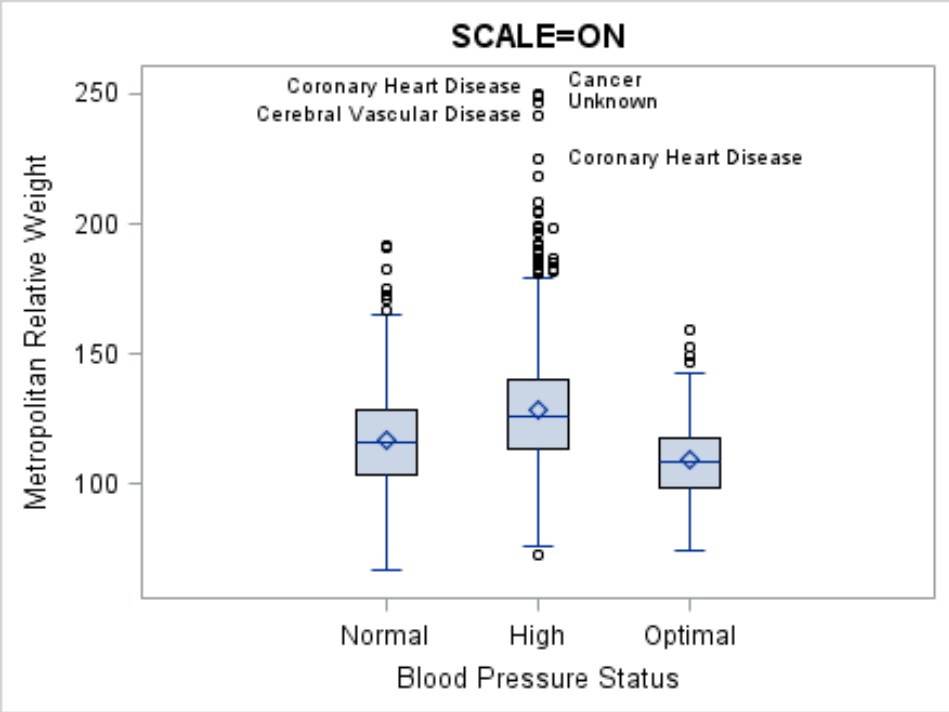
Scaling Graphs

ODS Graphics uses style information to control the appearance of the graph. Style templates 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. Consider the following example.

```
proc template;
  define statgraph boxplot;
```

```
    dynamic title;
    begingraph / designwidth=640 designheight=480;
      entrytitle title;
      layout overlay;
        boxplot y=mrw x=bp_status / datalabel=deathcause
labelfar=true;
      endlayout;
    endgraph;
  end;
run;
ods graphics / reset width=440px height=330px scale=on;
proc sgrender data=sashelp.heart template=boxplot;
  where status="Dead";
  dynamic title="SCALE=ON";
run;
```

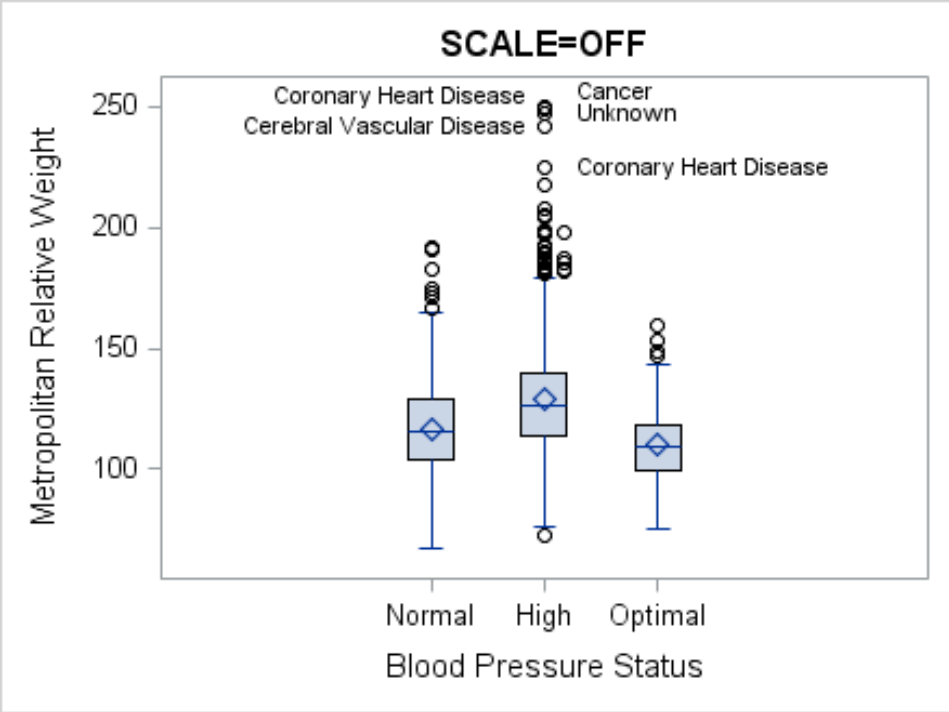
Here is the output.



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 / reset width=440px height=330px scale=off;
proc sgrender data=sashelp.heart template=boxplot;
  where status="Dead";
  dynamic title="SCALE=OFF";
run;
```

Here is the result.



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.

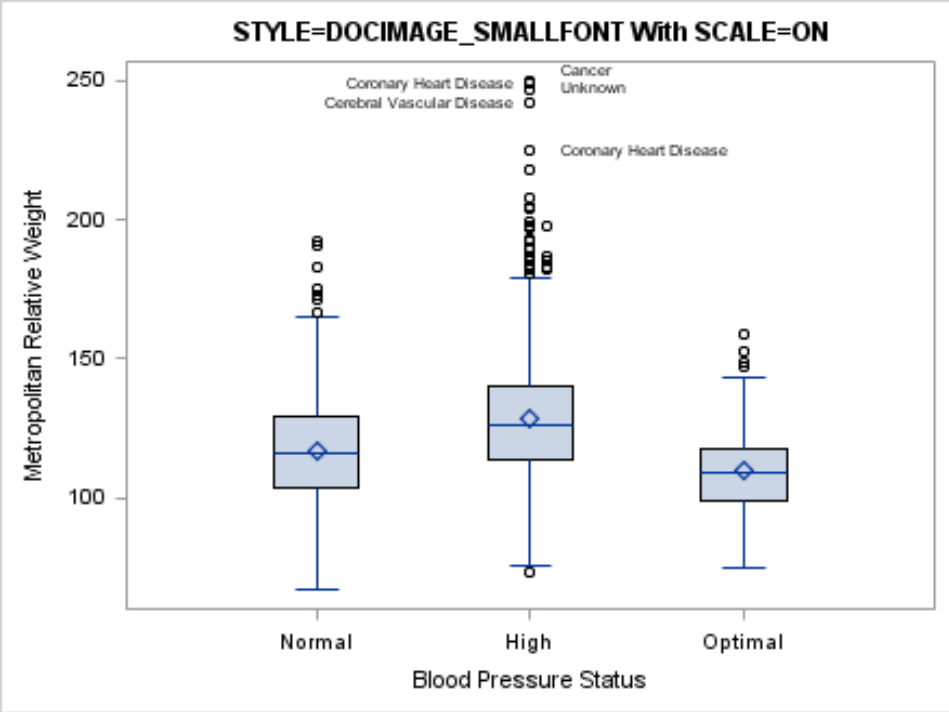
The examples in this document were created with different styles that varied only in the font sizes that they used. In some cases, smaller graphs look better when rendered in a smaller set of fonts. The style examples below use the HTMLBLUE style as a parent, but you could use any style as the parent. The DOCIMAGE style keeps fonts close to the default sizes and weights while the DOCIMAGE_SMALLFONT style reduces the font sizes by a few points. See "Using ODS Styles to Control Graph Appearance" on page 495 for a discussion of defining your own styles and what parts of the graph are affected by various style elements.

```
proc template;
  define style Styles.docimage;
    parent=styles.htmlblue;
    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>",11pt)
      'GraphFootnoteFont'=("<sans-serif>,<MTsans-serif>",8pt)
      'GraphTitleFont'=("<sans-serif>,<MTsans-serif>",12pt,bold);
  end;

  define style Styles.docimage_smallfont;
    parent=styles.htmlblue;
    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>",9pt)
```

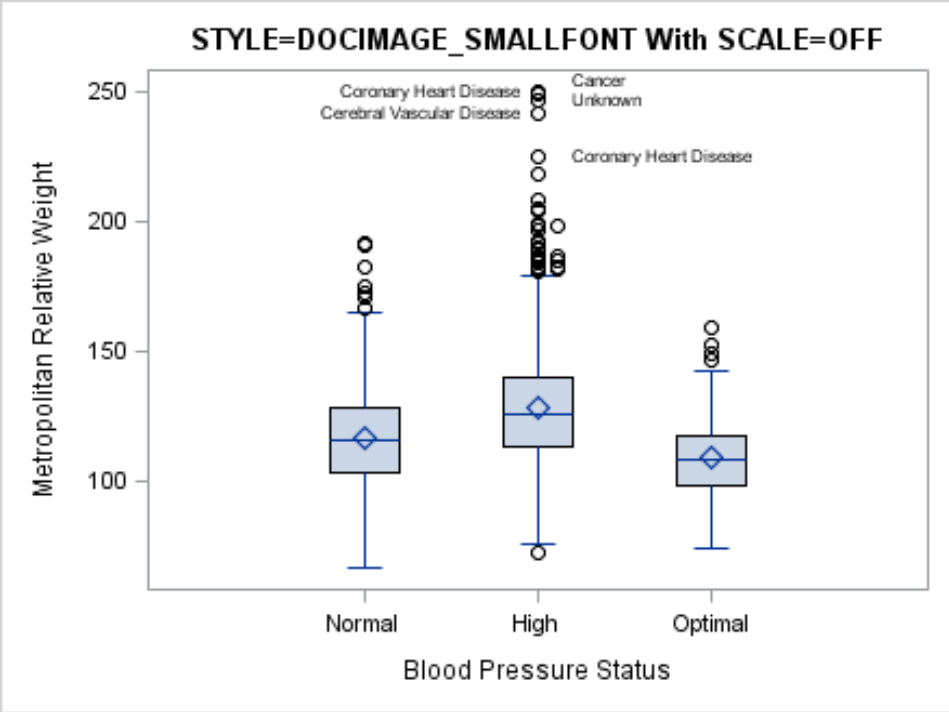
```
        'GraphFootnoteFont'=("<sans-serif>,<MTsans-serif>",8pt)
        'GraphTitleFont'=("<sans-serif>,<MTsans-serif>",10pt,bold);
    end;
run;
```

The previous two graphs were rendered using the DOCIMAGE style. These next two graphs were rendered with the DOCIMAGE_SMALLFONT style.

```
/* Specify a path for the ODS output */
filename odsout "output-path";
ods graphics / reset width=440px height=330px scale=on
imagename="smallfont";
ods _all_ close;
ods html path=odsout file="smallfont.html" style=docimage_smallfont;
proc sgrender data=sashelp.heart template=boxplot;
  where status="Dead";
  dynamic title="STYLE=DOCIMAGE_SMALLFONT With SCALE=ON";
run;
ods html close;
ods html; /* Not required in SAS Studio */
```




```
ods graphics / reset width=440px height=330px scale=off;
ods html style=docimage_smallfont;
proc sgrender data=sashelp.heart template=boxplot;
  where status="Dead";
  dynamic title="STYLE=DOCIMAGE_SMALLFONT With SCALE=OFF";
run;
```



In both of these graphs that use the DOCIMAGE_SMALLFONT 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.

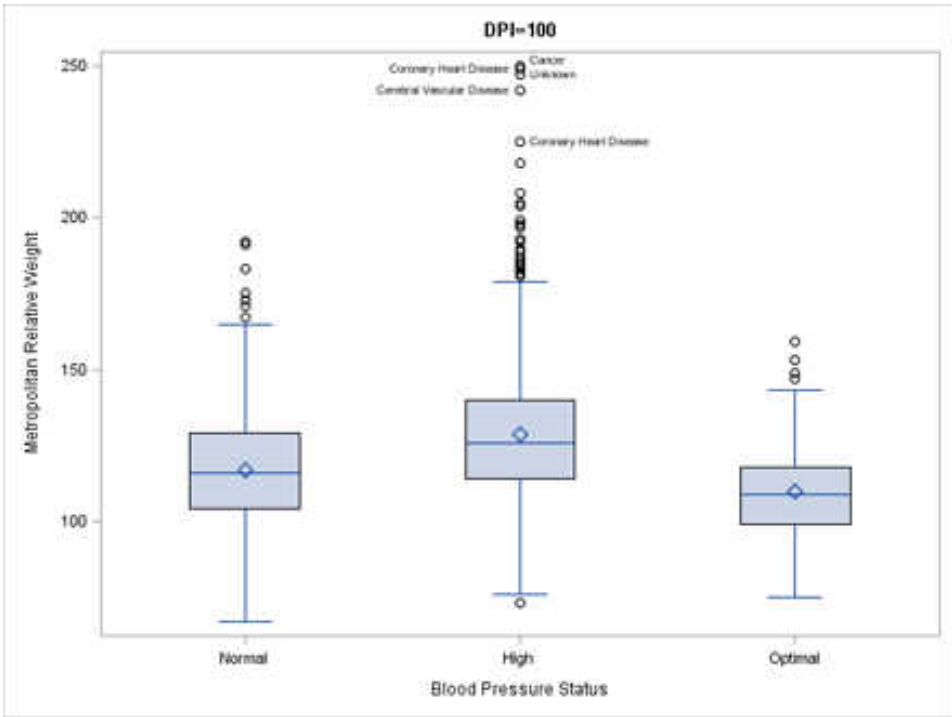
Controlling Image Resolution

All ODS destinations use a default DPI (dots per inch) setting when creating ODS Graphics image output. By default, LISTING and HTML use 96 DPI, PDF uses 150 DPI, and RTF uses 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 300 DPI 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, HTML, and RTF destinations, use the IMAGE_DPI= option. For PDF destination, use the DPI= option. Here is an example for the HTML destination that sets the image DPI to 100.

```
proc template;
  define statgraph boxplot;
    dynamic title;
    begingraph / designwidth=640 designheight=480;
      entrytitle title;
      layout overlay;
        boxplot y=mrw x=bp_status / datalabel=deathcause
labelfar=true;
      endlayout;
    endgraph;
  end;
run;
```

```
/* Specify a path for the ODS output */
filename odsout "output-path";
ods graphics / width=440px height=330px scale=off
imagename="smallfont100";
ods _all_ close;
ods html image_dpi=100 style=docimage_smallfont path=odsout
   file="smallfont100.html";
proc sgrender data=sashelp.heart template=boxplot;
  where status="Dead";
  dynamic title="DPI=100";
run;
ods html close;
ods html; /* Not required in SAS Studio */
```

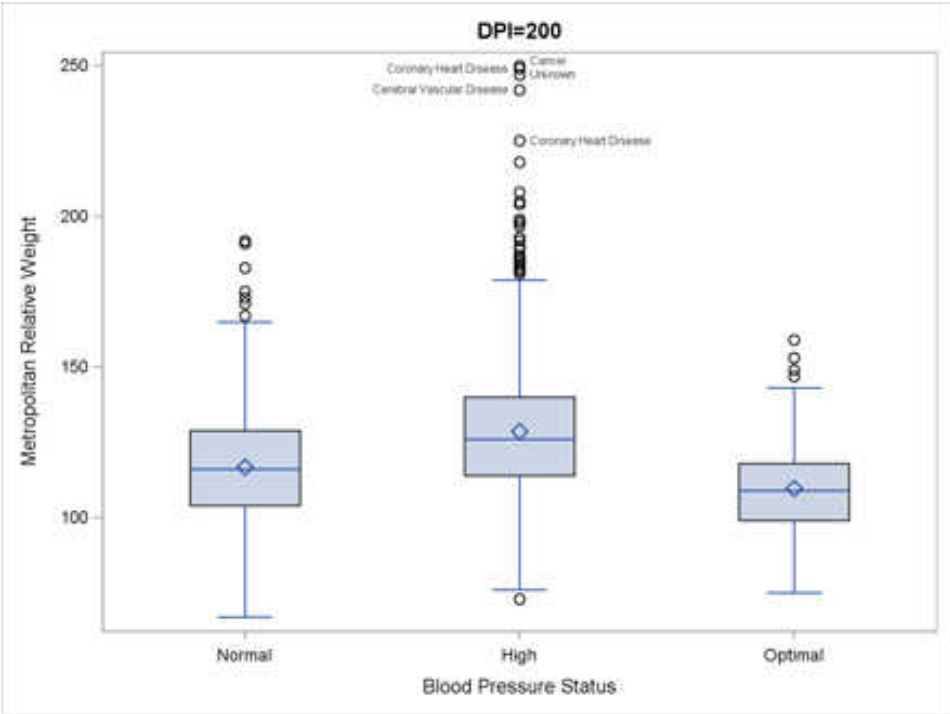
Here is the output.



Here is an example that sets the image DPI to 200.

```
/* Specify a path for the ODS output */
filename odsout "output-path";
ods graphics / width=440px height=330px scale=off
imagename="smallfont200";
ods _all_ close;
ods html image_dpi=200 style=docimage_smallfont path=odsout
   file="smallfont200.html";
proc sgrender data=sashelp.heart template=boxplot;
  where status="Dead";
  dynamic title="DPI=200";
run;
ods html close;
ods html; /* Not required in SAS Studio */
```

Here is the output.



In these examples, the text in the 200 DPI graph is slightly more legible. Markers and lines are also more legible.

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 ODS Graphics 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.

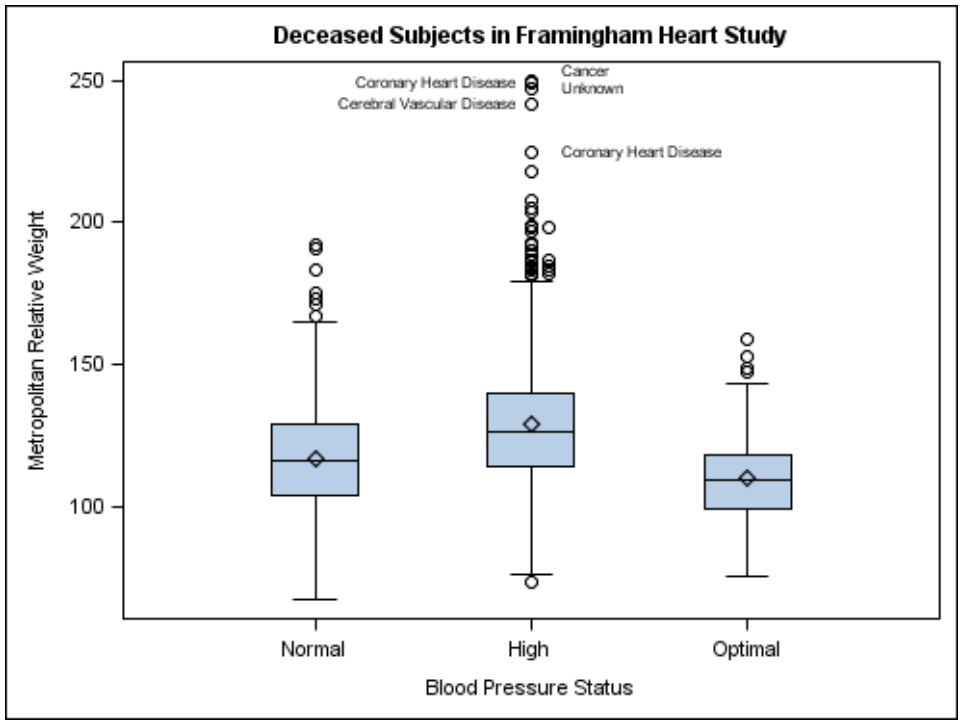
For example, suppose the following template is used to create box plots in a graph and you want to indicate that the labeled outliers are far outliers (more than 3 IQR above 75th percentile).

