# Contents

*About This Book* ................................................................. ix

**PART 1**  Introduction to the Graph Template Language  1

*Chapter 1* • *About the Graph Template Language* ................................................................. 3
   What is the Graph Template Language? ................................................................. 3
   About ODS Graphics ......................................................................................... 3
   When You Need to Use the Graph Template Language ........................................... 8
   Overview of How a Graph Is Generated By Using the GTL ....................................... 8

*Chapter 2* • *Summary of the Graph Template Language Statements and Features* ......... 11
   Introduction to GTL Statements ............................................................................. 11
   Categories of Statements ..................................................................................... 13
   Common Features Supported by the Plot, Layout, Legend, and Text Statements ........ 18

*Chapter 3* • *Getting Started with the Graph Template Language* ............................... 27
   Creating the Graph Template .............................................................................. 27
   Compiling and Executing the Template .................................................................. 30
   Managing the Graphical Output ............................................................................ 32
   Example: Building a Model Fit Plot ....................................................................... 35
   Examples and Resources on the Web ..................................................................... 44

**PART 2**  Plots  47

*Chapter 4* • *Gallery of the GTL Plots* ................................................................. 49
   Gallery of the GTL Plots ..................................................................................... 49

*Chapter 5* • *How the GTL Plot Statements Are Categorized* ................................. 61
   Plot Types ........................................................................................................... 61
   Plot Categories .................................................................................................... 63
   Additional Plot Category: the Primary Plot .......................................................... 63

*Chapter 6* • *Plot Statements by Category* .............................................................. 65
   Stand-alone, 2-D, Computed Plots ......................................................................... 65
   Stand-alone, 2-D, Parameterized Plots .................................................................. 67
   Stand-alone, 3-D, Parameterized Plots .................................................................. 69
   Dependent Plots ..................................................................................................... 69

**PART 3**  Layouts  71

*Chapter 7* • *Overview of the Graph Template Language Layouts* ............................... 73
   Layouts in the GTL ............................................................................................... 73
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Creating Overlay Graphs Using the OVERLAY Layout</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>The LAYOUT OVERLAY Statement</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Statements You Can Use in an OVERLAY Block</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>Restrictions on Allowed Statements</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>Restrictions on Statement Combinations</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>Statement Order</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>Managing Axes in OVERLAY Layout</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>Avoiding Plot Data Conflicts</td>
<td>144</td>
</tr>
<tr>
<td></td>
<td>Overlay Examples</td>
<td>145</td>
</tr>
<tr>
<td>9</td>
<td>Creating Overlay Graphs with Equated Axes Using the OVERLAYEQUATED Layout</td>
<td>157</td>
</tr>
<tr>
<td></td>
<td>The LAYOUT OVERLAYEQUATED Statement</td>
<td>157</td>
</tr>
<tr>
<td></td>
<td>Managing Axes in OVERLAYEQUATED Layout</td>
<td>159</td>
</tr>
<tr>
<td></td>
<td>Equated Overlay Layout Examples</td>
<td>162</td>
</tr>
<tr>
<td>10</td>
<td>Creating Overlay 3-D Graphs Using the OVERLAY3D Layout</td>
<td>167</td>
</tr>
<tr>
<td></td>
<td>The LAYOUT OVERLAY3D Statement</td>
<td>167</td>
</tr>
<tr>
<td></td>
<td>Data Requirements for 3-D Plots</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td>Managing Axes in OVERLAY3D Layout</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>Display Features of the OVERLAY3D Layout</td>
<td>182</td>
</tr>
<tr>
<td>11</td>
<td>Creating Gridded Graphs Using the GRIDDED Layout</td>
<td>187</td>
</tr>
<tr>
<td></td>
<td>The LAYOUT GRIDDED Statement</td>
<td>187</td>
</tr>
<tr>
<td></td>
<td>Defining a Basic Grid</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>Building a Table of Text</td>
<td>193</td>
</tr>
<tr>
<td></td>
<td>Sizing Issues</td>
<td>194</td>
</tr>
<tr>
<td>12</td>
<td>Creating Lattice Graphs Using the LATTICE Layout</td>
<td>201</td>
</tr>
<tr>
<td></td>
<td>The LAYOUT LATTICE Statement</td>
<td>201</td>
</tr>
<tr>
<td></td>
<td>Defining a Basic Lattice</td>
<td>204</td>
</tr>
<tr>
<td></td>
<td>Managing Axes in LATTICE Layout</td>
<td>210</td>
</tr>
<tr>
<td></td>
<td>Adjusting the Sizes of Rows and Columns</td>
<td>216</td>
</tr>
<tr>
<td></td>
<td>Adjusting the Graph Size</td>
<td>218</td>
</tr>
<tr>
<td></td>
<td>Examples: Lattice Layout</td>
<td>221</td>
</tr>
<tr>
<td>13</td>
<td>Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>About Classification Panels</td>
<td>236</td>
</tr>
<tr>
<td></td>
<td>The LAYOUT DATALATTICE Statement</td>
<td>236</td>
</tr>
<tr>
<td></td>
<td>The LAYOUT DATAPANEL Statement</td>
<td>237</td>
</tr>
<tr>
<td></td>
<td>The LAYOUT PROTOTYPE Statement</td>
<td>238</td>
</tr>
<tr>
<td></td>
<td>Distinction between DATALATTICE and DATAPANEL</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>Organizing Panel Contents</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>Managing Axes in DATALATTICE and DATAPANEL Layouts</td>
<td>247</td>
</tr>
<tr>
<td></td>
<td>Controlling the Classification Headers</td>
<td>251</td>
</tr>
<tr>
<td></td>
<td>Using Sidebars</td>
<td>255</td>
</tr>
<tr>
<td></td>
<td>Controlling the Interactions of Classifiers</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>Using Non-computed Plots in Classification Panels</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>Adding an Inset to Each Cell</td>
<td>271</td>
</tr>
</tbody>
</table>
Chapter 26 • Using Dynamics and Macro Variables in Your Templates ........ 555
  Introduction to Dynamic Variables and Macro Variables ............. 555
  Declaring Dynamic Variables and Macro Variables ................ 555
  Referencing Dynamic Variables and Macro Variables ............... 556
  Initializing Dynamic Variables and Macro Variables ............... 557
  Special Dynamic Variables ..................................... 562