```
proc template;
  define statgraph boxplot;
    begingraph;
      entrytitle "Deceased Subjects in Framingham Heart Study";
      layout overlay;
        boxplot y=mrw x=bp_status / datalabel=deathcause
          labelfar=true;
      endlayout;
    endgraph;
  end;
run;
```

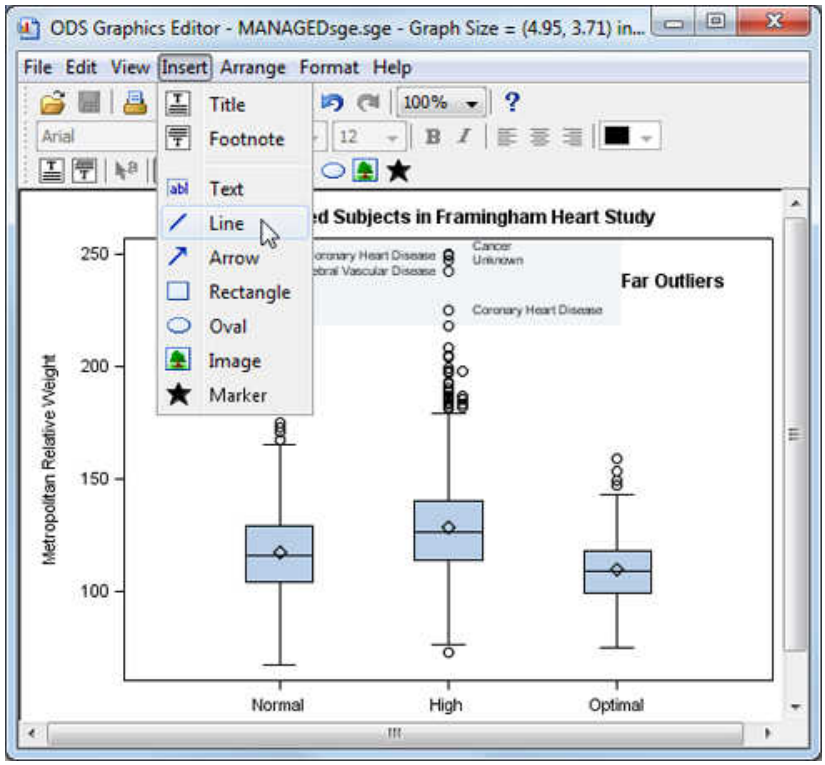
To create ODS Graphics output that can be edited, you must specify the SGE=ON option in the ODS LISTING destination statement before creating the graph:

```
/* Specify a path for the ODS output */
filename odsout "output-path";
ods _all_ close;
ods listing sge=on gpath=odsout;

proc sgrender data=sashelp.heart template=boxplot;
  where status="Dead";
run;
ods listing close;
ods html; /* Not required in SAS Studio */
```

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 That You Can Import into Microsoft Office Applications

The default height for a graph is 480 pixels. At a 100 dots 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 standalone image that is sized appropriately and has high resolution, say 200 DPI or 300 DPI.

```
ods graphics / reset width=5in imagename="fitplot" outputfmt=png
  antialias=on;
ods _all_ close;
ods listing gpath="output-path" image_dpi=200 style=analysis;

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

This code produces a 5 inch, 200 DPI image **.\fitplot.png**, which can be inserted into Microsoft Office 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 "Using Anti-Aliasing" on page 559 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, re-create the graph with different font sizes using the techniques discussed in "Scaling Graphs" on page 582.

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.

PART 9

Graph Templates

29

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 do the following:

- specify the input template
- specify the input data set
- associate a label with one or more input variables by using a LABEL statement
- associate a format with one or more input variables by 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.

Example Code 29.1 *Simple GTL Template*

```
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;
```

Note the following:

- 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 columns 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 columns the template uses, you can restrict the input columns 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 data set 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.

The following example executes template MYGRAPHS.SCATTER, which is defined in Example Code 29.1 on page 596. The first PROC SGRENDER uses a WHERE statement to select only female observations for the graph. The second PROC SGRENDER uses the OBS= input data set 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 columns that are in the data set Sashelp.Class. The transformations are stored in a temporary data set named Class, which is then used in the SGRENDER statement 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 the SGRENDER procedure:

```
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 Dynamic Variables and Macro Variables

A useful technique for generalizing templates is to define dynamic variables 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:

Example Code 29.2 *REGFIT Template*

```
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;
```

Note the following:

- The DYNAMIC statement declares dynamic variables XVAR and YVAR. On the statements that later execute this template, you must initialize these dynamic variables by assigning them to variables from the input data source so that they have values at run time.
- The ENTRYTITLE statement concatenates dynamic variables XVAR and YVAR into a string that will be displayed as the graph title. At run time, the dynamic variables are replaced by the names of the variables that are assigned to the dynamic variables when they are initialized.
- The SCATTERPLOT and REGRESSIONPLOT statements each reference the dynamic variables on their X= and Y= arguments. At run time 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 run time, 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 run-time value of SYSDATE will be displayed as a right-justified footnote.

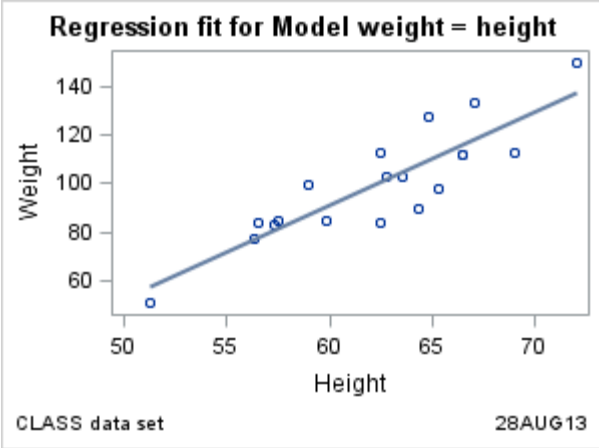
As with all GTL templates, the MYGRAPHS.REGFIT template can be executed with either a PROC SGRENDER statement or a DATA step. Either way, any dynamic variables and new macro variables that are declared in the template must be initialized to provide run-time values for them. The following example executes the template with PROC SGRENDER:

```
%let study=CLASS dataset;
proc sgrender data=sashelp.class template=mygraphs.regfit;
  dynamic xvar="height" yvar="weight";
run;
```

Note the following:

- 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.

Here is the output.



The DATA step uses the DYNAMIC= suboption of the ODS= option to initialize dynamic variables. 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 DATA step executes the MYGRAPHS.REGFIT template.

```
data _null_;
  if _n_=1 then call symput("study","CLASS data set");
  set sashelp.class;
  file print ods=( template="mygraphs.regfit"
                   dynamic=( xvar="height" yvar="weight" ) );
  put _ods_;
run;
```

Note the following:

- The CALL SYMPUT routine initializes the macro variable STUDY with the string value "CLASS data set." 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 30, “Using Dynamic Variables and Macro Variables in Your Templates,” on page 605.

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. (See Example Code 29.1 on page 596.) 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:

```
/* Create the graph template */
proc template;
  define statgraph mygraphs.scatter;
    begingraph;
      entrytitle "Height and Weight by Sex";
      layout overlay;
        scatterplot x=height y=weight /
          group=sex name="scatter" datalabel=name;
        discretelegend "scatter";
      endlayout;
    endgraph;
  end;
run;

/* 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;
```

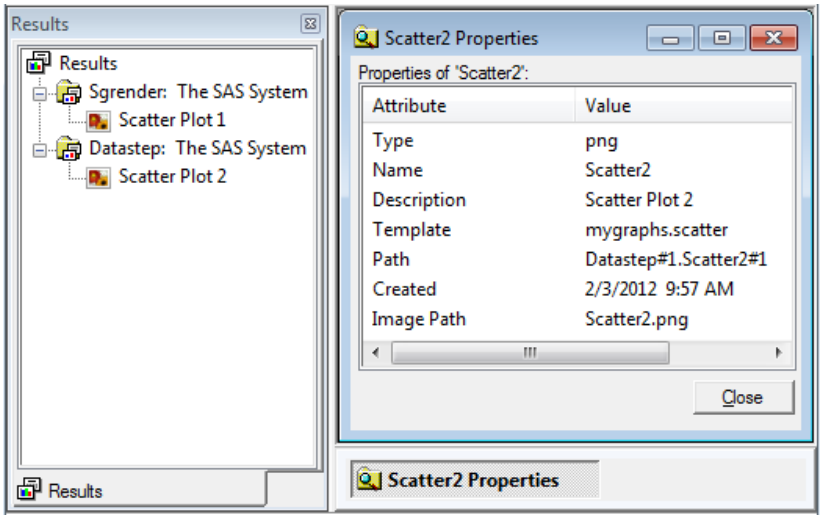
Viewing the Data Object Name and Label in the SAS Windowing Environment

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.

In the SAS windowing environment, object names and labels appear in the Results window. To view them, do the following:

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 601. 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*.



Note: The ODS data object Properties window is not available in SAS Studio.

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 and then renders the MYGRAPHS.REGFIT template. (See Example Code 29.2 on page 598).

```
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;
```

30

Using Dynamic Variables and Macro Variables in Your Templates

Introduction to Dynamic Variables 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.

Declaring Dynamic Variables 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 variable or macro variable.

Referencing Dynamic Variables and Macro Variables

After dynamic variables and macro variables are declared, you can make one or more template references to them by simply using the name of the dynamic variable 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;
```

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). Dynamic variables and run-time macro variable references cannot be used in place of punctuation that is part of the syntax (parentheses, semicolons, and so on). In most contexts, you can use a dynamic variable reference or a macro variable reference as an option value. However, there are other contexts in which a dynamic variable or run-time macro variable reference cannot be used. The following table provides general information about some of the common contexts in which dynamic variables and run-time macro variable references are not supported.

Table 30.1 Common Contexts in Which Dynamic Variable References and Macro Variable References Are Not Supported

Context	Dynamic Variable Reference and Macro Variable Reference Support	Example Variable Definition and Usage
A quoted reference name specified in an option such as NAME=, CLI=, or CLM=, or one or more quoted reference names required by statements such as DISCRETELEGEND, MERGEDLEGEND, AXISLEGEND, and MODELBAND	Not supported.	

A list of variable names in options such as CLASSVARS= in a LAYOUT DATALATTICE statement, CATEGORY= in a MOSAICPLOTPARM statement, and so on	Partially supported. A dynamic or MVAR variable can be used to specify a variable in the list. A dynamic or MVAR variable cannot be used to specify the entire list.	Dynamic variable definition:
		`dynamic VAR1="type" VAR2="origin";`
		Example usage:
		`mosaicPlotParm category=(VAR1 VAR2)`
		`count=count / name="mosaic"`
		`colorresponse=avgMpg;`

A free-form *name=value* pair list, where *name* is an option name, keyword, or role name	Partially supported. A dynamic or MVAR variable can be used to specify a *value* in the list, in most cases. A dynamic or MVAR variable cannot be used to replace a *name* or to specify the entire list of *name=value* pairs, in most cases.[1]	Dynamic variable definition:
		`dynamic MTICKS="true" MTICKCNT=5`
		`ROLE1="age" TIPS="X Y ROLE1"`
		`MARKERSIZE="12pt";`
		Example usage:
		`layout overlay /`
		`yaxisopts=(linearopts=(`
		`minorticks=MTICKS`
		`minortickcount=MTICKCNT));`
		`scatterplot x=height y=weight /`
		`rolename=(tip1=ROLE1) tip=TIPS`
		`markerattrs=(size=MARKERSIZE);`

Simple scalar lists	Partially supported. A dynamic or MVAR variable can be used to replace the entire list. A dynamic, MVAR, or NMVAR variable cannot be used to replace one or more values in the list, in most cases.[1]	Dynamic variable definition:
		`dynamic TICKVALUES="10 20 30 40"`
		`TICKDISPLAY="A B C D";`
		Example usage:
		`layout overlay / xaxisopts=(type=discrete`
		`discreteopts=(tickvaluelist=TICKVALUES`
		`tickdisplaylist=TICKDISPLAY));`

One or more *start–end* range value pairs	Not supported in the RANGE statement in a RANGEATTRMAP block.
	Partially supported in linear axis option INCLUDERANGES=. A
		Dynamic variable definition:
		`dynamic RANGES="50 - 52 54 - 73";`
		Example usage:
		`layout overlay / xaxisopts=(linearopts=`
		`(includeranges=RANGES));`

Context	Dynamic Variable Reference and Macro Variable Reference Support	Example Variable Definition and Usage
	dynamic or MVAR variable can be used to specify the entire list of *start–end* range value pairs. A dynamic or NMVAR variable cannot be used to specify a value in the range list.	

Special text-item command {*text-command*}	Partially supported. A dynamic or MVAR variable can be used to replace a *text-command* value in text statements such as ENTRY, ENTRYTITLE, and FOOTNOTE. A dynamic or MVAR variable cannot be used to replace a SUP, SUB, or UNICODE command keyword.	Dynamic variable definition:
		`dynamic VAR="'03b1'"; VARVAL=0.05;`
		Example usage:
		`entry {unicode VAR} " = " VARVAL;`

1 Where applicable, exceptions are noted in the option descriptions in *SAS Graph Template Language: Reference*.

Initializing Dynamic Variables and Macro Variables

The main difference between dynamic variables and macro variables is that they are initialized differently.

For dynamic variables, use the DYNAMIC statement with PROC SGRENDER as shown in the following example.

```
proc sgrender data=financial template=timeseries;
  dynamic yvar="inflation" ylabel="Inflation Rate"
     ylabelrotation=90;
run;
```

For dynamic variables that resolve to column names or strings, enclose the value in quotation marks. For dynamic variables that resolve to numeric values, specify the value without quotation marks.

For macro variables, use the current symbol table (local or global) to look up the macro variable values at run time as shown in the following example.

```
%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 dynamic variables 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 dynamic variables or macro variables. Conditional logic is discussed in Chapter 31, “Using Conditional Logic and Expressions in Your Templates,” on page 619.

If a dynamic variable 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 variable 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 variable 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.

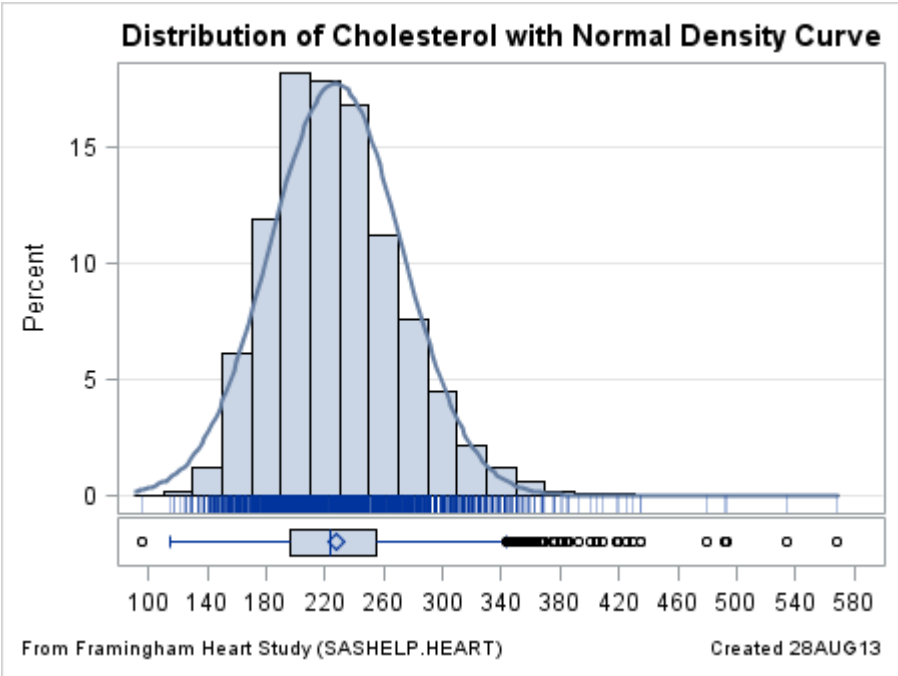
The following example shows how to create a generalized template that can be used to show the distribution of any numeric variable. The dynamic variable named VAR must be set, but the other dynamic variables are optional: BINS (sets the number of histogram bins) and FOOTNOTE. In the following example, 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;
```

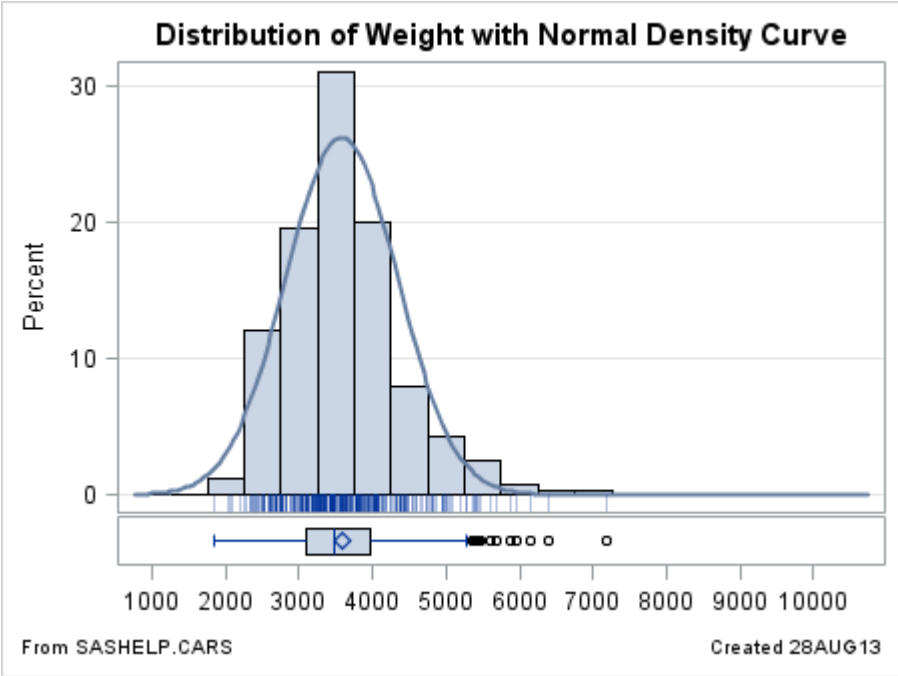
Here is the output.