Chapter 27 • Using Conditional Logic and Expressions in Your Templates .... 565
  Constructs Available for Run-time Programming ..................... 565
  Expressions .................................................................. 565
  Conditional Logic ...................................................... 567

Chapter 28 • Using Functions in Your Templates .......................... 573
  Overview ...................................................................... 573
  SAS Functions .......................................................... 573
  Functions Defined Only in GTL .................................... 576
  GTL Summary Statistic Functions ................................. 580

Chapter 29 • Sharing Your Custom Templates ................................ 585
  Creating Shared Templates ........................................... 585

Chapter 30 • Modifying Predefined Templates .............................. 587
  Predefined Templates for SAS Analytical Procedures ............ 587
  Modify a Predefined Template ...................................... 589

PART 9  Appendixes 591

Appendix 1 • Reserved Keywords and Unicode Values .................... 593

Appendix 2 • Graph Style Elements Used by ODS Graphics ........... 597
  About the Graphical Style Elements ................................ 597
  General Graph Appearance Style Elements ........................ 597
  Graphical Data Representation Style Elements (Non-Grouped Data) 600
  Graphical Data Representation Style Elements (Grouped Data) .... 604
  Display Style Elements .............................................. 605

Appendix 3 • Display Attributes ......................................... 607
  General Syntax for Attribute Options ................................ 607
  Attributes Available for the Attribute Options ...................... 608
  Available Line Patterns ............................................... 612

Appendix 4 • Predefined Colors ........................................... 613

Appendix 5 • Tick Value Fit Policy Applicability .......................... 627

Appendix 6 • SAS Formats Not Supported ................................. 631
  Using SAS Formats .................................................... 631
  Unsupported Numeric Formats ....................................... 631
  Unsupported Date and Time Formats Related to ISO 8601 ............ 632
  Other Unsupported Date and Time Formats .......................... 632
  Unsupported Currency Formats ...................................... 633
Unsupported User-Defined Formats ........................................ 633

**Appendix 7 • Memory Management for ODS Graphics** .......................... 635
  SAS Options Affecting Memory ............................................. 635
  Managing a Java Out of Memory Error ................................... 635

**Appendix 8 • Understanding Hexadecimal Values** ............................... 639

**Appendix 9 • ODS Graphics and SAS/GRAPH** .................................. 643

  **Recommended Reading** .................................................... 645
  **Glossary** ................................................................. 647
  **Index** ................................................................. 659
### About This Book

### Using This Book

#### Prerequisites

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke the SAS System at your site</td>
<td>Instructions provided by the on-site SAS support personnel</td>
</tr>
<tr>
<td>Use Base SAS software</td>
<td>Base documentation library:</td>
</tr>
<tr>
<td>Use the DATA step to create and manipulate SAS data sets</td>
<td>• <em>SAS Language Reference: Concepts</em></td>
</tr>
<tr>
<td>Use the SAS windowing environment or SAS Enterprise Guide to enter, edit, and submit program code</td>
<td>• <em>SAS Data Set Options: Reference</em></td>
</tr>
<tr>
<td></td>
<td>• <em>SAS Formats and Informats: Reference</em></td>
</tr>
<tr>
<td></td>
<td>• <em>SAS Functions and CALL Routines: Reference</em></td>
</tr>
<tr>
<td></td>
<td>• <em>SAS Statements: Reference</em></td>
</tr>
<tr>
<td></td>
<td>• <em>SAS System Options: Reference</em></td>
</tr>
<tr>
<td></td>
<td>• <em>Base SAS Utilities: Reference</em></td>
</tr>
<tr>
<td>Allocate SAS libraries and assign librefs</td>
<td>Documentation for using the SAS System in your operating environment:</td>
</tr>
<tr>
<td>Create external files and assign filerefs</td>
<td>• <em>SAS Companion for Windows</em></td>
</tr>
<tr>
<td></td>
<td>• <em>SAS Companion for UNIX Environments</em></td>
</tr>
<tr>
<td></td>
<td>• <em>SAS Companion for z/OS</em></td>
</tr>
<tr>
<td>Manipulate SAS data sets by using SAS procedures</td>
<td><em>Base SAS Procedures Guide</em></td>
</tr>
</tbody>
</table>
Conventions

Syntax Conventions

Particular fonts have special meanings when they are used in the presentation of GTL syntax in this document. For example, items presented in italics identify arguments or values that you supply. Angle brackets (< >) indicate optional arguments. The conventions used in this document are the same conventions used in Base SAS documentation. For a complete explanation, see the Base SAS documentation listed in “Prerequisites” on page ix.

Value-Type Notation

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

boolean

specifies a string value that is true or false, or a style reference to a Boolean value.

For all attributes that support a value of ON, the following forms are equivalent:

ATTRIBUTE-NAME
ATTRIBUTE-NAME=ON

For all attributes that support a value of column, column can be any variable that you declare in the GTL template with a DYNAMIC, MVAR, or NMVAR statement. If the attribute is a Boolean value, then the value of variable should resolve to either true or false. The following values represent a true or false value.

<table>
<thead>
<tr>
<th>Values for True</th>
<th>Values for False</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>OFF</td>
</tr>
<tr>
<td><em>ON</em></td>
<td><em>OFF</em></td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td><em>YES</em></td>
<td><em>NO</em></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: The values are not case-sensitive.

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. These names include
  - a predefined SAS color (for example, blue or V1YG)
  - a red/green/blue (RGB) value (for example, CX0023FF or #0023FF)
  - a hue/light/saturation (HLS) value (for example, H14E162D)
• a gray-scale value (for example, GRAYBB).
• 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, integer-column
• number, numeric-column
• string, string-column

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

<table>
<thead>
<tr>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM</td>
<td>centimeters</td>
</tr>
<tr>
<td>IN</td>
<td>inches</td>
</tr>
<tr>
<td>MM</td>
<td>millimeters</td>
</tr>
<tr>
<td>PCT or %</td>
<td>percentage</td>
</tr>
<tr>
<td>PT</td>
<td>point size (72 points = 1 inch)</td>
</tr>
<tr>
<td>PX</td>
<td>pixels</td>
</tr>
</tbody>
</table>

The default value for the Printer destination is units of 1/150 of an inch.

**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:
\[ x = \text{EVAL(expression)} \]

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:

\[ \text{EVAL(MEAN(Time) + ACF(Time, NLags=10))} \]

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

**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 609.

**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 610.

**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 the GTL, a space enclosed in quotation marks (" ") and empty quotation marks ("") 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 (" ").

**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 \textit{style-element:attribute}, 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:

\begin{verbatim}
scatterplot x=X y=Y /
    markersymbol=GraphDataDefault:markersymbol
    markercolor=GraphDataDefault:contrastcolor
    markersize=GraphDataDefault:markersize
\end{verbatim}

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.