In this case, the template option `bins=BINS` drops out because the BINS dynamic variable has not been initialized. This following execution of the template assigns values to each of the dynamic variables 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;
```

Here is the output.



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 run time, using dynamic variables and macro variables.

```
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;
```

For more information about coding insets in graphs, see Chapter 21, “Adding Insets to Your Graph,” on page 383.

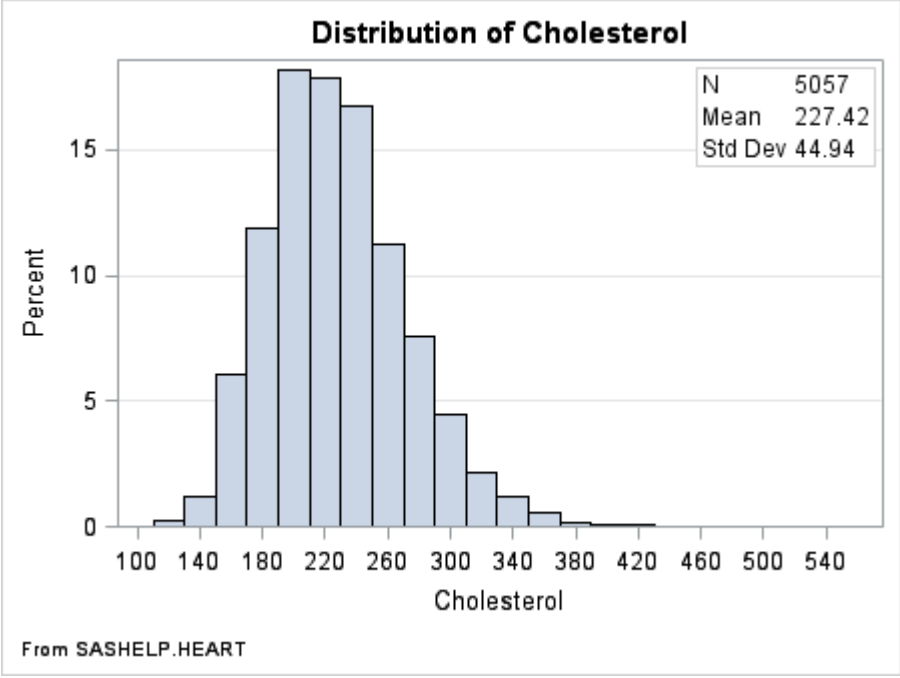
We will now define a macro that can pass values to this template. For a given numeric variable, the macro computes the number of observations, the mean, and the standard deviation, storing these statistics in macro variables N, MEAN, and STD. The macro variables are available to the SGRENDER step when the macro executes. Here is the definition for the macro, which is named HIST.

```
%macro hist(dsn,numvar,footnote);
  /* these macro variables are declared in the template */
```

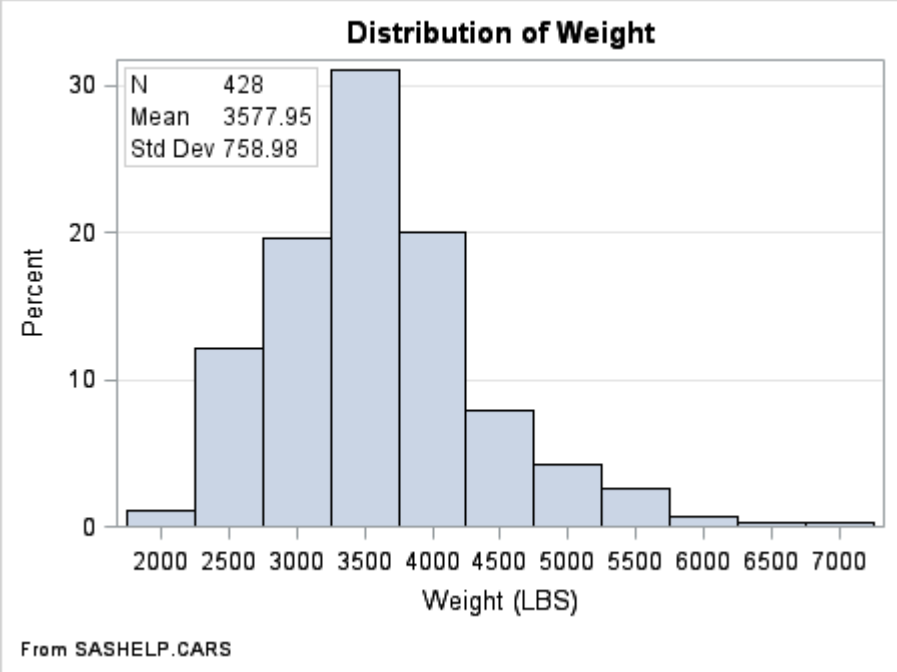
```
  %local N MEAN STD;
  proc sql noprint;
    select  put(n(&numvar),12. -L),
            put(mean(&numvar),12.2 -L),
            put(std(&numvar),12.2 -L) into :N, :MEAN, :STD
    from &dsn;
  quit;

  /* remove trailing blanks */
  %let N=&N; %let MEAN=&MEAN; %let STD=&STD;

  ods graphics / reset width=450px;
  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 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 31, “Using Conditional Logic and Expressions in Your Templates,” on page 619.

GTL supports user-defined computed expressions within compiled templates. This means that the inset statistics could have been computed inside the template, eliminating the need to pass them in with dynamic variables or macro variables. An example of how to do this is also discussed in Chapter 31, “Using Conditional Logic and Expressions in Your Templates,” on page 619.

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 639.

Special Dynamic Variables

Several special predefined dynamic variables are available that you can use with your templates. These special variables are listed in Table 30.2 on page 614.

Table 30.2 *Special Dynamic Variables*

Dynamic Variable	Description
LIBNAME	This variable represents the name of the library that contains the data set.

Dynamic Variable	Description
MEMNAME	This variable represents the name of the library member that contains the data set.
BYFOOTNOTE[1]	This variable is set to 1 when you specify a BY statement and the ODS GRAPHICS BYLINE= option is set to FOOTNOTE. Otherwise, it is set to 0 or is NULL. For more information about the ODS GRAPHICS BYLINE= option, see "ODS GRAPHICS" in *SAS Graph Template Language: Reference*.
BYLINE	This variable represents the complete BY line when you specify a BY statement.
BYTITLE[1]	This variable is set to 1 when you specify a BY statement and the ODS GRAPHICS BYLINE= option is set to TITLE. Otherwise, it is set to 0 or is NULL.
BYVAR	This variable represents the name of the first BY variable when you specify a BY statement.
BYVAR*n*	This variable represents the name of the nth BY variable when you specify a BY statement with multiple variables.
BYVAL	This variable represents the first BY value when you specify a BY statement.
BYVAL*n*	This variable represents the value of the nth BY variable when you specify a BY statement with multiple variables.

1 This variable is set automatically only for analytical procedures that support ODS Graphics. For all other procedures, this variable is not set automatically (NULL).

To use a special variable in your template, you must define it in the DYNAMIC statement in your template. However, you do not have to define and initialize the special variable in your SGRENDER procedure statement. The special variables are defined and initialized automatically at run time for the SGRENDER procedure.

Here is an example that shows you how special variables can be used in a template. This example generates a graph of the monthly stock closing price for each year for the years 2000 through 2005. It uses a BY statement in the SGRENDER statement to generate the graphs by year and provides the following options for displaying the BY value in each graph:

- Display the BY line in the graph title.
- Display the BY line in a footnote.
- Display the BY value in the graph title.

The _BYLINE_ variable is used to include the BY line in the graph title or in a footnote. The _BYTITLE_ and _BYFOOTNOTE_ variables are used with the BYLINE= ODS GRAPHICS option to select between displaying the BY line in the title or footnote. When neither _BYTITLE_ nor _BYFOOTNOTE_ is set, the _BYVAL_ variable is used to include the BY value in the graph title as the default behavior. Finally, before the graph is rendered, system option NOBYLINE is set to

suppress the default BY line that is normally displayed when the BY statement is used.

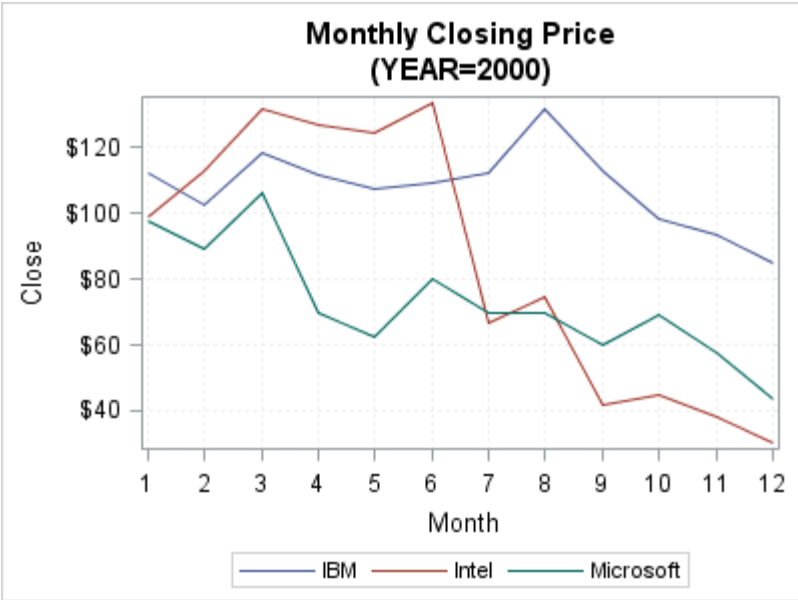
Here is the code for this example.

```
/* Add a YEAR column to the data set. */
data stocks;
  set sashelp.stocks;
  year=year(date);
  label year="YEAR";
run;

/* Sort the data by YEAR. */
proc sort data=stocks;
  by year;
run;

/* Create the template for the graph. */
proc template;
  define statgraph seriesgroup;
    begingraph;
      /* Define the special variables that will be used. */
      dynamic _BYVAL_ _BYLINE_ _BYTITLE_ _BYFOOTNOTE_;

      /* If BYTITLE is set, put the BY line in the graph title. */
      if (_BYTITLE_)
        entrytitle "Monthly Closing Price";
        entrytitle "(" _BYLINE_ ")";
      else
        /* If BYFOOTNOTE is set, put the BY line in a footnote. */
        if (_BYFOOTNOTE_)
          entrytitle "Monthly Closing Price";
          entryfootnote halign=right "(" _BYLINE_ ")";
        else
          /* Otherwise, include _BYVAL_ in the title. */
          entrytitle "Monthly Closing Price In " _BYVAL_;
        endif;
      endif;
      layout overlay / xaxisopts=(label="Month" type=discrete
        griddisplay=on gridattrs=(pattern=dot))
        yaxisopts=(griddisplay=on gridattrs=(pattern=dot));
        seriesplot x=eval(month(date)) y=close /
          group=stock name="s";
        discretelegend "s";
      endlayout;
    endgraph;
  end;
run;

/* Disable the default BY line. */
options nobyline;

/* Put the BY line in the graph title */
ods graphics / byline=title;

/* Generate the graph. */
proc sgrender data=stocks template=seriesgroup;
```

```
  by year;
  where year between 2000 and 2005;
run;
```

Notice that _BYVAL_, _BYLINE_, _BYTITLE_, and _BYFOOTNOTE_ are included in the DYNAMIC statement of the template definition but are not declared or initialized in the SGRENDER statement. Conditional logic is used to test whether the _BYTITLE_ or _BYFOOTNOTE_ variable is set and to place the BY line accordingly. If neither variable is set, the conditional logic includes the _BYVAL_ value in the graph title. The BYLINE=TITLE option in the ODS GRAPHICS statement specifies that the BY line be displayed in the graph title.

The following figure shows the first graph. As shown in the figure, the first BY line, YEAR=2000, is included in the graph title.



To display the BY line in a footnote, specify BYLINE=FOOTNOTE in the ODS GRAPHICS statement. To display the BY value in the title instead, use the BYLINE=NOBYLINE option in the ODS GRAPHICS statement or remove the ODS GRAPHICS statement.

31

Using Conditional Logic and Expressions in Your Templates

Constructs Available for Run-Time Programming

GTL has several constructs that can take advantage of the following run-time programming features:

- dynamic variables and macro variables
- expressions
- conditional processing

This chapter discusses expressions and conditional processing. Dynamic variables and macro variables are discussed in Chapter 30, “Using Dynamic Variables and Macro Variables in Your Templates,” on page 605.

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")
```

Conditional Logic

GTL supports conditional logic that enables you to include or exclude one or more GTL statements at run time:

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*.

Note: Dynamic variables that are initialized to an ODS-recognized value, such as YES, NO, TRUE, or FALSE, do not work as expected when used as an expression in an IF-THEN statement. In that case, the expression is treated as a string, which always evaluates to a positive integer value (FALSE).

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 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 dynamic variables. 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 dynamic variables that are uninitialized, simply "drop out" at run time 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;
```

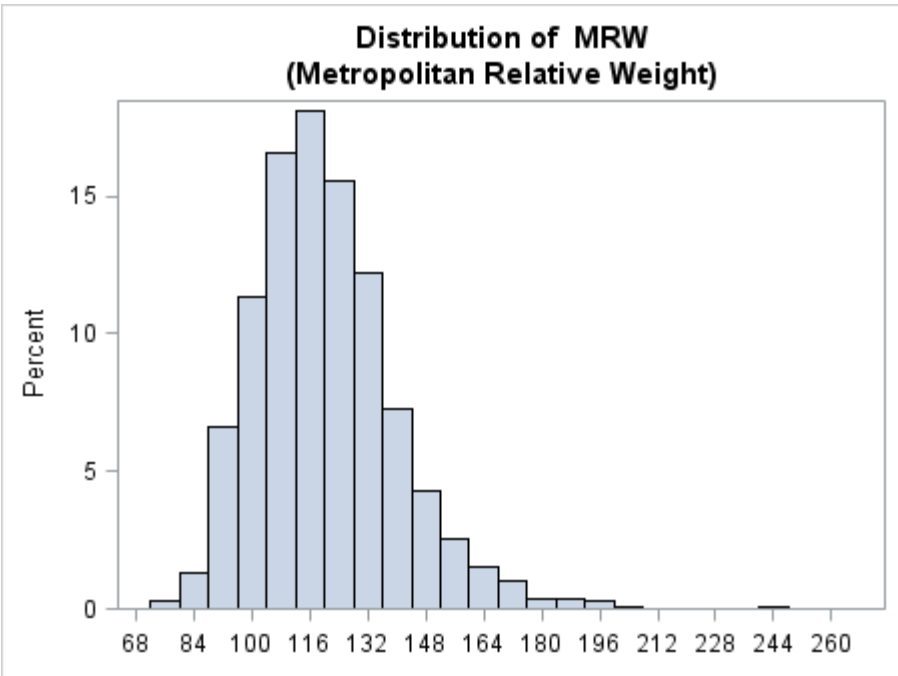
- The DYNAMIC statement identifies the dynamic variables.
- The first IF block specifies an ENTRYTITLE statement that is conditionally executed if the column name differs from the column label.
- The next two IF blocks evaluate the value of the dynamic variable CURVE. If CURVE is not used, the code in 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, you can execute the template with various combinations of dynamic values.

In this first execution, the NUMVAR dynamic variable 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 variable is not initialized, so the template does not generate a density plot.

```
proc sgrender data=sashelp.heart template=conditional;
  dynamic numvar="mrw";
run;
```

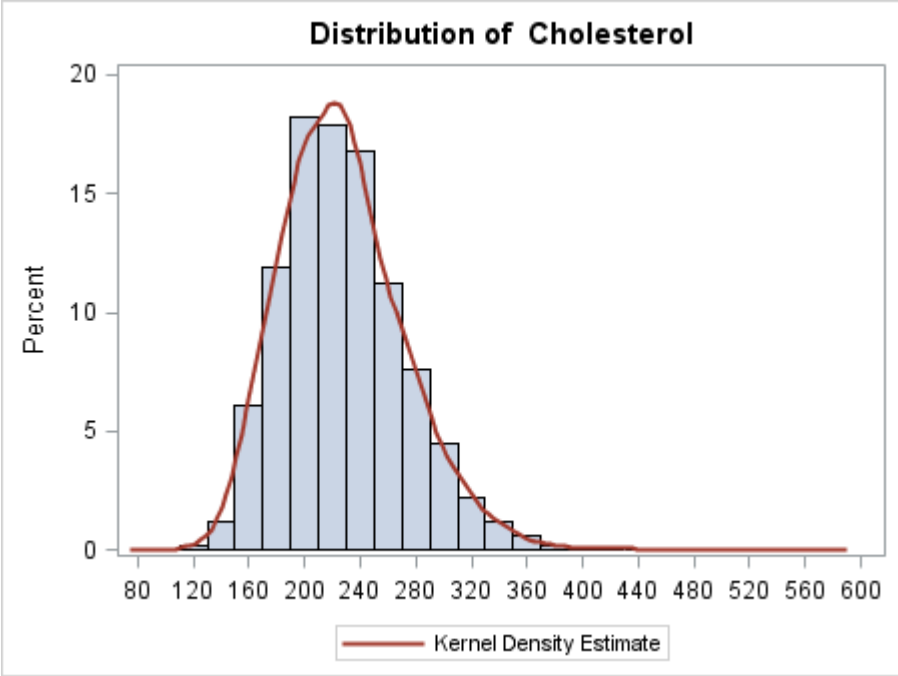
Here is the output.