\textbf{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 for 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. In the SAS windowing
environment, the default ODS destination is ODS HTML. For information about the
default ODS output in SAS Studio, see “SAS Studio and ODS” in \textit{SAS Output Delivery
Guide}.

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. Chapters Chapter 23, “Managing Your Graph’s Appearance,” on
page 445 and Chapter 24, “Managing Your Graphics Output,” on page 519 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.
About This Book
Part 1

Introduction to the Graph Template Language

Chapter 1
About the Graph Template Language

Chapter 2
Summary of the Graph Template Language
Statements and Features

Chapter 3
Getting Started with the Graph Template Language
Chapter 1

About 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 illustrating often used features. You also need access to the SAS Graph Template Language: Reference, which is the language dictionary for GTL.

About ODS Graphics

A Quick Overview of ODS Graphics

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

- flexible syntax to create complex graphs
• high-quality graphical output

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

ODS Graphics is driven by the GTL, which provides the power and flexibility to create many complex graphs. Standard GTL templates that you can use to generate graphics output for the SAS analytical procedures are delivered with SAS. You can modify these templates in order to customize the appearance of the graphics output of these procedures. You can also create your own templates for creating custom graphics. Although you can use GTL to modify the SAS analytical procedure graphics output or to create custom graphics, its power and flexibility come with some complexity. For that reason, this document discusses the ways in which the SAS System leverages the power of GTL to create graphics by using other tools and systems. You might find that these other tools meet all of your needs.

For more information about ODS Graphics, see *SAS ODS Graphics: Getting Started with Business and Statistical Graphics*.

**SAS Statistical Graphics Procedures**

You can obtain analytical graphs automatically from SAS analytical procedures. Furthermore, you can edit or customize these graphs with the ODS Graphics Editor, all without any need to learn the GTL syntax. However, frequently you might need to get a better understanding of the data in a study or survey by creating preliminary graphical views of the data. Such views might be necessary before a decision can be made about the detailed analysis process. Also, your task might require data analysis using multiple procedures and some custom data management. After such analysis process, the results contained in the output data sets might need to be displayed as custom graphs.

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

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

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

For more information about the SG procedures, see the *SAS ODS Graphics: Procedures Guide*

**Automatic Graphics from SAS Analytical Procedures**

The SAS analytical procedures that support ODS Graphics can produce graphs automatically when ODS Graphics is enabled. (See *Table 1.1 on page 6.*) When ODS Graphics is enabled, the graphs defined as part of any procedure's output are written to the active ODS destinations. You can control the specific graphs that are produced by using the PLOTS= options on the procedure statement or by using the ODS SELECT and ODS EXCLUDE statements. When ODS Graphics is disabled, these procedures do no produce graphs.
In SAS 9.4 and later releases, ODS Graphics is enabled by default. To disable automatic graphics, use the following statement:

```sas
ods graphics off </options>;
```

To re-enable automatic graphics, use the following statement:

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

The procedures that support ODS graphics are listed in the following table.

**Table 1.1 SAS Analytical Procedures That Support ODS Graphics**

<table>
<thead>
<tr>
<th>Product</th>
<th>Procedure Name</th>
<th>Procedure Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FREQ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UNIVARIATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIMILARITY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AUTOREG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENTROPY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ESM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXPAND</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MODEL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PANEL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSLIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIMESERIES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UCM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARMAX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPFENGINE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPABILITY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CUSUM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MACONTROL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PARETO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RELIABILITY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHEWHART</td>
<td></td>
</tr>
<tr>
<td>Dimensions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The ODS Graphics Designer

If you prefer to create custom graphs without having to know the details of GTL, you can use the ODS Graphics Designer. The SAS ODS Graphics Designer is a graphical application that enables you to design and create custom graphs interactively using its point-and-click graphical user interface. The ODS Graphics Designer is based on GTL, which is used by the SAS analytical procedures and the SAS ODS Graphics procedures. However, you do not need to know the details of GTL to create graphs using the ODS Graphics Designer.

Using the ODS Graphics Designer, you can design sophisticated graphs using a wide array of plot types. You can design multi-cell graphs, classification panels, and scatter plot matrices. You can add titles, footnotes, legends, and other graphics elements to your graphs. You can save your graphs as image files for inclusion in other documents, or as ODS Graphics Designer (SGD) files that you can edit later using the ODS Graphics Editor.


The ODS Graphics Editor

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

• Edit or add titles and footnotes to the graphs.

• Change graph styles and visual attributes, such as marker shapes, line patterns, colors, and so on.
• Add free-form text, arrows, lines, and other graph elements to call out various elements of the results.

Changes made to a graph with the ODS Graphics Editor do not affect the template that defined the graph. For more information about the ODS Graphics Editor, see the SAS ODS Graphics Editor: User's Guide

---

When You Need to Use the Graph Template Language

You might need to use the GTL in the following cases:

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

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

---

Overview of How a Graph Is Generated By Using the GTL

Steps for Generating a Graph

Creating a graph by using the GTL involves the following steps:

1. Define the structure of the graph by using the GTL syntax in a DEFINE STATGRAPH block in a TEMPLATE procedure statement.

2. Run the TEMPLATE procedure to compile and save the template definition.

3. Run an SGRENDER procedure statement that specifies the template and graph data set by name in order to generate the graph.

Structure of a Basic 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 figure shows the basic structure of a GTL template. Call outs identify the components.
Figure 1.1  A Basic Template Definition

1. the PROC TEMPLATE statement, which compiles and saves an ODS template.
2. the DEFINE statement, which 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, which starts the graph statement block. The
   ENDGRAPH statement terminates the graph statement block. GTL statements
   are placed inside the graph statement block.
4. GTL statements, which define the graph layout and generate the graph output.

Templates are described in more detail later in this guide.

Compiling and Saving the Template

When a GTL template is executed in SAS, it is compiled and stored for you
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, for example, use the
following PROC TEMPLATE statement:

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 PROC TEMPLATE statement:

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

You can also use the SAS windowing environment to see a list of the templates and to
view their contents. For more information, see SAS Output Delivery System: User's
Guide.
Creating the Graph Using the SGRENDER Procedure

The SGRENDER procedure generates a graph from a graph template. The following SGRENDER statement renders the graph that is defined in Figure 1.1 on page 9, using the currently open ODS output destination.

```sas
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 command shown in the following example:

```sas
ods path show;
```

```
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.

Chapter 2
Summary of the Graph Template Language Statements and Features

Introduction to GTL Statements ........................................... 11
Statements .................................................................. 11
Blocks .................................................................. 12
Categories of Statements .................................................. 13
Overview of GTL Statement Categories .............................. 13
Plot Statements ............................................................. 13
Legend Statements .......................................................... 13
Text Statements ............................................................. 14
Layout Statements ........................................................... 15
Draw Statements ............................................................. 16
Annotation Statements ..................................................... 17
Common Features Supported by the Plot, Layout, Legend,
and Text Statements ........................................................ 18
Features Supported by Many Plot Statements ......................... 18
Features Supported by Layout, Legend, and Text Statements .... 23

Introduction to GTL Statements

The GTL encompasses a large number of statements and options. Just as you can think of the SAS language syntax in terms of statements, functions, formats, and system options, you can apply a classification scheme to GTL. The primary classifications for the GTL syntax are statements and blocks. These terms are used throughout this book in describing GTL code.

Statements

All GTL statements have the following syntax:

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

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;

**Blocks**

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

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.
Categories of Statements

Overview of 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

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

Legend Statements

The 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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Required Arguments</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISCRETELEGEND</td>
<td>Name(s) of associated plot(s)</td>
<td>Traditional legend with entries for grouped markers or lines, or overlaid plots.</td>
</tr>
<tr>
<td>CONTINUOUSLEGEND</td>
<td>Name of an associated plot</td>
<td>Shows a numeric scale with a color ramp. Used in conjunction with contours, surfaces, and scatter plots.</td>
</tr>
</tbody>
</table>
The following figure shows a graph with a discrete legend.

![Graph with a discrete legend](image)

For more information, see Chapter 16, “Adding Legends to Your Graph,” on page 309.

**Text Statements**

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Required Arguments</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTRYTITLE</td>
<td>String</td>
<td>Text to appear above graph. The ENTRYTITLE statement is specified inside the BEINGRAPH block but outside of the outermost layout.</td>
</tr>
<tr>
<td>ENTRYFOOTNOTE</td>
<td>String</td>
<td>Text to appear below graph. The ENTRYFOOTNOTE statement is specified inside the BEINGRAPH block but outside of the outermost layout.</td>
</tr>
<tr>
<td>ENTRY</td>
<td>String</td>
<td>Text to appear within graph. The ENTRY statement is specified inside a layout block.</td>
</tr>
</tbody>
</table>
The following figure shows a graph with a text entry.