In this next execution of the template, the NUMVAR dynamic variable 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 variable 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;
```

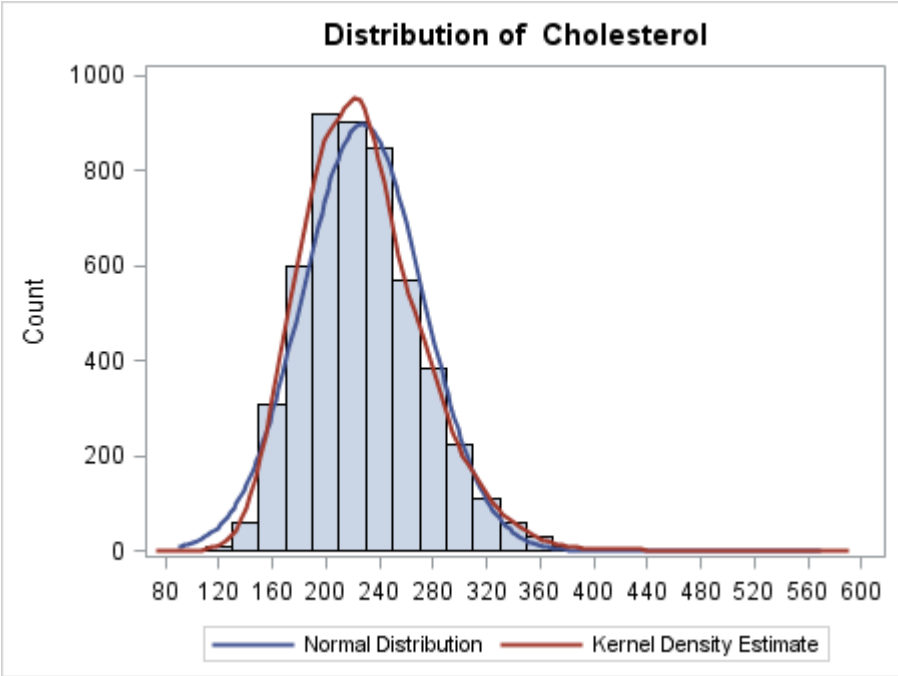
Here is the output.



In this final execution of the template, the CURVE dynamic variable 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;
```

Here is the output.



The value of the SCALE dynamic does not need to be verified. If it is not one of COUNT, DENSITY, PERCENT, or PROPORTION (not case sensitive), the default scale is used with no warning or error.

32

Using Functions in Your Templates

Overview

GTL supports a large number of functions, including:

- SAS functions that can be used in the context of a WHERE expression
- functions that are defined only in GTL
- summary statistic functions

SAS Functions

SAS Functions That Can Be Used in a GTL Template

Most of the SAS functions that are available in WHERE expressions can be used in a GTL template. These SAS functions include:

- character-handling functions
- date and time functions
- mathematical and statistical functions

Not all SAS functions are available in WHERE expressions. Call routines and other DATA-step-only functions (for example, LAG, VNAME, OPEN) are some examples of functions that cannot be used. Not all functions that are available in WHERE expressions are supported in GTL templates in all cases. The following form of the PUT function is an example:

```
markercharacter = eval(put(amount, dollar7.2 -L))
```

This form results in an error when the template is compiled. However, the following form is supported.

```
markercharacter = eval(put(amount, dollar7.2))
```

If you want to justify a string that is generated by the PUT function, use the LEFT or RIGHT function with the PUT function as shown in the following example:

```
markercharacter = eval(left(put(amount, dollar7.2)))
```

Functions that accept null parameter values also might not be supported when you specify a null parameter value.

For more information about SAS functions, see "Dictionary of Functions and CALL Routines" in *SAS Functions and CALL Routines: Reference*.

SAS Functions That Can Be Used to Create Flexible Templates

The following table shows some of the SAS functions can be used to increase the flexibility of your template code.

SAS Function Name	Description
IFC(*logical-expression*, "*true-value*", "*false-*	Returns the character value *true-value* if *logical-expression* resolves to TRUE, *false-value* if it resolves to FALSE, or

SAS Function Name	Description
value" <,"*missing-value*">)	*missing-value* if it resolves to a missing value. The TRUE, FALSE, and MISSING values must be enclosed in quotation marks.
IFN(*logical-expression*, *true-value*, *false-value* <,*missing-value*>)	Returns the numeric value *true-value* if *logical-expression* resolves to TRUE, *false-value* if it resolves to FALSE, or *missing-value* if it resolves to a missing value.

Examples of Using the IFC and IFN SAS Functions

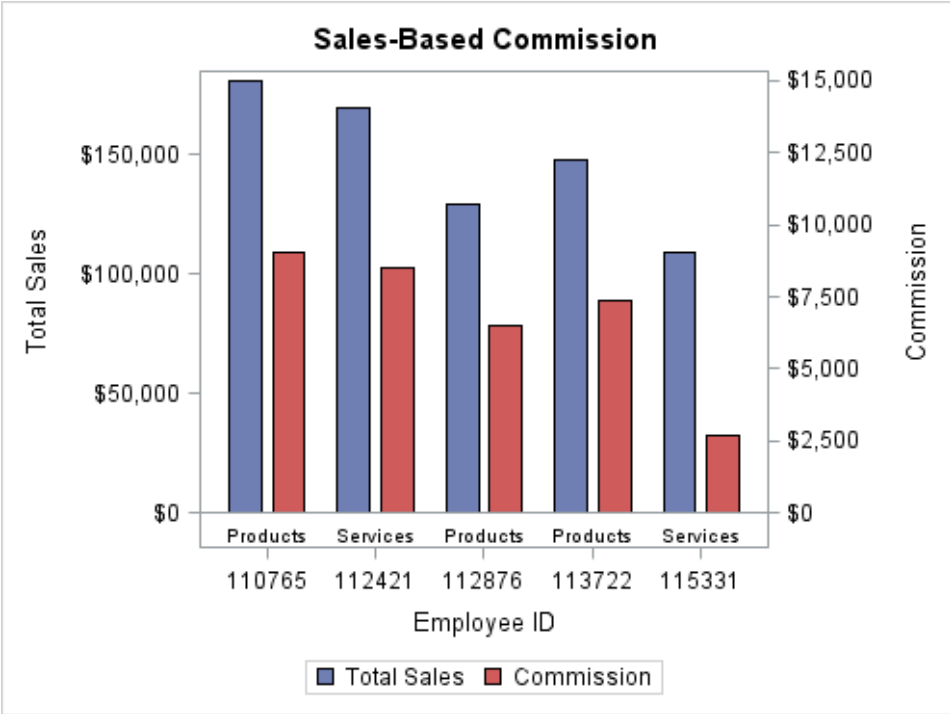
The IFC and IFN functions return one of two character or numeric values based on whether a conditional expression resolves to TRUE or FALSE. They can also return an optional third value if the conditional expression resolves to a missing value. These functions enable you to specify a value based on a conditional expression, effectively creating a new data column. In some cases, these functions can be used in place of IF-THEN-ELSE statements in your template code. As with other functions, you must enclose the IFC and IFN functions in the EVAL function.

Here is an example that uses both the IFN and IFC functions for creating a sales-based commission chart for employees in a sales group. Each employee in the group works in one of two sales units: Products and Services. The data for this example includes the employee ID, total sales, and sales unit code for each member of the sales group. Here is the data.

```
data sales;
  input empID totalSales salesUnit $18;
  format totalSales dollar9.;
datalines;
112876 129489.44 P
112421 169842.97 S
115331 108763.51 S
110765 181009.22 P
113722 147688.78 P
;
```

The TotalSales column contains the total sales for each employee. The SalesUnit column contains a code that identifies the sales unit in which each employee works. The codes are P for the Products unit and S for the Services unit.

Here is the output for this example.



The two bar charts show the total sales and earned commission for each employee. The IFN function is used to compute commission for each employee based on his or her total sales. Employees that achieved a sales total of $120,000 or more earn a commission of 5% of their total sales. All other employees earn a commission of 2.5% of their total sales.

An axis table along the X axis shows the sales unit for each employee. The IFC function is used to convert the P and S SalesUnit codes into more descriptive values in the axis table. Because only two sales unit codes are used in this case, the IFC function can be used for this purpose. This eliminates the need to add a new column to the data in a DATA step or to create and apply a custom format to the SalesUnit column.

Here is the SAS code that defines the template and generates the graph.

```
proc template;
  define statgraph commission;
    begingraph;
      entrytitle "Sales-Based Commission";
      layout overlay /
        xaxisopts=(label="Employee ID")
        yaxisopts=(label="Total Sales")
        y2axisopts=(label="Commission"
          linearopts=(viewmax=15000 tickvalueformat=dollar9.));
        /* Generate the sales bar chart. */
        barchart category=empID response=totalSales /
          name="Sales" legendlabel="Total Sales" barwidth=0.3
          discreteoffset=-0.2 fillattrs=graphData1;

        /* Generate the commission bar chart. */
        barchart category=empID
          /* Use IFN to compute the commission. */
          response=eval(ifn(totalSales >= 120000,
            totalSales * 0.05,      /* 5% if TRUE */
```

```
            totalSales * 0.025)) /   /* 2.5% if FALSE */
          name="Commission"  legendlabel="Commission"
          barwidth=0.3 yaxis=y2 discreteoffset=0.2
          fillattrs=graphData2;

        /* Add an axis table that shows the sales unit for each
          employee. */
        innermargin / align=bottom;
          axistable x=empID
            /* Use IFC to convert the codes to meaningful
               values. */
            value=eval(ifc(salesUnit = 'P',"Products",
              "Services")) / display=(values);
        endinnermargin;
        discretelegend "Sales" "Commission";
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sales template=commission;
run;
```

Functions Defined Only in GTL

GTL Functions Used with the EVAL Function

The following table shows some functions that are used only in GTL. As with other functions, these must be enclosed within an EVAL. In all these functions, *column* can be either the name of a column in the input data set or a dynamic / macro variable that resolves to such a column.

Function Name	Description
COLC("*string-1*", "*string-1*"<, "*string-n*" … >)	Converts a list of comma-separated string values into a temporary character column. Starting with SAS 9.4M2, you can use this function to specify values in options that accept a character column.
COLN(*n-1*, *n-1*<, *n-N*… >)	Converts a list of comma-separated numeric values into a temporary numeric column. You can use this function to specify values in options that accept a numeric column.
COLNAME(*column*)	Returns the case-sensitive name of the *column*.

Function Name	Description
COLLABEL(*column*)	Returns the case-sensitive label of the *column*. If no label is defined for the *column*, then the case-sensitive name of the *column* is returned.
EXISTS(*item*)	Returns 1 if specified *item* exists, 0 otherwise. If *item* is a column, then it tests for the presence of the column in the input data set. If *item* is a dynamic / macro variable, then it tests whether there has been a run-time initialization of the variable.
EXPAND(*numeric-column*, *freq-column*)	Creates a new column as (*numeric-column* * *frequency-column*) .
ASORT(*column*, RETAIN=ALL)	Sorts all columns of the data object by the values of *column* in ascending order. SORT is an alias for ASORT. **Warning:** if the RETAIN=ALL argument is not included, *column* alone is sorted, not the other columns, causing rowwise information to be lost. **Limitation:** only one sort operation (whether an ASORT() or DSORT() function) can be used within a single template definition.
DSORT(*column*, RETAIN=ALL)	Sorts all columns of the data object by the values of *column* in descending order. **Warning:** if the RETAIN=ALL argument is not included, *column* alone is sorted, not the other columns, causing rowwise information to be lost. **Limitation:** only one sort operation (whether an ASORT() or DSORT() function) can be used within a single template definition.
NUMERATE(*column*)	Returns a *column* that contains the ordinal position of each observation in the input data set (similar to an Obs column).

Examples

```
/* arrange bars in descending order of response values */
barchartparm category=region response=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,
```

```
   which is passed by a dynamic */
entrytitle "Distribution for " eval(colname(DYNVAR));
```

Using the TYPEOF SAS Function

The TYPEOF function returns the type of a specified column at run time.

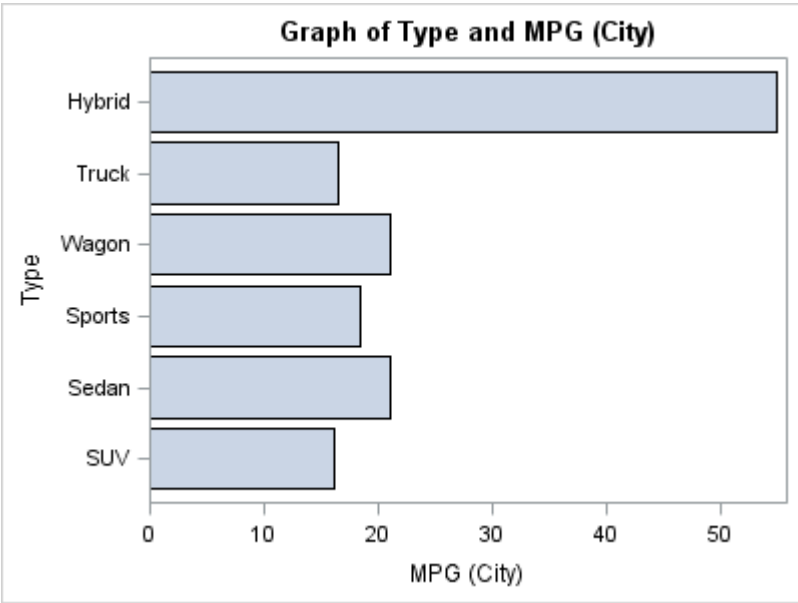
TYPEOF(*column*)

This function returns the character ‘C’ if the specified column is a character column or ‘N’ if it is a numeric column.

You can use the TYPEOF function to take specific actions in your template at run time based on the input data type. Here is an example that creates a graph of two columns and uses the TYPEOF function to select a graph type that is appropriate for the column types. The result returned by the TYPEOF function determines the graph type as follows:

- If both columns are numeric, then it creates a scatter plot.
- If the X column is character and the Y column is numeric, then it creates a vertical bar chart.
- If the X column is numeric and the Y column is character, then it swaps the category and response columns in the BARCHART statement and orients the chart horizontally.

Here is the output for the third case, a numeric X column and a character Y column.



Here is the SAS code.

```
/* Define the graph template. */
proc template;
  define statgraph plot;
    dynamic cat resp; /* Category and response columns. */
    begingraph;
      entrytitle "Graph of " eval(collabel(resp)) " and "
```

```
        eval(collabel(cat));
      layout overlay;
        /* If cat and resp are numeric, then generate a scatter plot.
          Otherwise, generate a bar chart.  */
        if (typeof(cat) = "N" and typeof(resp) = "N")
          scatterplot x=cat y=resp;
        else
          /* If cat is a character column, then generate a vertical bar
            chart. Otherwise, generate a horizontal bar chart. */
          if (typeof(cat) = "C")
            barchart category=cat response=resp / stat=mean;
          else
            barchart category=resp response=cat /
              stat=mean orient=horizontal;
          endif;
        endif;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.cars template=plot;
  dynamic cat="MPG_CITY" resp="TYPE";
run;
```

Note: See "Functions Defined Only in GTL" on page 631 for information about the COLLABEL function.

GTL Summary Statistic Functions

Commonly Used Summary Statistic Functions

Several GTL summary statistic functions are available that return a numeric constant, based on a summary operation on a numeric column. The results are the same as if the corresponding statistics were requested with PROC SUMMARY. These functions take a single argument that resolves to the name of a numeric column. Here is an example.

```
number = EVAL(function-name(numeric-column))
```

The GTL summary statistic functions take precedence over similar multi-argument DATA step functions. The following table lists the GTL summary statistic functions that you can use.

Function Name	Description
CSS	Corrected sum of squares

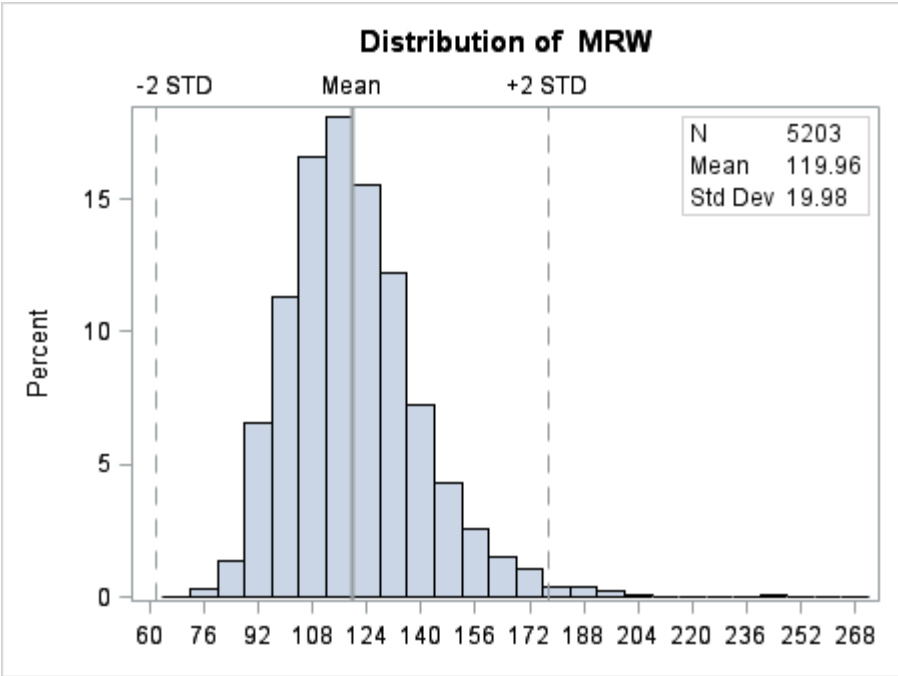
Function Name	Description
CV	Coefficient of variation
KURTOSIS	Kurtosis
LCLM	One-sided confidence limit below the mean
MAX	Largest (maximum) value
MEAN	Mean
MEDIAN	Median (50th percentile)
MIN	Smallest (minimum) value
N	Number of nonmissing values
NMISS	Number of missing values
P1	1st percentile
P5	5th 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

Function Name	Description
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
VAR	Variance

Example

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 run time.

Here is the graph for this example.



Here is the SAS code.

```
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;
```

33

Sharing Your Custom Templates

Creating Shared Templates

When creating templates (especially with dynamic variables 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)
```

34

Modifying Predefined Templates

Predefined Templates for SAS Analytical Procedures

The graphs that are produced by the SAS analytical procedures are created from precompiled STATGRAPH templates that are 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 subfolder of the Sashelp.Tmplmst item store. All of the template subfolder names have the prefix Tmpl. The template subfolder in which a specific predefined template is stored depends on the procedure.

To locate the predefined templates that a procedure uses, you can run the procedure with ODS TRACE ON in effect. In that case, trace information written to the SAS log indicates the path to the predefined templates that the procedure uses to generate output. For example, the following code shows how to display the predefined templates that the FREQ procedure uses.

```
ods trace on;
ods graphics on;
proc freq data=sashelp.cars order=freq;
   tables Type / plots=freqplot(type=dotplot);
   weight Weight;
   title "Vehicle Weight";
run;
ods graphics off;
ods trace off;
```

In this case, the FREQ procedure prints a table of frequency statistics for Type followed by a dot plot of the Type distribution. With ODS TRACE ON in effect, the following trace information about the output is written to the SAS log.

```
Output Added:
-------------
Name:       OneWayFreqs
Label:      One-Way Frequencies
Template:   Base.Freq.OneWayFreqs
Path:       Freq.Table1.OneWayFreqs
-------------

Output Added:
-------------
Name:       FreqPlot
Label:      Frequency Plot
Template:   Base.Freq.Graphics.OneWayFreqDotPlot
Path:       Freq.Table1.OneWayFreqPlots.FreqPlot
```

The second Output Added item in the trace output provides information about the dot plot. The Template line identifies the predefined template that is used to generate the dot plot. When searching the template catalogs, ODS searches a specific list of search paths. To see the search paths that are currently in effect, run the following command.