![Regression Fit Plot](image)

For more information, see Chapter 15, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 291.

**Layout Statements**

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

- **Single-cell layout statements**, which place the plots, legends, and entries in a common region. The statements that are placed within these "overlay" containers are processed in order. Each plot is drawn on top of the previous plot, with the last one drawn on top.

- **Multi-cell layout statements**, which partition the graph region into multiple smaller "cells." Each cell can be populated by an individual plot, an overlay, or a nested multi-cell layout. The layout of the "cells" is determined by the user, or by classification variables.

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

<table>
<thead>
<tr>
<th>Layout (Description)</th>
<th>Graphics Allowed</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVERLAY (Single Cell)</td>
<td>2-D</td>
<td>General purpose layout for superimposing 2-D plots.</td>
</tr>
<tr>
<td>OVERLAYEREQUATED (Single Cell)</td>
<td>2-D</td>
<td>Specialized OVERLAY with equated axes.</td>
</tr>
</tbody>
</table>
### Layout (Description) | Graphics Allowed | Comments
---|---|---
PROTOTYPE (Single Cell) | 2-D | Specialized LAYOUT used only as child layout of DATAPANEL or DATALATTICE. Only 2-D computed plots or 2-D plots without expressions are allowed.
REGION (Single Cell) | 2-D | Specialized LAYOUT used only for graphs that do not have an axis, such as a PIECHART. Only one plot statement can be used at a time, but other statements such as legends, entries and nested layouts can be added.
OVERLAY3D (Single Cell) | 3-D | General purpose 3-D layout for superimposing 3-D plots.
LATTICE (Multi-cell Panel) | 2-D or 3-D | All cells must be predefined. Axes can be shared across columns or rows, and they can be external to the grid. Many grid labeling and alignment features.
GRIDDED (Multi-cell Grid) | 2-D or 3-D | All cells must be predefined. Axes independent for each cell. Very simple multi-cell container.
DATAPANEL (Classification Panel) | 2-D | Displays a panel of similar graphs based on data subsetted by classification variable(s). Number of cells is based on crossings of \( n \) classification variable(s). Only one DATAPANEL statement is allowed per template.
DATALATTICE (Classification Panel) | 2-D | Displays a panel of similar graphs based on data subsetted by classification variable(s). Number of cells is based on crossings of one or two classification variables. Only one DATALATTICE statement is allowed per template.

To learn more about layouts, see Chapter 7, “Overview of the Graph Template Language Layouts,” on page 73.

### Draw Statements

The GTL provides several draw statements that enable you to draw custom graphical elements on your graph. The graphical elements that these statements draw do not change with the graph data. They are strictly code-driven. You can use these statements to draw text, lines and arrows, and various geometric shapes. You can also draw images such as logos that are stored in a file. These statements are most useful for drawing annotations that highlight the non-data portions of your graph and for creating custom graph features such as broken axes.
<table>
<thead>
<tr>
<th>Statements</th>
<th>Required Arguments</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGINPOLYGON</td>
<td>Two constants: the polygon starting-point coordinates</td>
<td>Starts a polygon statement block and specifies the coordinates of the polygon starting point. The polygon block contains DRAW statements that define the polygon segments. The ENDPOLYGON statement terminates the polygon block.</td>
</tr>
<tr>
<td>BEGINPOLYLINE</td>
<td>Two constants: the polyline starting-point coordinates</td>
<td>Starts a polyline statement block and specifies the coordinates of the polyline starting point. The polyline block contains DRAW statements that define the polyline segments. The ENDPOLYLINE statement terminates the polyline block.</td>
</tr>
<tr>
<td>DRAW</td>
<td>Two constants: the segment ending coordinates</td>
<td>Draws a polygon or polyline segment. The segment is drawn from the polygon or polyline starting point to the ending point, or from the ending point of the last segment drawn to the ending point.</td>
</tr>
<tr>
<td>DRAWARROW</td>
<td>Four constants: the starting and ending coordinates</td>
<td>Draws an arrow between starting and ending coordinates. You can control the arrow direction, and the arrowhead size and shape by using options.</td>
</tr>
<tr>
<td>DRAWIMAGE</td>
<td>A quoted string</td>
<td>Draws an image that is stored in a file on the local file system.</td>
</tr>
<tr>
<td>DRAWLINE</td>
<td>Four constants: the starting and ending coordinates</td>
<td>Draws a line between the starting and ending coordinates.</td>
</tr>
<tr>
<td>DRAWOVAL</td>
<td>Four constants: the center-point coordinates, width, and height</td>
<td>Draws an oval.</td>
</tr>
<tr>
<td>DRAWRECTANGLE</td>
<td>Four constants: the center-point coordinates, width, and height</td>
<td>Draws a rectangle.</td>
</tr>
<tr>
<td>DRAWTEXT</td>
<td>A quoted string</td>
<td>Draws a text string.</td>
</tr>
</tbody>
</table>

For information about the GTL draw statements, see Chapter 18, “Adding Code-Driven Graphics Elements to Your Graph,” on page 393.

**Annotation Statements**

The GTL provides a facility that enables you to draw graphical elements on your graph by using annotation instructions that are stored in a SAS data set. The annotations can vary with the data. You can use annotation instructions in the annotation data set to draw
text, lines and arrows, and various geometric shapes. You can also draw images such as
logos that are stored in a file. The annotations that are drawn by this facility are most
useful for highlighting the actual data in your graphs. To use this facility with the GTL,
you must do the following:

• Create a data set that contains the annotation instructions. Macros are available that
  simplify this process.

• Include at least one ANNOTATE statement in your graph template.

• Use the SGANNO= option in the SGRENDER statement to specify the name of the
  annotation data set when you render the graph.

For more information about how to use the annotation facility in the GTL, see
Chapter 19, “Adding Data-Driven Annotations to Your Graph,” on page 405.

Common Features Supported by the Plot, Layout,
Legend, and Text 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY=(feature ...)</td>
<td>Specifies the plot features to be displayed. Features are plot specific.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARKERATTRS=(marker-options )</td>
<td>Specifies the symbol, size, color, and weight of markers.</td>
</tr>
<tr>
<td>LINEATTRS=(line-options )</td>
<td>Specifies the pattern, thickness, and color of lines.</td>
</tr>
<tr>
<td>TEXTATTRS=(text-options )</td>
<td>Specifies the text color, font, font size, font weight, and font style.</td>
</tr>
<tr>
<td>FILLATTRS=(fill-options )</td>
<td>Specifies the fill color and transparency.</td>
</tr>
<tr>
<td>DATASKIN=(data-skin-name)</td>
<td>Specifies a skin to be applied to filled areas of a plot. See “Using Data Skins” on page 508.</td>
</tr>
</tbody>
</table>
**Plot Transparency**

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATATRANSPARENCY= number</td>
<td>Specifies the degree of transparency. Default is 0 (fully opaque). 1 is fully transparent.</td>
</tr>
</tbody>
</table>

For more information, see “Using Transparency” on page 506.

**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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME= &quot;string&quot;</td>
<td>Specifies a unique name for a plot in order to associate it with another statement.</td>
</tr>
<tr>
<td>LEGENDLABEL= &quot;string&quot;</td>
<td>Specifies a description of a plot to appear in a legend.</td>
</tr>
</tbody>
</table>

For more information about legends, see Chapter 16, “Adding Legends to Your Graph,” on page 309.

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 Statement” 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATALABEL= column</td>
<td>Specifies a column to label data points in a scatter plot, series plot, needle plot, step plot, or vector plot.</td>
</tr>
<tr>
<td>DATALABELATTRS= text-properties</td>
<td>Specifies text properties for data labels.</td>
</tr>
<tr>
<td>DATALABELPOSITION=TOPRIGHT</td>
<td>TOP</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CURVELABEL=&quot;string&quot;</td>
<td>column</td>
</tr>
<tr>
<td>CURVELABELUPPER=&quot;string&quot;</td>
<td>column</td>
</tr>
<tr>
<td>CURVELABELLOWER=&quot;string&quot;</td>
<td>column</td>
</tr>
<tr>
<td>CURVELABELATTRS= text- properties</td>
<td>Specifies text properties for curve label(s).</td>
</tr>
<tr>
<td>CURVELABELLOCATION= INSIDE</td>
<td>OUTSIDE</td>
</tr>
<tr>
<td>Note: The CURVELABELLOCATION=OUTSIDE option is not supported in the LATTICE, DATALATTICE, and DATAPANEL layouts.</td>
<td></td>
</tr>
<tr>
<td>CURVELABELPOSITION=</td>
<td>Specifies positioning options for the curve label(s).</td>
</tr>
</tbody>
</table>

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

![Regression Fit Plot](image)

For more information, see Chapter 15, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 291.