```
ods path show;
```

```
Current ODS PATH list is:

1. SASUSER.TEMPLAT(UPDATE)
2. SASHELP.TMPLMST(READ)
```

As indicated in the output, in this case, ODS searches the Sasuser.Templat catalog first, and then searches the Sashelp.Tmplmst catalog next. Once you have determined the path to the predefined template, you can then use the TEMPLATE procedure SOURCE statement to display the contents of the template.

```
proc template;
  source Base.Freq.Graphics.OneWayFreqDotPlot;
run;
```

When the TEMPLATE procedure is executed, ODS searches the template catalogs in the order of its search paths in order to locate the template. The first instance found is used. Once the template is located, the TEMPLATE procedure writes the template contents to the SAS log. Following the template contents in the SAS log is a note that indicates where the template was found.

```
NOTE: Path 'Base.Freq.Graphics.OneWayFreqDotPlot' is in: SASHELP.TMPLBASE
  (via SASHELP.TMPLMST).
```

As indicated in the note, the OneWayFreqDotPlot predefined template was found in the Sashelp.Tmplbase catalog.

Note: In the SAS Windowing Environment, you can also use the Templates window to browse the template catalogs. To open the Templates window, type `ODSTEMPLATES` in the command line, and then press Enter.

For more information about the ODS Graphics predefined templates, see *SAS/STAT User's Guide*.

Modify a Predefined Template

If you want to make persistent changes to the SAS statistical procedures automatic graphics, you can do so by editing and recompiling their predefined graph templates.

To modify a predefined template:

1. Determine the name of the template that you want to modify. For more information, see “Predefined Templates for SAS Analytical Procedures” on page 641.
2. Edit the template code as needed.
3. Submit the modified template code in order to compile it and save it in your Sasuser.Templat item store.
4. Verify that the Sasuser.Templat item store appears first in the ODS search path list.

To modify a template, you must be familiar with the structure of the templates, which requires a thorough knowledge of GTL. If you are not familiar with GTL, use this guide and *SAS Graph Template Language: Reference* to gain a working knowledge of GTL before you proceed. You must also follow the recommended guidelines for modifying these templates. For more information about modifying the predefined templates, see *SAS/STAT User's Guide*.

PART 10

Appendixes

Appendix 1

Reserved Keywords and Unicode Values

Reserved Keywords and Unicode Values

Overview

The tables in this section show some of the reserved keywords and Unicode values that can be used with the UNICODE text command. For information about rendering Unicode characters, see “Managing the String on Text Statements” on page 324.

Note the following:

- Keywords and Unicode values are not case-sensitive: "03B1"x is the same code point as "03b1"x.
- The word `blank` is the keyword for a blank space.

Lowercase 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 lambda
mu	μ	03BC	lowercase mu
nu	ν	03BD	lowercase nu
xi	ξ	03BE	lowercase xi
omicron	ο	03BF	lowercase omicron
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

Keyword	Glyph	Unicode	Description
psi	ψ	03C8	lowercase psi
omega	ω	03C9	lowercase omega

Uppercase Greek Letters

Keyword	Glyph	Unicode	Description
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
sigma_u	Σ	03A3	uppercase sigma

Keyword	Glyph	Unicode	Description
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[1]
bar2	‾‾	033F	combining double overline[1]
tilde	˜	0303	combining tilde[1]
hat	ˆ	0302	combining circumflex accent[1]

1 This is an overstriking character that requires a Unicode font to render properly.

Appendix 2

Graph Style Elements Used by ODS Graphics

About the Graphical Style Elements

The style elements that are described in this appendix affect template-based graphics. These style elements can be specified by Graph Template Language appearance options or used in style templates. The graphical style elements fall into the following categories:

- general graph appearance
- graphical data representation (non-grouped data)
- graphical data representation (grouped data)
- display

The following sections list the graphical style elements that are in each of these categories. For additional information about ODS style elements, see *SAS Output Delivery System: User's Guide*

General Graph Appearance Style Elements

The following table lists the general graph appearance style elements.

Table A2.1 *Graph Style Elements: General Graph Appearance*

Style Element	Portion of Graph Affected	Recognized Attributes
GraphAnnoLine	Annotation lines and arrows, and outlines for closed annotation shapes such as circles and squares.	ContrastColor LineStyle LineThickness
GraphAnnoShape	Fill for closed annotation shapes such as circles and squares, and the default annotation marker in ODS Graphics Editor.	Color MarkerSize MarkerSymbol Transparency
GraphAnnoText	Annotation text	Color Font or *font-attributes*[1]
GraphAxisLines	X-, Y-, and Z-axis lines	ContrastColor LineStyle LineThickness TickDisplay
GraphBackground	Background of the graph	Color Transparency
GraphBorderLines	Border line around the graph and around the graph legend	ContrastColor LineStyle LineThickness
GraphDataText	Text font and color for point and line labels	Color Font or *font-attributes*[1]
GraphFootnoteText	Text font and color for footnote(s)	Color Font

Style Element	Portion of Graph Affected	Recognized Attributes
		or *font-attributes*[1]
GraphGridLines	Horizontal and vertical grid lines drawn at major tick marks	ContrastColor DisplayOpts LineStyle LineThickness Transparency
GraphHeaderBackground	Background color of the cell headers in a data-driven lattice or data-driven panel graph	Color Transparency
GraphLabelText	Text font and color for axis labels and legend titles	Color Font or *font-attributes*[1]
GraphLegendBackground	Background color of the legend	Color Transparency
GraphMinorGridLines	Appearance of the horizontal and vertical minor grid lines.	ContrastColor DisplayOpts LineStyle LineThickness
GraphOutlines	Outline properties for fill areas such as bars, pie slices, box plots, ellipses, and histograms	Color ContrastColor LineStyle LineThickness
GraphReference	Horizontal and vertical reference lines and drop lines	ContrastColor LineStyle LineThickness
GraphTitleText	Text font and color for title(s)	Color Font or *font-attributes*[1]
GraphUnicodeText	Text font for Unicode values	Color Font or*font-attributes*[1]
GraphValueText	Text font and color for axis tick values and legend values	Color Font

Style Element	Portion of Graph Affected	Recognized Attributes
		or *font-attributes*[1]
GraphWalls	Vertical wall(s) bounded by axes	Color ContrastColor FrameBorder LineStyle LineThickness

1 *Font-attributes* can be one or more of the following: FONTFAMILY=, FONTSIZE=, FONTSTYLE=, FONTWEIGHT=.

Graphical Data Representation Style Elements (Non-Grouped Data)

The following table lists the graphical data representation style elements that affect non-grouped data.

Table A2.2 *Style Elements Affecting Graphical Data Representation*

Style Element	Portion of Graph Affected	Recognized Attributes
GraphBoxMean	Marker for mean	ContrastColor MarkerSize MarkerSymbol
GraphBoxMedian	Line for median	ContrastColor LineStyle LineThickness
GraphBoxWhisker	Box whiskers and serifs	ConstrastColor LineStyle LineThickness
GraphConfidence	Primary confidence lines and bands, colors for bands and lines	ContrastColor Color LineStyle LineThickness MarkerSize MarkerSymbol

Style Element	Portion of Graph Affected	Recognized Attributes
		Transparency
GraphConfidence2	Secondary confidence lines and bands, color for bands, and contrast color for lines	ContrastColor Color LineStyle LineThickness MarkerSize MarkerSymbol Transparency
GraphConnectLine	Line for connecting boxes or bars	ContrastColor LineStyle LineThickness
GraphCutLine	Cutline attributes for a dendogram	Color LineStyle
GraphDataDefault	Primitives related to non-grouped data items, colors for filled areas, markers, and lines	Color ContrastColor EndColor LineStyle LineThickness NeutralColor MarkerSymbol MarkerSize StartColor
GraphError	Error line or error bar fill, ContrastColor for lines, Color for bar fill	CapStyle Color ContrastColor LineStyle Transparency
GraphFit	Primary fit lines such as a normal density curve	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol

Style Element	Portion of Graph Affected	Recognized Attributes
GraphFit2	Secondary fit lines such as a kernel density curve	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol
GraphFinal	Final data for the waterfall chart. Color applies to filled areas.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol TextColor
GraphInitial	Initial data for the waterfall chart. Color applies to filled areas.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol TextColor
GraphMissing	Properties for graph items representing missing values	Color ContrastColor FillPattern[1] LineStyle LineThickness MarkerSymbol MarkerSize Transparency
GraphOther	Other data for the graph. Color applies to filled areas.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol TextColor

Style Element	Portion of Graph Affected	Recognized Attributes
GraphOverflow	Overflow data for the graph. Color applies to filled areas. ContrastColor applies to markers and lines.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol TextColor
GraphOutlier	Outlier data for the graph	ContrastColor Color MarkerSize MarkerSymbol LineStyle LineThickness
GraphPrediction	Prediction lines	ContrastColor Color LineStyle LineThickness MarkerSize MarkerSymbol Transparency
GraphPredictionLimits	Fills for prediction limits	ContrastColor Color LineStyle LineThickness MarkerSize MarkerSymbol Transparency
GraphSelection	For interactive graphs, visual properties of selected item. Color for selected fill area, ContrastColor for selected marker or line.	ContrastColor Color LineStyle LineThickness MarkerSymbol MarkerSize
GraphUnderflow	Underflow data for the graph. Color applies to filled areas.	Color ContrastColor

Style Element	Portion of Graph Affected	Recognized Attributes
	ContrastColor applies to markers and lines.	LineStyle LineThickness MarkerSize MarkerSymbol TextColor
ThreeColorAltRamp	Line contours, markers, and data labels with segmented range color response	EndColor NeutralColor StartColor
ThreeColorRamp	Gradient contours, surfaces, markers, and data labels with continuous color response	EndColor NeutralColor StartColor
TwoColorAltRamp	Line contours, markers, and data labels with segmented range color response	EndColor StartColor
TwoColorRamp	Gradient contours, surfaces, markers, and data labels with continuous color response	EndColor StartColor

1 Attribute FillPattern is available in style element GraphMissing starting with SAS 9.4M5.

Graphical Data Representation Style Elements (Grouped Data)

The following table lists and describes the data-related style elements that affect grouped data.

Table A2.3 Graphical Style Elements: Data Related (Grouped)

Style Elements	Portion of Graph Affected	Attributes Defined
GraphData1 GraphData2 GraphData3 GraphData4 GraphData5 GraphData6	Primitives related to the first seven grouped data items. Color applies to filled areas. ContrastColor applies to markers and lines.	Color ContrastColor FillPattern[1] LineStyle MarkerSymbol

Style Elements	Portion of Graph Affected	Attributes Defined
GraphData7		
GraphData8 GraphData9 GraphData10 GraphData11	Primitives related to the 8th through 11th grouped data items.	Color ContrastColor FillPattern[1] LineStyle MarkerSymbol[2]
GraphData12	Primitives related to the 12th grouped data item.	Color ContrastColor FillPattern[1] MarkerSymbol[2]
GraphData13[3] GraphData14 GraphData15	Primitives related to the 13th through 15th grouped data items.	FillPattern[1] MarkerSymbol

1 Prior to SAS 9.4M5, style attribute FillPattern is available only with the JOURNAL2, JOURNAL3, and MONOCHROMEPRINTER styles. Starting with SAS 9.4M5, style attribute FillPattern in GraphData1–GraphData11 is also available with the DEFAULT ODS style and all styles that are derived from it.

2 Style attribute MarkerSymbol in these style elements is defined for styles JOURNAL2, JOURNAL3, and MONOCHROMEPRINTER only.

3 Style elements GraphData13–GraphData15 are available only with the JOURNAL2, JOURNAL3, and MONOCHROMEPRINTER styles.

Display Style Elements

The following table lists the display style elements.

Table A2.4 *Display Style Elements*

Style Element	Portion of Graph Affected	Recognized Attributes	Possible Values
GraphAltBlock	Alternate fill color for block plots	Color	GraphColors("gablock")
GraphBand	Display options for confidence bands	DisplayOpts	"Fill fillpattern outline" [1]
GraphBar	Display options for bar charts	DisplayOpts	"Connect fill fillpattern outline"

Style Element	Portion of Graph Affected	Recognized Attributes	Possible Values
GraphBlock	Fill color for block plots	Color	GraphColors("gblock")
GraphBox	Display options for box plots	DisplayOpts	"Caps connect fill fillpattern mean median notches outliers" [1]
		CapStyle	"Serif"
		Connect	"Mean"
GraphContour	Display options for contours	DisplayOpts	"LabeledLineGradient"
		EndColor	GraphColors(‘gramp3cend’)
		NeutralColor	GraphColors(‘gramp3cneutral’)
		StartColor	GraphColors(‘gramp3cstart’)
GraphEllipse	Display options for confidence ellipses	DisplayOpts	"Fill fillpattern outline" [1]
GraphHighLow [2]	Display options for high-low plots	DisplayOpts	"Fill fillpattern outline"
GraphHistogram	Display options for histograms	DisplayOpts	"Fill fillpattern outline" [1]
GraphPolygon [2]	Display options for polygon plots	DisplayOpts	"Fill fillpattern outline"
GraphSkins	Display skins	DataSkin	"Crisp" "Gloss" "Matte" "None" "Pressed"
		KpiSkin	"Basic" "Modern" "None" "Onyx" "Satin"

1 Display option FillPattern is available starting with SAS 9.4M5.

2 Style elements GraphHighLow and GraphPolygon are valid starting with SAS 9.4M5.

Appendix 3

Display Attributes

General Syntax for Attribute Options

Most statements provide options that enable you to specify attributes for the fills, lines, data markers, or text that is used in the display. For example, many plots provide a DATALABELATTRS= option that specifies the attributes of the data labels. This appendix discusses the general syntax for those options and the valid values for they accept.

A statement's attribute options use the following general syntax:

attribute-option-name = *style-element* | *style-element (options)* | *(options)*

style-element
specifies the name of a style element. Only style attributes that are relevant for rendering the fill, line, data marker, or text are used.

Example `fillattrs = graphdata1`

style-element (options)
specifies the name of a style element, plus individual options to be used as style overrides. *Options* is a space-delimited list of *option* = *value* pairs. Any options that are not specified are derived from the specified style element.

Example `fillattrs = graphdata1(color = blue)`

(options)
specifies individual options as a space-delimited list of *option* = *value* pairs. Any options that are not specified are derived from the default style element.

Example
```
datalabelattrs = (family = "Arial" size = 10pt)
datalabelattrs = (family = graphvaluetext:fontfamily)
```

Depending on the attribute option used, the options might be fill options , line options , marker options , or text options .

In general, any relevant attribute that is not specified defaults to an internal value, which is typically derived from the style element that you specify for the attributes. When choosing a style element, you should use an element of the correct type. See Appendix 2, “Graph Style Elements Used by ODS Graphics,” on page 651 for a list of style elements and their types.

Attributes Available for the Attribute Options

Depending on the attribute option used on a statement, the available attributes might be fill-color options on page 664, fill-pattern options on page 665, line options on page 666, marker options on page 668, or text options on page 670.

Fill Color Options

Use one or more of the following options to specify fill-color attributes. The options must be enclosed in parentheses and specified as a space-delimited list of *name* = *value* pairs.

COLOR=*style-reference* | *color*
specifies the fill color. *Style-reference* must specify a valid style attribute such as Color, ContrastColor, StartColor, NeutralColor, or EndColor in the form `style-element-name:attribute-name`. If *style-reference* is not defined in the active ODS style, the COLOR= option is ignored and the default color is used.

Color must be a valid color name, such as RED, or a color code, such as CXFF0000 or #FF0000. The color name must not exceed 64 characters. A color

code must be a valid code for a SAS color-naming scheme, such as RGB, CMYK, HLS, or HSV (HSB). See "Color-Naming Schemes" on page 673.

TRANSPARENCY=*number*
specifies the degree of the transparency of the filled area. This setting enables you to set the transparency for the filled elements of some graph types. You can set just this fill transparency, or set the fill independently of the other transparent elements in the graph. For example, you can use this setting to set the transparency level for the filled bars of a bar chart, and use the bar chart's DATATRANSPARENCY= option to set a different transparency level for the bar outlines.

Default The same as the setting of the statement's DATATRANSPARENCY= option.

Range 0–1, where 0 is opaque and 1 is entirely transparent

Interaction This setting overrides the statement's DATATRANSPARENCY= setting for the fills but not for the outlines.

Example
```
fillattrs = (transparency = 0.5)
```

Fill Pattern Options

Use one or more of the following options to specify fill-pattern attributes. The options must be enclosed in parentheses and specified as a space-delimited list of *name* = *value* pairs.

COLOR=*style-reference* | *color*
specifies the fill color.*Style-reference* must specify a valid style attribute such as Color, ContrastColor, StartColor, NeutralColor, or EndColor in the form `style-element-name:attribute-name`. If *style-reference* is not defined in the active ODS style, the COLOR= option is ignored and the default color is used.

Color must be a valid color name, such as RED, or a color code, such as CXFF0000 or #FF0000. The color name must not exceed 64 characters. A color code must be a valid code for a SAS color-naming scheme, such as RGB, CMYK, HLS, or HSV (HSB). See "Color-Naming Schemes" on page 673.

PATTERN=*style-reference* | *fill-pattern*
specifies the fill pattern. *Style-reference* is valid starting with SAS 9.4M5. It must specify a FILLPATTERN attribute in the form `style-element-name:FILLPATTERN`. If *style-reference* is not defined in the active ODS style, the PATTERN= option is ignored and the default fill pattern is used, if one is specified by the active ODS style.

Fill-pattern is a two-character specification that consists of a line-direction prefix (R for right, L for left, and X for cross hatch) and a line-identification number, 1–5. The following table shows the patterns for each of the possible combinations.

Table A3.1 *Fill Patterns*

Pattern Name	Example	Pattern Name	Example	Pattern Name	Example
L1		R1		X1	
L2		R2		X2	
L3		R3		X3	
L4		R4		X4	
L5		R5		X5	

Line Options

Use one or more of the following options to specify line attributes. The options must be enclosed in parentheses and specified as a space-delimited list of *name* = *value* pairs.

COLOR=*style-reference* | *color*
specifies the line color. *Style-reference* must specify a valid style attribute such as Color, ContrastColor, StartColor, NeutralColor, or EndColor in the form `style-element-name:attribute-name`. If *style-reference* is not defined in the active ODS style, the COLOR= option is ignored and the default color is used.

Color must be a valid color name, such as RED, or a color code, such as CXFF0000 or #FF0000. The color name must not exceed 64 characters. A color code must be a valid code for a SAS color-naming scheme, such as RGB, CMYK, HLS, or HSV (HSB). See "Color-Naming Schemes" on page 673.

PATTERN=*style-reference* | *line-pattern-name* | *line-pattern-number*
specifies the line pattern. *Style-reference* must specify a LineStyle attribute in the form `style-element-name:LINESTYLE`. Line patterns can be specified as a pattern name or pattern number. The following table lists commonly used line patterns.

Table A3.2 *Commonly Used Line Patterns*

Pattern Number	Pattern Name	Example
1	Solid	

Pattern Number	Pattern Name	Example
2	ShortDash	
4	MediumDash	
5	LongDash	
8	MediumDashShortDash	
14	DashDashDot	
15	DashDotDot	
20	Dash	
26	LongDashShortDash	
34	Dot	
35	ThinDot	
41	ShortDashDot	
42	MediumDashDotDot	

Valid line-pattern numbers range from 1 to 46. Not all line-pattern numbers have names. See "Available Line Patterns" on page 670 for a list of all possible line patterns. We recommend that you use the named patterns because they have been optimized to provide good discriminability when used in the same plot.

Note Anti-aliasing might alter the appearance of some line patterns that have fine detail such as line patterns 33 through 46. For example, if you specify the color black and the pattern 33 for a line, and anti-aliasing is enabled, then the line might appear gray. In that case, you can use the ODS GRAPHICS statement to disable anti-aliasing as shown in the following example:

```
ods graphics / antialias = off;
```

THICKNESS=*style-reference* | *dimension*

specifies the line thickness. *Style-reference* must specify a LINETHICKNESS attribute in the form `style-element-name:LINETHICKNESS`.

See "dimension" in *SAS Graph Template Language: Reference*

Marker Options

Use one or more of the following options to specify data-marker attributes. You must enclose the options in parentheses and specify the options as a space-delimited list of *name* = *value* pairs.

COLOR=*style-reference* | *color*
specifies the line color. *Style-reference* must specify a valid style attribute such as Color, ContrastColor, StartColor, NeutralColor, or EndColor in the form `style-element-name:attribute-name`. If *style-reference* is not defined in the active ODS style, the COLOR= option is ignored and the default color is used.

Color must be a valid color name, such as RED, or a color code, such as CXFF0000 or #FF0000. The color name must not exceed 64 characters. A color code must be a valid code for a SAS color-naming scheme, such as RGB, CMYK, HLS, or HSV (HSB). See "Color-Naming Schemes" on page 673.

Restriction This option has no effect on marker symbols that are created with the SYMBOLIMAGE statement.

See "SYMBOLIMAGE" in *SAS Graph Template Language: Reference*

SIZE=*style-reference* | *dimension*
specifies the marker size (both width and height). *Style-reference* must specify a MARKERSIZE attribute in the form `style-element-name:MARKERSIZE`.

See "dimension" in *SAS Graph Template Language: Reference*

SYMBOL=*style-reference* | *marker-name*
specifies the name of the marker. *Style-reference* must specify a MARKERSYMBOL attribute in the form `style-element-name:MARKERSYMBOL`.The following table lists the SAS symbols that are supported.

Table A3.3 *Supported Marker Symbols*

Symbol Name	Plot Symbol	Symbol Name	Plot Symbol
ArrowDown	↓	StarFilled	★
Asterisk	✳	Tack	⊤
Circle	○	Tilde	∿
CircleFilled	●	Triangle	△
Diamond	◇	TriangleFilled	▲
DiamondFilled	◆	TriangleDown	▽

Symbol Name	Plot Symbol	Symbol Name	Plot Symbol
GreaterThan	>	TriangleDownFilled	▼
Hash	#	TriangleLeft	◁
HomeDown	⌂	TriangleLeftFilled	◀
HomeDownFilled	⌂	TriangleRight	▷
IBeam	I	TriangleRightFilled	▶
LessThan	<	Union	∪
Plus	+	X	×
Square	□	Y	Y
SquareFilled	■	Z	Z
Star	☆		

You can also specify the names of symbols that are created by the SYMBOLCHAR and SYMBOLIMAGE statements.

See "SYMBOLCHAR" in *SAS Graph Template Language: Reference*

"SYMBOLIMAGE" in *SAS Graph Template Language: Reference*

TRANSPARENCY=*number*
specifies the degree of transparency for the plot markers.

Default The transparency that is specified by the DATATRANSPARENCY= option, which is 0 by default.

Range 0–1, where 0 is opaque and 1 is entirely transparent

Interaction This suboption overrides the DATATRANSPARENCY= option for the plot markers only.

WEIGHT=NORMAL | BOLD
specifies the marker weight.

Restriction This option has no effect on marker symbols that are created with the SYMBOLCHAR and SYMBOLIMAGE statements.

See "SYMBOLCHAR" in *SAS Graph Template Language: Reference*

"SYMBOLIMAGE" in *SAS Graph Template Language: Reference*

Text Options

Use one or more of the following options to specify text attributes. The options must be enclosed in parentheses and specified as a space-delimited list of *name* = *value* pairs.

COLOR=*style-reference* | *color*
specifies the line color. *Style-reference* must specify a valid style attribute such as Color, ContrastColor, StartColor, NeutralColor, or EndColor in the form `style-element-name:attribute-name`. If *style-reference* is not defined in the active ODS style, the COLOR= option is ignored and the default color is used.

Color must be a valid color name, such as RED, or a color code, such as CXFF0000 or #FF0000. The color name must not exceed 64 characters. A color code must be a valid code for a SAS color-naming scheme, such as RGB, CMYK, HLS, or HSV (HSB). See "Color-Naming Schemes" on page 673.

FAMILY=*style-reference* | *"string"*
specifies the font family of the text. *Style-reference* must specify a FONTFAMILY attribute in the form `style-element-name:FONTFAMILY`.

SIZE=*style-reference* | *dimension*
specifies the font size of the text. *Style-reference* must specify a FONTSIZE attribute in the form `style-element-name:FONTSIZE`.

Restriction The font size cannot be less than the minimum font size in SAS, which is determined by the SAS system. If you specify a font size that is less than the minimum font size, the minimum size is used.

See "dimension" in *SAS Graph Template Language: Reference*

STYLE=*style-reference* | NORMAL | ITALIC
specifies the font style of the text. *Style-reference* must specify a FONTSTYLE attribute in the form `style-element-name:FONTSTYLE`.

WEIGHT=*style-reference* | NORMAL | BOLD
specifies the font weight of the text. *Style-reference* must specify a FONTWEIGHT attribute in the form `style-element-name:FONTWEIGHT`.

Available 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.

Table A3.4 Full List of Line Patterns

Pattern Number	Pattern Name	Example
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		

Pattern Number	Pattern Name	Example
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		

Color-Naming Schemes

Overview of Color-Naming Schemes

The valid color-naming schemes in SAS are as follows:

- RGB (red green blue)
- CMYK (cyan magenta yellow black)
- HLS (hue lightness saturation)
- HSV (hue saturation brightness), also called HSB
- Gray scale
- SAS color names (from the SAS Registry)
- SAS Color Naming System (CNS)

Table A3.5 on page 673 shows examples of each color-naming scheme.

Table A3.5 *Examples of SAS Color-Naming Schemes*

Color-Naming Scheme	Example
RGB	`COLOR=(CX98FB98)`
CMYK	`COLOR=(FF00FF00)` `COLOR=(CMYK00FFFF00)`
HLS	`COLOR=(H14055FF)`
HSV	`COLOR=(V0F055FF)`
Gray Scale	`COLOR=(GRAY4F)`
SAS Registry Colors	`COLOR=(palegreen)`
CNS Color Names	`COLOR=(VeryLightPurplishBlue)` `COLOR=(Very_Light_Purplish_Blue)`

You can also mix color-naming schemes in the same statement. For example:

```
DATACOLORS=(CXEE0044 vivid_blue darkgreen);
```

Each of the color-naming schemes has its advantages and disadvantages based on how the output is used. For example, if you are creating a report that will be viewed only online, then specifying colors using the RGB naming scheme or the SAS color

names defined in the registry might produce better results. If you are creating a report for publishing in printed form, you might want to use the CMYK color-naming scheme.

Note: If you specify an invalid color name, the default color is used instead.

For additional information about color-naming schemes, see *Effective Color Displays: Theory and Practice* by David Travis, *Computer Graphics: Principles and Practice* by Foley, van Dam, Feiner, and Hughes, and http://colorbrewer2.org.

RGB Color Codes

The RGB color-naming scheme is usually used to define colors for a display screen. This color-naming scheme is based on the properties of light. An RGB color code defines a color by combining red, green, and blue colors in different ratios. All the colors combined together create white. The absence of all color creates black.

Color names are in the form CX*rrggbb*, where the following is true:

- CX indicates to SAS that this is an RGB color specification.
- *rr* is the red component.
- *gg* is the green component.
- *bb* is the blue component.

The components are hexadecimal numbers in the range 00–FF (0% to 100%). Each hexadecimal number indicates how much of the red, green, or blue is included in the color. Lower percentage values are darker and higher values are lighter. This scheme allows for up to 256 levels of each color component (more than 16 million different colors).

Table A3.6 *Examples of RGB Color Values*

Color	RGB Value
Red	CXFF0000
Green	CX00FF00
Blue	CX0000FF
White	CXFFFFFF
Black	CX000000

Any combination of the color components is valid. Some combinations match the colors produced by predefined SAS color names. See “Predefined Colors” on page 685.

CMYK Color Codes

The CMYK color-naming scheme is used in four-color printing. CMYK is based on the principles of objects reflecting light. Combining equal values of cyan, magenta, and yellow produces process black, which might not appear as pure black. The black component (K) of CMYK can be used to specify the level of blackness in the output. A lack of all colors produces white when the output is printed on white paper.

To specify the colors from a printer's Pantone Color Lookup Table, you can use the CMYK color-naming scheme. Specify colors in terms of their cyan, magenta, yellow, and black components. Color names are in the form CMYK*ccmmyykk* or K*ccmmyykk*, where the following is true:

- CMYK or K is an optional prefix that indicates to SAS that this is a CMYK color specification.
- *cc* is the cyan component.
- *mm* is the magenta component.
- *yy* is the yellow component.
- *kk* is the black component.

The color-value components are hexadecimal numbers in the range 00–FF, where higher values are darker and lower values are brighter. This scheme allows for up to 256 levels of each color component. When the color value begins with a letter (A–F), you can omit the CMYK or K prefix. When the color value begins with a number (0–9), you must use the CMYK or K prefix.

Table A3.7 *Examples of CMYK Color Values*

Color	CMYK Value
Red	CMYK00FFFF00
Green	FF00FF00
Blue	CMYKFFFF0000
White	K00000000
Process black (using cyan, magenta, and yellow ink)	FFFFFF00
Pure black (using only black ink)	K000000FF

Note: You can specify a CMY value by specifying zero (00) for *kk*, the color's black component.

CMYK color specifications are for output devices that support four colors. If you specify CMYK colors on an output device that supports three colors such as RGB,

the CMYK colors are converted to colors in the three-color space. Because the four-color space supports many more colors than a three-color space, the CMYK colors might map to different colors in the three-color space. To preserve your CMYK colors in that case, specify a device that supports the CMYK color space. See "Color Support for Universal Printers" in *SAS 9.4 Universal Printing*.

HLS Color Codes

The HLS color scheme specifies colors in terms of hue, lightness, and saturation levels. It is based on the Tektronix Color Standard. HLS color names are of the form H*hhhllss*, where the following is true:

- H indicates to SAS that this is an HLS color specification.
- *hhh* is the hue component, which is expressed as an angle.
- *ll* is the lightness component.
- *ss* is the saturation component.

The components are hexadecimal numbers. The hue component is in the range 000–168 hexadecimal, which represents an angular value in the range 0–360 decimal. Hue starts with the primary color blue at 0 degrees, and then progresses to red at 120 degrees, to green at 240 degrees, and back to blue at 360 degrees. Both the lightness and saturation components are in the range 00–FF hexadecimal (0–255 decimal), which provides 256 levels that represent 0% to 100% for each component.

Table A3.8 *Examples of HLS Color Codes*

Color	HLS Color Code
Red	H07880FF
Green	H0F080FF
Blue	H00080FF
Light gray	H000BB00
White[1]	H*xxx*FF00, such as H000FF00
Black[1]	H*xxx*0000 such as H0000000

1 When the saturation is set to 00, the color is a shade of gray that is determined by the lightness value. Therefore, white is defined as H*xxx*FF00 and black as H*xxx*0000, where *xxx* can be any hue.

HSV (or HSB) Color Codes

The HSV color-naming scheme specifies colors in terms of hue, saturation, and value (or brightness) components. HSV color names are of the form V*hhhssvv*, where the following is true:

- V indicates to SAS that this is an HSV color specification.
- *hhh* is the hue component, which is expressed as an angle.
- *ss* is the saturation component.
- *vv* is the value or brightness component.

The components are hexadecimal numbers. The hue component is in the range 000–168 hexadecimal, which represents an angular value in the range 0–360 decimal. Hue starts with the primary color red at 0 degrees, and then progresses to green at 120 degrees, to blue at 240 degrees, and back to red at 360 degrees. Both the saturation and value (brightness) components are in the range 00–FF hexadecimal (0–255 decimal), which provides 256 levels for each component.

Table A3.9 *Examples of HSV (or HSB) Color Codes*

Color	HSV Color Code
Red	V000FFFF
Green	V078FFFF
Blue	V0F0FFFF
Light gray[1]	V*xxx*00BB such as V07900BB
White[1]	V*xxx*00FF such as V07900FF
Black[1]	V*xxx*00000 such as V0790000

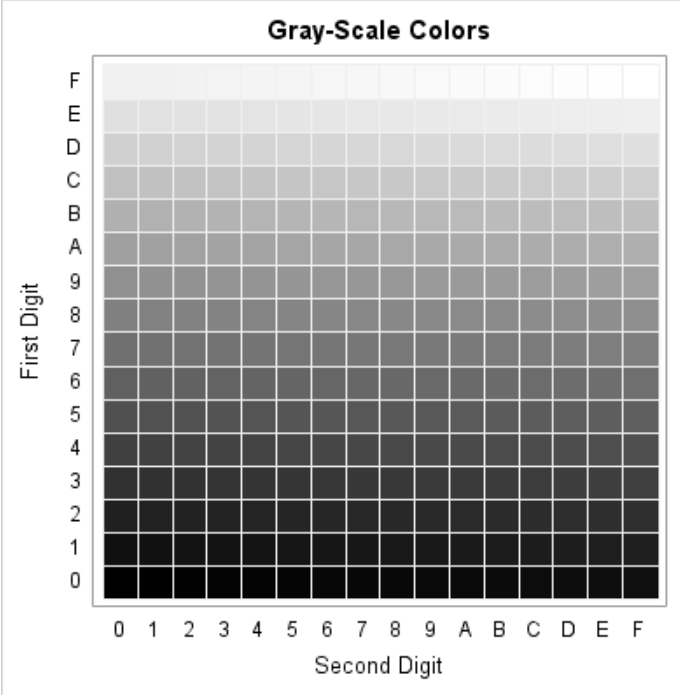
1 When the saturation is set to 00, the color is a shade of gray. The value component determines the intensity of gray level. The *xxx* value can be any hue.

Gray-Scale Color Codes

Gray-scale colors are specified using the word GRAY and a lightness value in the form GRAY*hh*. The value *hh* is the lightness of the gray and is a hexadecimal number in the range 00–FF, which provides 256 levels on the gray scale.

Note: GRAY, without a lightness value, is a SAS color name defined in the SAS registry. (See "SAS Color Names and RGB Values in the SAS Registry" on page 679.) Its value is CX808080. Invalid color specifications are mapped to GRAY.

The following figure shows the gray-scale color for each hexadecimal value.



Color Naming System (CNS) Values

For CNS, you specify a color value by specifying lightness, saturation, and hue, in that order, using the terms shown in the following table.

Table A3.10 *Color Naming System Values*

Lightness	Saturation	Hue
Black	Gray	Blue
Very Dark	Grayish	Purple
Dark	Moderate	Red
Medium	Strong	Orange/Brown
Light	Vivid	Yellow
Very Light		Green

Lightness	Saturation	Hue
White		

Follow these rules when you are determining the CNS color name:

- You should not use the lightness values Black and White with saturation or hue values.
- If you do not specify default values, medium is the default lightness value and vivid is the default saturation value.
- Gray is the only saturation value that can be used without a hue.
- Unless the color that you want is black, white, or some form of gray, you must specify at least one hue.

You can use one or two hue values in the CNS color name. When you use two hue values, the hues must be adjacent to each other in the following list: blue, purple, red, orange/brown, yellow, green, and then returning to blue. Two hue values result in a color that is a combination of both colors. Use the suffix “ish” to reduce the effect of a hue when two hues are combined. Reddish purple is less red than red purple. The color specified with the “ish” suffix must precede the color without the “ish” suffix.

You can write color names in the following ways:

- without space separators between words
- with an underscore to separate words

Do not enclose color names in quotation marks in GTL. For example, the following color specifications are valid:

- verylightmoderatepurplishblue
- very_light_moderate_purplish_blue

Note: If a CNS color name is also a color name in the SAS Registry, the SAS Registry color value takes precedence. Some CNS color names and color names in the SAS Registry have different color values. Dark blue is an example. To use a CNS color value when the color name is also in the SAS Registry, include an underscore to separate the words. Here is an example.

```
color=dark_blue
```

SAS Color Names and RGB Values in the SAS Registry

The SAS Registry provides a set of color names and RGB values that you can use to specify colors. These color names and RGB values are common to most web browsers. You can specify the name itself or the RGB value associated with that color name. To view the color names as associated RGB values that are defined in the registry, submit the following code:

```
proc registry list
 startat="COLORNAMES";
run;
```

SAS prints the output in the SAS log. Here is a partial listing.

```
NOTE: Contents of SASHELP REGISTRY starting at subkey [COLORNAMES]
[  COLORNAMES]
  Active="HTML"
[    HTML]
    AliceBlue=hex: F0,F8,FF
    AntiqueWhite=hex: FA,EB,D7
    Aqua=hex: 00,FF,FF
    Aquamarine=hex: 7F,FD,D4
    Azure=hex: F0,FF,FF
    Beige=hex: F5,F5,DC
    ...
```

For a list of the color names that are defined in the SAS Registry, see "Predefined Colors" on page 685.

You can also create your own color values by adding them to the SAS registry. For information about viewing and modifying the list of color names, see "The SAS Registry" in *SAS Programmer's Guide: Essentials*.

Converting Color Values between Color-Naming Schemes

If the SAS/GRAPH software is included in your SAS installation, several macros are provided with SAS/GRAPH that enable you to perform selected conversions between color-naming schemes. The following table shows the conversions that are possible using these macros.

From Color Value	To Color Value	Conversion Macro
CMY	RGB	%CMY
CNS	HLS	%CNS
RGB	HLS	%RGB2HLS
HLS	RGB	%HLS2RGB

For information about these macros, see *SAS/GRAPH: Reference*.

Converting Numeric Color Component Values to Color Names

If the SAS/GRAPH software is included in your SAS installation, several macros are provided with SAS/GRAPH that enable you to convert numeric color component values to color names. The following table shows the macros that you can use for this purpose.

From Numeric Color Component Values	To Color Name	Conversion Macro
cyan, magenta, yellow, black	CMYK	%CMYK
hue, lightness, saturation	HLS	%HLS
hue, saturation, value	HSV	%HSV
red, green, blue	RGB	%RGB

Converting Color Values from Other Applications

Many software programs enable you to change or customize various colors. A dialog box typically provides a means of selecting a different color or modifying the attributes of an existing color. In the SAS ODS Graphics Editor, for example, the More Colors dialog box shown in the following figure serves this purpose.

Figure A3.1 More Colors Dialog Box

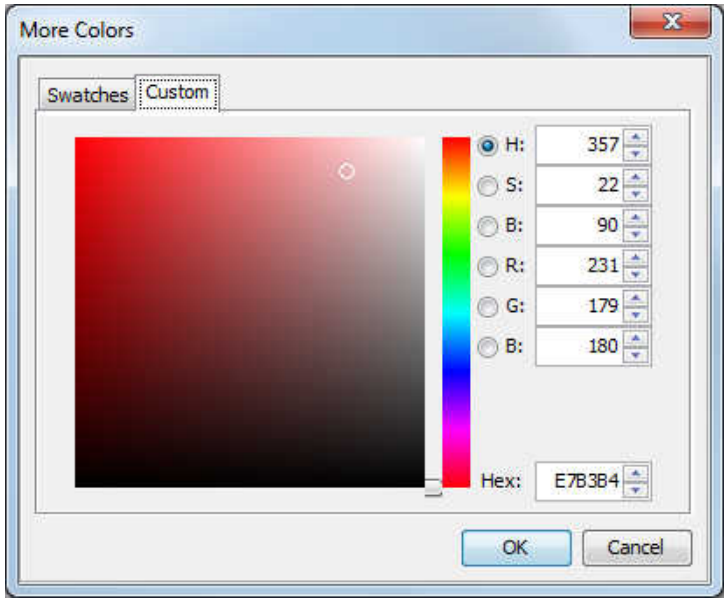


Figure A3.1 on page 682 shows the HSB and RGB numeric color component values for the currently selected color. It also shows the RGB values in hexadecimal. To use the RGB specification for this color in a SAS program, add the CX prefix to the **Hex** value E7B3B4. If the application provides only the numeric component values, you must convert the decimal component values to hexadecimal. In Figure A3.1 on page 682, the **H** value is in the range 0–360 degrees, and the **S** and **B** values are each in the range 0–100 percent. The **R**, **G**, and **B** values are in the range 0–255 each. You can convert the component values manually. (See "Understanding Hexadecimal Values" on page 713.) Be aware that you must first convert the **S** and **B** values from percentages to 255-based values by rounding the result of the following computation to the nearest integer:

$$\left(\frac{Value}{100}\right) \times 255$$

If the SAS/GRAPH software is included in your SAS installation, you can also use the SAS/GRAPH color utility macros to convert the values. The %HSV color utility macro converts HSV (HSB) numeric color component values to an HSV color name. Likewise, the %RGB macro converts RGB numeric color component values to an RGB color name. The following example shows how to use the %HSV color utility macro to convert the **H**, **S**, and **B** color component values in Figure A3.1 on page 682 to an HSV color name.

```
%COLORMAC;
data _null_;
  put "%HSV(357,22,90)";
run;
```

Note: The %COLORMAC macro compiles all of the SAS/GRAPH color utility macros. You need to run it only once during a SAS session.

Because the %HSV macro accepts values in the range 0–100 as a percentage of 255 for saturation and value, use the **S** and **B** values as shown. The result is V16538E6.

The following example shows how to use the %RGB color utility macro to convert the **R**, **G**, and **B** numeric color component values in Figure A3.1 on page 682 to an RGB color name.

```
/* Compute the RGB percentages */
data _null_;
  r = 231;
  g = 179;
  b = 180;
  call symputx("r", round((r/255)*100));
  call symputx("g", round((g/255)*100));
  call symputx("b", round((b/255)*100));
run;

/* Convert to RGB color name */
%COLORMAC;
data _null_;
  put "%RGB(&r,&g,&b)";
run;
```

Because the %RGB color utility macro accepts integer values in the range 0–100 as a percentage of 255, you must convert the 255-based values to integer percentages in order to use them in the %RGB macro call. The result is CXE8B3B5. The result from the %RGB macro is not exact due to rounding. If you want more exact results, use the following program.

```
data _null_;
  r = 231;
  g = 179;
  b = 180;
  rgb="CX" || put(r,hex2.) || put(g,hex2.) || put(b,hex2.);
  put rgb;
run;
```

The result is CXE7B3B4.

To convert the RGB color name to an HLS color name, use the %RGB2HLS macro as shown in the following program.

```
%COLORMAC;
data _null_;
  put "%RGB2HLS(CXE7B3B4)";
run;
```

The result is H077CD84.

For more information about the SAS/GRAPH color utility macros, see *SAS/GRAPH: Reference*.

Appendix 4

Predefined Colors

Predefined Colors

The following table shows the colors that are predefined in the SAS Registry, arranged by name. For each color, the equivalent CX color code (RGB), the equivalent HLS color code, and a color sample are shown. For a table of the SAS registry colors arranged by hue, see Table A4.2 on page 692.

Table A4.1 *SAS Registry Colors by Name*

Color Name	Color Sample	CX Color Code (RGB)	HLS Color Code[1]
AliceBlue		CXF0F8FF	H148F8FF
AntiqueWhite		CXFAEBD7	H09AE9C6
Aqua		CX00FFFF	H12C80FF
Aquamarine		CX7FFDD4	H118BEF7
Azure		CXF0FFFF	H12CF8FF
Beige		CXF5F5DC	H0B4E98E
Bisque		CXFFE4C4	H098E2FF
Black		CX000000	H0000000

Color Name	Color Sample	CX Color Code (RGB)	HLS Color Code[1]
BlanchedAlmond		CXFFEBCD	H09CE6FF
Blue		CX0000FF	H00080FF
BlueViolet		CX8A2BE2	H01F87C2
Brown		CXA52A2A	H0786898
Burlywood		CXDEB887	H099B391
CadetBlue		CX5F9EA0	H12D8041
Chartreuse		CX7FFF00	H0D280FF
Chocolate		CXD2691E	H09178BF
Coral		CXFF7F50	H088A8FF
CornflowerBlue		CX6495ED	H152A9CA
Cornsilk		CXFFF8DC	H0A7EEFF
Crimson		CXDC143C	H06C78D5
Cyan		CX00FFFF	H12C80FF
DarkBlue		CX00008B	H00046FF
DarkCyan		CX008B8B	H12C46FF
DarkGoldenrod		CXB8860B	H0A262E2
DarkGray		CXA9A9A9	H000A900
DarkGreen		CX006400	H0F032FF
DarkKhaki		CXBDB76B	H0AF9462
DarkMagenta		CX8B008B	H03C46FF
DarkOliveGreen		CX556B2F	H0CA4D63
DarkOrange		CXFF8C00	H09880FF

Color Name	Color Sample	CX Color Code (RGB)	HLS Color Code[1]
DarkOrchid		CX9932CC	H0287F9B
DarkRed		CX8B0000	H07846FF
DarkSalmon		CXE9967A	H087B2B7
DarkSeaGreen		CX8FBC8F	H0F0A640
DarkSlateBlue		CX483D8B	H0086463
DarkSlateGray		CX2F4F4F	H12C3F41
DarkSlateGrey		CX2F2F2F	H0002F00
DarkTurquoise		CX00CED1	H12C69FF
DarkViolet		CX9400D3	H02A6AFF
DeepPink		CXFF1493	H0578AFF
DeepSkyBlue		CX00BFFF	H13B80FF
DimGray		CX696969	H0006900
DodgerBlue		CX1E90FF	H1498FFF
Firebrick		CXB22222	H0786AAD
FloralWhite		CXFFFAF0	H0A0F8FF
ForestGreen		CX228B22	H0F0579B
Fuchsia		CXFF00FF	H03C80FF
Gainsboro		CXDCDCDC	H000DC00
GhostWhite		CXF8F8FF	H000FCFF
Gold		CXFFD700	H0AA80FF
Goldenrod		CXDAA520	H0A27DBE
Gray or Grey		CX808080	H0008000

Color Name	Color Sample	CX Color Code (RGB)	HLS Color Code[1]
Green		CX008000	H0F040FF
GreenYellow		CXADFF2F	H0CB97FF
Honeydew		CXF0FFF0	H0F0F8FF
HotPink		CXFF69B4	H05AB4FF
IndianRed		CXCD5C5C	H0789587
Indigo		CX4B0082	H02241FF
Ivory		CXFFFFF0	H0B4F8FF
Khaki		CXF0E68C	H0AEBEC4
Lavender		CXE6E6FA	H000F0AA
LavenderBlush		CXFFF0F5	H064F8FF
LawnGreen		CX7CFC00	H0D27EFF
LemonChiffon		CXFFFACD	H0AEE6FF
LightBlue		CXADD8E6	H13ACA88
LightCoral		CXF08080	H078B8C9
LightCyan		CXE0FFFF	H12CF0FF
LightGoldenrodYellow		CXFAFAD2	H0B4E6CC
LightGray		CXD3D3D3	H000D300
LightGreen		CX90EE90	H0F0BFBB
LightPink		CXFFB6C1	H06EDBFF
LightSalmon		CXFFA07A	H089BDFF
LightSeaGreen		CX20B2AA	H12869B1
LightSkyBlue		CX87CEFA	H142C1EB

Color Name	Color Sample	CX Color Code (RGB)	HLS Color Code[1]
LightSlateGray		CX778899	H14A8824
LightSteelBlue		CXB0C4DE	H14DC769
LightYellow		CXFFFFE0	H0B4F0FF
Lime		CX00FF00	H0F080FF
LimeGreen		CX32CD32	H0F0809B
Linen		CXFAF0E6	H095F0AA
Magenta		CXFF00FF	H03C80FF
Maroon		CX800000	H07840FF
MediumAquamarine		CX66CDAA	H1179A81
MediumBlue		CX0000CD	H00067FF
MediumOrchid		CXBA55D3	H0309496
MediumPurple		CX9370DB	H013A698
MediumSeaGreen		CX3CB371	H10A787F
MediumSlateBlue		CX7B68EE	H008ABCB
MediumSpringGreen		CX00FA9A	H1147DFF
MediumTurquoise		CX48D1CC	H1298D99
MediumVioletRed		CXC71585	H0526ECE
MidnightBlue		CX191970	H00045A2
MintCream		CXF5FFFA	H10EFAFF
MistyRose		CXFFE4E1	H07EF0FF
Moccasin		CXFFE4B5	H09EDAFF
NavajoWhite		CXFFDEAD	H09BD6FF

Color Name	Color Sample	CX Color Code (RGB)	HLS Color Code[1]
Navy		CX000080	H00040FF
OldLace		CXFDF5E6	H09FF2D9
Olive		CX808000	H0B440FF
OliveDrab		CX6B8E23	H0C7599A
Orange		CXFFA500	H09E80FF
OrangeRed		CXFF4500	H08880FF
Orchid		CXDA70D6	H03EA596
PaleGoldenrod		CXEEE8AA	H0AECCAA
PaleGreen		CX98FB98	H0F0CAEC
PaleTurquoise		CXAFEEEE	H12CCFA6
PaleVioletRed		CXDB7093	H064A698
PapayaWhip		CXFFEFD5	H09DEAFF
Peachpuff		CXFFDAB9	H094DCFF
Peru		CXCD853F	H0958696
Pink		CXFFC0CB	H06DE0FF
Plum		CXDDA0DD	H03CBF79
PowderBlue		CXB0E0E6	H132CB84
Purple		CX800080	H03C40FF
Red		CXFF0000	H07880FF
RosyBrown		CXBC8F8F	H078A640
RoyalBlue		CX4169E1	H15991B9
SaddleBrown		CX8B4513	H0914FC2

Color Name	Color Sample	CX Color Code (RGB)	HLS Color Code[1]
Salmon		CXFA8072	H07EB6EE
SandyBrown		CXF4A460	H093AADE
SeaGreen		CX2E8B57	H10A5D80
Seashell		CXFFF5EE	H090F7FF
Sienna		CXA0522D	H08B678F
Silver		CXC0C0C0	H000C000
SkyBlue		CX87CEEB	H13DB9B6
SlateBlue		CX6A5ACD	H0089488
SlateGray		CX708090	H14A8020
Snow		CXFFFAFA	H078FDFF
SpringGreen		CX00FF7F	H10D80FF
SteelBlue		CX4682B4	H1477D70
Tan		CXD2B48C	H09AAF70
Teal		CX008080	H12C40FF
Thistle		CXD8BFD8	H03CCC3E
Tomato		CXFF6347	H081A3FF
Turquoise		CX40E0D0	H12690B8
Violet		CXEE82EE	H03CB8C2
Wheat		CXF5DEB3	H09FD4C4
White		CXFFFFFF	H000FF00
WhiteSmoke		CXF5F5F5	H000F500
Yellow		CXFFFF00	H0B480FF

Color Name	Color Sample	CX Color Code (RGB)	HLS Color Code[1]
YellowGreen		CX9ACD32	H0C7809B

1 The HLS color naming scheme in SAS follows the Tektronix Color Standard, which places blue at 0 degrees on the hue coordinate. For more information, see "HLS Color Codes" on page 676.

The following table shows the colors that are predefined in the SAS Registry, arranged by hue according to the Tektronix model. For each color, a color sample, and the equivalent RGB (CX) and HLS color codes are shown.

Table A4.2 *SAS Registry Colors by Hue*

Color Sample	Color Name	CX Code (RGB)	HLS Code[1]
	Black	CX000000	H0000000
	DarkSlateGrey	CX2F2F2F	H0002F00
	DimGray	CX696969	H0006900
	Gray or Grey	CX808080	H0008000
	DarkGray	CXA9A9A9	H000A900
	Silver	CXC0C0C0	H000C000
	LightGray	CXD3D3D3	H000D300
	Gainsboro	CXDCDCDC	H000DC00
	WhiteSmoke	CXF5F5F5	H000F500
	White	CXFFFFFF	H000FF00
	MidnightBlue	CX191970	H00045A2
	Lavender	CXE6E6FA	H000F0AA
	Navy	CX000080	H00040FF
	DarkBlue	CX00008B	H00046FF
	MediumBlue	CX0000CD	H00067FF
	Blue	CX0000FF	H00080FF

Color Sample	Color Name	CX Code (RGB)	HLS Code[1]
	GhostWhite	CXF8F8FF	H000FCFF
	SlateBlue	CX6A5ACD	H0089488
	DarkSlateBlue	CX483D8B	H0086463
	MediumSlateBlue	CX7B68EE	H008ABCB
	MediumPurple	CX9370DB	H013A698
	BlueViolet	CX8A2BE2	H01F87C2
	Indigo	CX4B0082	H02241FF
	DarkOrchid	CX9932CC	H0287F9B
	DarkViolet	CX9400D3	H02A6AFF
	MediumOrchid	CXBA55D3	H0309496
	Thistle	CXD8BFD8	H03CCC3E
	Plum	CXDDA0DD	H03CBF79
	Violet	CXEE82EE	H03CB8C2
	Purple	CX800080	H03C40FF
	DarkMagenta	CX8B008B	H03C46FF
	Fuchsia	CXFF00FF	H03C80FF
	Magenta	CXFF00FF	H03C80FF
	Orchid	CXDA70D6	H03EA596
	MediumVioletRed	CXC71585	H0526ECE
	DeepPink	CXFF1493	H0578AFF
	HotPink	CXFF69B4	H05AB4FF
	LavenderBlush	CXFFF0F5	H064F8FF

Color Sample	Color Name	CX Code (RGB)	HLS Code[1]
	PaleVioletRed	CXDB7093	H064A698
	Crimson	CXDC143C	H06C78D5
	Pink	CXFFC0CB	H06DE0FF
	LightPink	CXFFB6C1	H06EDBFF
	RosyBrown	CXBC8F8F	H078A640
	IndianRed	CXCD5C5C	H0789587
	Brown	CXA52A2A	H0786898
	Firebrick	CXB22222	H0786AAD
	LightCoral	CXF08080	H078B8C9
	Maroon	CX800000	H07840FF
	DarkRed	CX8B0000	H07846FF
	Red	CXFF0000	H07880FF
	Snow	CXFFFAFA	H078FDFF
	MistyRose	CXFFE4E1	H07EF0FF
	Salmon	CXFA8072	H07EB6EE
	Tomato	CXFF6347	H081A3FF
	DarkSalmon	CXE9967A	H087B2B7
	Coral	CXFF7F50	H088A8FF
	OrangeRed	CXFF4500	H08880FF
	LightSalmon	CXFFA07A	H089BDFF
	Sienna	CXA0522D	H08B678F
	Seashell	CXFFF5EE	H090F7FF

Color Sample	Color Name	CX Code (RGB)	HLS Code[1]
	Chocolate	CXD2691E	H09178BF
	SaddleBrown	CX8B4513	H0914FC2
	SandyBrown	CXF4A460	H093AADE
	Peachpuff	CXFFDAB9	H094DCFF
	Peru	CXCD853F	H0958696
	Linen	CXFAF0E6	H095F0AA
	Bisque	CXFFE4C4	H098E2FF
	DarkOrange	CXFF8C00	H09880FF
	Burlywood	CXDEB887	H099B391
	Tan	CXD2B48C	H09AAF70
	AntiqueWhite	CXFAEBD7	H09AE9C6
	NavajoWhite	CXFFDEAD	H09BD6FF
	BlanchedAlmond	CXFFEBCD	H09CE6FF
	PapayaWhip	CXFFEFD5	H09DEAFF
	Moccasin	CXFFE4B5	H09EDAFF
	Orange	CXFFA500	H09E80FF
	Wheat	CXF5DEB3	H09FD4C4
	OldLace	CXFDF5E6	H09FF2D9
	FloralWhite	CXFFFAF0	H0A0F8FF
	DarkGoldenrod	CXB8860B	H0A262E2
	Goldenrod	CXDAA520	H0A27DBE
	Cornsilk	CXFFF8DC	H0A7EEFF

Color Sample	Color Name	CX Code (RGB)	HLS Code[1]
	Gold	CXFFD700	H0AA80FF
	Khaki	CXF0E68C	H0AEBEC4
	LemonChiffon	CXFFFACD	H0AEE6FF
	PaleGoldenrod	CXEEE8AA	H0AECCAA
	DarkKhaki	CXBDB76B	H0AF9462
	Beige	CXF5F5DC	H0B4E98E
	LightGoldenrodYellow	CXFAFAD2	H0B4E6CC
	Olive	CX808000	H0B440FF
	Yellow	CXFFFF00	H0B480FF
	LightYellow	CXFFFFE0	H0B4F0FF
	Ivory	CXFFFFF0	H0B4F8FF
	OliveDrab	CX6B8E23	H0C7599A
	YellowGreen	CX9ACD32	H0C7809B
	DarkOliveGreen	CX556B2F	H0CA4D63
	GreenYellow	CXADFF2F	H0CB97FF
	Chartreuse	CX7FFF00	H0D280FF
	LawnGreen	CX7CFC00	H0D27EFF
	DarkSeaGreen	CX8FBC8F	H0F0A640
	ForestGreen	CX228B22	H0F0579B
	LimeGreen	CX32CD32	H0F0809B
	LightGreen	CX90EE90	H0F0BFBB
	PaleGreen	CX98FB98	H0F0CAEC

Color Sample	Color Name	CX Code (RGB)	HLS Code[1]
	DarkGreen	CX006400	H0F032FF
	Green	CX008000	H0F040FF
	Lime	CX00FF00	H0F080FF
	Honeydew	CXF0FFF0	H0F0F8FF
	SeaGreen	CX2E8B57	H10A5D80
	MediumSeaGreen	CX3CB371	H10A787F
	SpringGreen	CX00FF7F	H10D80FF
	MintCream	CXF5FFFA	H10EFAFF
	MediumSpringGreen	CX00FA9A	H1147DFF
	MediumAquamarine	CX66CDAA	H1179A81
	Aquamarine	CX7FFDD4	H118BEF7
	Turquoise	CX40E0D0	H12690B8
	LightSeaGreen	CX20B2AA	H12869B1
	MediumTurquoise	CX48D1CC	H1298D99
	DarkSlateGray	CX2F4F4F	H12C3F41
	PaleTurquoise	CXAFEEEE	H12CCFA6
	Teal	CX008080	H12C40FF
	DarkCyan	CX008B8B	H12C46FF
	Aqua	CX00FFFF	H12C80FF
	Cyan	CX00FFFF	H12C80FF
	LightCyan	CXE0FFFF	H12CF0FF
	Azure	CXF0FFFF	H12CF8FF

Color Sample	Color Name	CX Code (RGB)	HLS Code[1]
	DarkTurquoise	CX00CED1	H12C69FF
	CadetBlue	CX5F9EA0	H12D8041
	PowderBlue	CXB0E0E6	H132CB84
	LightBlue	CXADD8E6	H13ACA88
	DeepSkyBlue	CX00BFFF	H13B80FF
	SkyBlue	CX87CEEB	H13DB9B6
	LightSkyBlue	CX87CEFA	H142C1EB
	SteelBlue	CX4682B4	H1477D70
	AliceBlue	CXF0F8FF	H148F8FF
	DodgerBlue	CX1E90FF	H1498FFF
	SlateGray	CX708090	H14A8020
	LightSlateGray	CX778899	H14A8824
	LightSteelBlue	CXB0C4DE	H14DC769
	CornflowerBlue	CX6495ED	H152A9CA
	RoyalBlue	CX4169E1	H15991B9

1 The HLS color naming scheme in SAS follows the Tektronix Color Standard, which places blue at 0 degrees on the hue coordinate. For more information, see "HLS Color Codes" on page 676.

Appendix 5

Tick Value Fit Policy Applicability

Tick Value Fit Policy Applicability Matrix

Table A5.1 on page 700 provides a matrix of the tick value fit policies and the axes to which each applies in the OVERLAY, LATTICE, DATALATTICE, DATAPANEL, and EQUATED layouts. In the matrix, the notations V, H, and B are used to indicate the axes to which each policy applies for a specific case. V indicates that the policy applies to the vertical axis only (Y, Y2, and row axes). H indicates that the policy applies to the horizontal axes only (X, X2, and column axes). B indicates that the policy applies to both the horizontal and vertical axes.

Table A5.1 *Tick Value Fit Policy Applicability Matrix*

Fit Policy	LAYOUT OVERLAY			LAYOUT LATTICE LAYOUT DATALATTICE LAYOUT DATAPANEL			LAYOUT EQUATED	Notes
	Discrete Axes	Linear Axes	Time Axes	Discrete Axes	Linear Axes	Time Axes		
EXTRACT	B			B				The tick values are extracted to an axis legend only if the values cannot be fit on the axis.
EXTRACTALWAYS	B			B				The tick values are always extracted to an axis legend even if the values can be fit on the axis.
NONE	V			V				No adjustment is attempted.
ROTATE	H	H	H	H	H	H	H	The tick values are rotated 45 degrees when a collision occurs.
ROTATEALWAYS	H	H	H	H	H	H	H	The tick values are always rotated 45 degrees regardless of whether a collision occurs.
ROTATEALWAYSDROP	H			H				The tick values are always rotated 45 degrees regardless of whether a collision occurs. If unsuccessful, the values are dropped.
ROTATETHIN	H	H	H	H	H	H	H	The ROTATE policy is attempted first. If unsuccessful, the THIN policy is applied.
SPLIT	B			B				The tick values on a discrete axis are split into multiple lines on blank spaces by default when a collision occurs.

Fit Policy	LAYOUT OVERLAY			LAYOUT LATTICE LAYOUT DATALATTICE LAYOUT DATAPANEL			LAYOUT EQUATED	Notes
	Discrete Axes	Linear Axes	Time Axes	Discrete Axes	Linear Axes	Time Axes		
SPLITALWAYS	B			B				The tick values on a discrete axis are always split into multiple lines on every blank space in the value by default regardless of whether a collision occurs.
SPLITALWAYSTHIN	V			V				The SPLITALWAYS policy is attempted first. If unsuccessful, the THIN policy is applied.
SPLITROTATE	H			H				The SPLIT policy is attempted first. If unsuccessful, the ROTATE policy is applied.
SPLITTHIN	V			V				The SPLIT policy is attempted first. If unsuccessful, the THIN policy is applied.
STACKEDALWAYS[1]	H	H	H	H	H	H		The tick values are always displayed vertically as stacked letters.
STACKEDALWAYSTHIN[1]	H	H	H	H	H	H		The STACKEDALWAYS policy is attempted first. If unsuccessful, the THIN policy is applied.
STAGGER	H	H	H	H	H	H	H	The tick values alternate between two rows.
STAGGERROTATE	H	H	H	H	H	H	H	The STAGGER policy is attempted first. If unsuccessful, the ROTATE policy is applied.
STAGGERTHIN	H	H	H	H	H	H	H	The STAGGER policy is attempted first. If unsuccessful, the THIN policy is applied.

Fit Policy	LAYOUT OVERLAY			LAYOUT LATTICE LAYOUT DATALATTICE LAYOUT DATAPANEL			LAYOUT EQUATED	Notes
	Discrete Axes	Linear Axes	Time Axes	Discrete Axes	Linear Axes	Time Axes		
STAGGERTRUNCATE	H			H				The STAGGER policy is attempted first. If unsuccessful, the TRUNCATE policy is applied.
THIN	B	B	H	B	B	H	B	Some tick values are removed.
TRUNCATE	H			H				The tick values are shortened.
TRUNCATEROTATE	H			H				The TRUNCATE policy is attempted first. If unsuccessful, the ROTATE policy is applied.
TRUNCATESTAGGER	H			H				The TRUNCATE policy is attempted first. If unsuccessful, the STAGGER policy is applied.
TRUNCATETHIN	H			H				The TRUNCATE policy is attempted first. If unsuccessful, the THIN policy is applied.

1 Valid starting with SAS 9.4M5.

Appendix 6

SAS Formats Not Supported

Assigning Formats

SAS formats and user-defined formats can be assigned to input data columns with the FORMAT statement of the SGRENDER procedure. Also, several GTL statement options accept a format as an option value. Examples include the TICKVALUEFORMAT= option for formatting axis tick values, and the TIPFORMAT= option for formatting data tips. For information about SAS formats, see *SAS Formats and Informats: Reference*. For information about user-defined formats, see "FORMAT Procedure" in *Base SAS Procedures Guide*.

Not all SAS formats and user-defined formats are supported in GTL or with the SGPLOT, SGSCATTER, SGPANEL, and SGRENDER procedures. See "SAS Formats That Are Not Supported" on page 704. When an unsupported format is encountered, a note similar to the following is written to the SAS log:

```
TICKVALUEFORMAT=bestx. is invalid. The format is invalid or unsupported.
 The default will be used.
```

Using Locale-Sensitive Decimal Separators with Numeric Formats

System option NLDESCEPARATOR= is not supported by ODS Graphics. When system option NLDECSEPARATOR is in effect, syntax errors or null debug errors might result when using an ODS Graphics enabled procedure. For locale-sensitive decimal separators in your graph's numeric values, specify system option NONLDESCEPARATOR, and then use appropriate NL formats to format your numeric variables. See "Dictionary of Formats for NLS" in *SAS National Language Support (NLS): Reference Guide*.

SAS Formats That Are Not Supported

Unsupported Numeric Formats

The following numeric formats are not supported in ODS Graphics:

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 ISO 8601

The following date and time formats are not supported in ODS Graphics:

$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 ODS Graphics:

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 ODS Graphics:

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

User-Defined Formats That Are Not Supported

Picture Formats with Date, Time, or Datetime Directives

ODS Graphics does not support user-defined picture formats that specify date, time, or datetime directives such as %a, %W, %H, and so on. Using a picture format with directives produces unexpected results in ODS Graphics. To work around this limitation, apply the picture format to a new column in your data, and then plot the new preformatted column instead. Here is an example that creates column NewDate by formatting the Date column as MM/YYYY.

```
/* Create picture format MonYear for MM/YYYY date */
proc format;
  picture monyear (min=7)
    low-high='%0m/%Y' (datatype=date);
run;

/* Add preformatted date column NewDate to the data */
data stocks;
  set sashelp.stocks
    (where=(year(date) eq 2001 and stock eq "IBM"));
  newdate = put(date, monyear.);
  label newdate="Date: MM/YYYY";
run;
```

The NewDate column can be used in ODS Graphics plot statements instead of the Date column.

For information about the PICTURE statement in the FORMAT procedure, see *Base SAS Procedures Guide*.

Unicode Values with a User-Defined ODS Escape Character

ODS Graphics supports Unicode values in user-defined formats only if they are preceded by the (*ESC*) escape sequence as shown in the following example.

```
"(*ESC*){unicode beta}"
```

ODS Graphics does not support the use of a user-defined ODS escape character to escape Unicode values in user-defined formats.

For an example of how to use Unicode values in user-defined formats with ODS Graphics, see “Formatting the Tick Values on a Discrete Axis” on page 141.

Appendix 7

Memory Management for ODS Graphics

SAS Options Affecting Memory

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. Also, the Java environment is automatically started and stopped for you. When Java is started, it allocates a fixed amount of memory. The memory 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 in the following cases:

- 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 scatter plot matrix has a large number of variables
- creating 3-D plots and 2-D 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

Managing a Java Out of Memory Error

If you encounter a Java OutOfMemoryError, then 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 shows the value of the JREOPTIONS option:

```
proc options option=jreoptions;
run;
```

After you submit this procedure code, a list of JREOPTIONS settings is written to 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 -Xmx and -Xms suboptions:

```
JREOPTIONS=(/* other Java suboptions */ -Xmx128m -Xms128m)
```

-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)
```

Alternatively, 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, then 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, then 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.

Appendix 8

Understanding Hexadecimal Values

Understanding Hexadecimal Values

This section provides an overview of the hexadecimal numbering system. The hexadecimal system is a base 16 numbering system where each digit represents one of the values shown in the following table.

Hexadecimal Value	Decimal Equivalent
0–9	0–9
A	10
B	11
C	12
D	13
E	14
F	15

The maximum decimal value that a hexadecimal value can represent is $16^n - 1$, where *n* is the number of digits in the hexadecimal value. A two-digit hexadecimal value can represent a maximum decimal value of 255, and a four-digit hexadecimal value can represent a maximum decimal value of 65,535. To convert a hexadecimal

number to decimal, sum the product of each hexadecimal character and its base power, 16^n, where *n* is the digit's significance. For example, to convert hexadecimal value C8A4 to decimal manually, do the following:

$$\left(12 \times 16^3\right) + \left(8 \times 16^2\right) + \left(10 \times 16\right) + 4 = 51364$$

To convert from decimal to hexadecimal manually, iteratively divide the decimal value by 16. In each iteration, multiply the remainder by 16 to get the hexadecimal character for that iteration, and then use the quotient in the next iteration until the quotient is zero. This method generates the hexadecimal value from the least-significant digit to the most significant digit. The following table demonstrates how to convert decimal value 51,364 back to its hexadecimal value.

Divide by 16	Quotient	Remainder	16 x Remainder	Hexadecimal Value
51364 / 16	3210	0.25	4	4
3210 / 16	200	0.625	10	A
200 / 16	12	0.5	8	8
12 / 16	0	0.75	12	C

Since the hexadecimal characters are generated from the least-significant digit to the most significant, the result is C8A4.

Apart from the base and the value representation, the methods for manipulating hexadecimal values, such as addition, subtraction, and multiplication, are the same as those that are used in the base 10 system. Here are some examples.

Base 10	Base 16
9 + 1 = 10	9 + 1 = A
8 + 8 = 16	8 + 8 = 10
3 x 85 = 255	3 x 55 = FF

Many pocket calculators and calculator programs enable you to manipulate hexadecimal values and convert values between hexadecimal and decimal. You can also use a DATA step in SAS to manipulate and convert hexadecimal values. Here is a simple macro that uses the INPUT statement in a DATA step to convert a hexadecimal value to its equivalent decimal value.

```
%macro hex2dec(hex);
  data _null_;
    msg=cat("#", %upcase("&hex"), " = ", input("&hex", hex.));
    put msg;
  run;
%mend hex2dec;
```

The following example shows how to use this macro to convert the hexadecimal value C8A4 to its decimal equivalent.

```
%hex2dec(c8a4);
```

```
#C8A4 = 51364
```

Here is a simple macro that uses the PUT statement in a DATA step to convert a decimal value to its equivalent hexadecimal value.

```
%macro dec2hex(dec);
  data _null_;
    msg=cat("&dec = #", put(&dec, hex4.));
    put msg;
  run;
%mend dec2hex;
```

The following example shows how to use this macro to convert the decimal value 51,364 back to its hexadecimal equivalent.

```
%dec2hex(51364);
51364 = #C8A4
```

For information about macros, see *SAS Macro Language: Reference*. For information about the INPUT and PUT DATA step functions, see *SAS Functions and CALL Routines: Reference*.

When you use the hexadecimal system to define you own colors, the exact value for the color is not as significant as the relationship of the digits in the value to each other, and the relationship of the value of this color to other colors.

Appendix 9

ODS Graphics and SAS/GRAPH

ODS Graphics and SAS/GRAPH

SAS produces graphics using two very distinct systems: ODS Graphics and SAS/GRAPH. ODS Graphics and GTL produce graphics through the Output Delivery System (ODS) using a template-based system. SAS/GRAPH produces graphics using a device-based system. You can use both systems to generate your graphical output. That is, you can use SAS/GRAPH to generate the output for some jobs, and ODS Graphics to generate the output for others. To help you understand the differences between the two systems in that case, here is a comparison:

- GTL does not produce GRSEGs or use device drivers. The output format that is produced is specified by the OUTPUTFMT= option in the ODS GRAPHICS statement. GTL produces all output in industry standard output formats such as PNG, GIF, JPEG, WMF, TIFF, PDF, EMF, PS, PCL, and SVG. Most SAS/GRAPH procedures produce a GRSEG entry in a SAS catalog. Other output formats in SAS/GRAPH, such as an image or metagraphics file, can be created by selecting an appropriate device driver such as PNG, JPEG, or GIF.
- GTL has a layout-centric architecture. Each graph contains components such as plots, insets, and legends that can be combined in flexible ways inside layout containers to 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.
- The GTL 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.
- The GTL statements and options provide control over the visual properties of a graph. The SAS/GRAPH global statements such as GOPTIONS, AXIS, LEGEND, PATTERN, SYMBOL, and NOTE control the properties for text, markers, and lines.

- GTL controls the size, format, and name of output images with the HEIGHT=, WIDTH=, OUTPUTFMT=, and IMAGENAME= options in the ODS GRAPHICS statement. The ODS GRAPHICS statement is similar in purpose to the GOPTIONS statement. SAS/GRAPH controls the size and format of graphical output with options such as HSIZE=, VSIZE=, and DEVICE= in the GOPTIONS statement.
- The GTL axes, backgrounds, titles, legends, and the other graph components are managed by the layout containers and do not belong to an individual plot.
- Titles and footnotes produced by the SAS TITLE and FOOTNOTE statements do not appear in graphs that are generated using GTL. GTL has its own statements for producing titles and footnotes. (However, the SGPLOT, SGPANEL, and SGSCATTER procedures support the TITLE and FOOTNOTE statements. They generate GTL behind the scenes.)
- The GTL plot type is determined by the plot statement. A plot statement is provided for each plot type. The SAS/GRAPH plot type is determined by global options for some graphs. For example, the INTERPOL= option in the SYMBOL statement might determine whether a graph is a scatter plot or a box plot.
- The GTL graphical attributes for markers, lines, color, and so on, are derived by default from the active ODS style, which cannot be turned off. SAS/GRAPH also uses ODS styles by default. However, with SAS/GRAPH, the style can be turned off and the appearance information that is specified in the device entries used instead. For more information about ODS styles, see “Using ODS Styles to Control Graph Appearance” on page 495.
- GTL supports all of the ODS destinations. For the LISTING destination, an image node is created for the graph in the Results tree. To view the graph, it must be manually opened in an external viewer or in the ODS Graphics Editor. SAS/GRAPH also supports all of the ODS destinations. However, for the LISTING destination, SAS/GRAPH creates a GRSEG node in the Results tree, and the image appears in the graph window automatically.
- GTL supports scaling of fonts and markers by default. This means that the sizes of fonts and markers are adjusted as appropriate to the size of your graph. Font and marker scaling is disabled by the NOSCALE option in the ODS GRAPHICS statement. SAS/GRAPH does not support scaling of fonts and markers.
- GTL does not support the SAS/GRAPH Annotate facility. The GTL annotation facility or the GTL draw statements can be used to add a variety of data-driven or non-data-driven graphical elements to graphs. For information about using the GTL annotation facility, see Chapter 23, “Adding Data-Driven Annotations to Your Graph,” on page 439. For information about using the GTL draw statements, see Chapter 22, “Adding Code-Driven Graphics Elements to Your Graph,” on page 425.

  You can also use the ODS Graphics Editor to add annotations to your graphs. See *SAS ODS Graphics Editor: User’s Guide*.
- GTL does not support RUN-group processing. SAS/GRAPH supports RUN-group processing for some procedures.