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.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarLabelFitPolicy=Auto</td>
<td>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.</td>
</tr>
<tr>
<td>DatalabelsSplit=TRUE</td>
<td>Specifies whether the data labels should be split into multiple lines, if necessary, to fit the label in the available space.</td>
</tr>
<tr>
<td>DatalabelsSplit=FALSE</td>
<td></td>
</tr>
<tr>
<td>DatalabelsSplitChar=&quot;character-list&quot;</td>
<td>Specifies one or more characters on which the data labels can be split.</td>
</tr>
<tr>
<td>DatalabelsSplitCharDrop=TRUE</td>
<td>Specifies whether the split characters on which a split occurs are removed from the displayed data label.</td>
</tr>
<tr>
<td>DatalabelsSplitJustify=&quot;justification&quot;</td>
<td>Specifies how each line of the split data labels is to be justified.</td>
</tr>
<tr>
<td>CurvelabelsSplit=TRUE</td>
<td>Specifies whether the curve labels should be split into multiple lines, if necessary, to fit the label in the available space.</td>
</tr>
<tr>
<td>CurvelabelsSplit=FALSE</td>
<td></td>
</tr>
<tr>
<td>CurvelabelsSplitChar=&quot;character-list&quot;</td>
<td>Specifies one or more characters on which the curve labels can be split.</td>
</tr>
<tr>
<td>CurvelabelsSplitCharDrop=TRUE</td>
<td>Specifies whether the split characters on which a split occurs are removed from the displayed curve label.</td>
</tr>
<tr>
<td>CurvelabelsSplitJustify=&quot;justification&quot;</td>
<td>Specifies how each line of the split curve labels is to be justified.</td>
</tr>
<tr>
<td>ValueFitPolicy=&quot;fit-policy&quot;</td>
<td>Specifies how block plot text values are adjusted to fit within the containing block.</td>
</tr>
<tr>
<td>ValueSplitChar=&quot;character-list&quot;</td>
<td>Specifies one or more characters on which the block labels can be split when value splitting is used in a block plot.</td>
</tr>
<tr>
<td>ValueSplitCharDrop=TRUE</td>
<td>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.</td>
</tr>
<tr>
<td>ValueSplitCharDrop=FALSE</td>
<td></td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LABELFITPOLICY=AUTO</td>
<td>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 label is always split into multiple lines regardless of the available space.</td>
</tr>
<tr>
<td>LABELSPLITCHAR=&quot;character-list&quot;</td>
<td>Specifies one or more characters on which the axis label can be split when the SPLIT or SPLITALWAYS fit policy is used.</td>
</tr>
<tr>
<td>LABELSPLITCHARDROP=TRUE</td>
<td>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.</td>
</tr>
<tr>
<td>LABELSPLITJUSTIFY=&quot;justification&quot;</td>
<td>Specifies how each line of the split axis label is to be justified when the SPLIT or SPLITALWAYS fit policy is used.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP= column</td>
<td>Specifies a group variable, always treated as having discrete values. For an example use, see the example for “Legend Statements” on page 13.</td>
</tr>
<tr>
<td>INDEX= positive-integer-column</td>
<td>Specifies an integer column that associates each distinct data value with a predefined graphical style element GraphData1, GraphData2, …</td>
</tr>
<tr>
<td>GROUPDISPLAY=STACK</td>
<td>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.</td>
</tr>
</tbody>
</table>

For more information about using groups, see Chapter 23, “Managing Your Graph’s Appearance,” on page 445.

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

For more information, see “Axis Management” on page 76.

**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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROLENAME= (role=column …)</td>
<td>Creates additional roles to customize data tips.</td>
</tr>
<tr>
<td>TIP= (role-names)</td>
<td>Specifies which plot roles are used for data tips.</td>
</tr>
<tr>
<td>TIPFORMAT=(role=format …)</td>
<td>Specifies a format to be applied to the data for a plot role.</td>
</tr>
<tr>
<td>TIPLABEL=(role=&quot;string&quot; …)</td>
<td>Specifies a label to be applied to the column for a plot role.</td>
</tr>
</tbody>
</table>

For more information and an example, see Chapter 20, “Adding Data Tips to Your Graph,” on page 429.

**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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPAQUE= FALSE</td>
<td>Specifies whether the background is transparent. By default, OPAQUE=FALSE</td>
</tr>
<tr>
<td>BACKGROUNDCOLOR= color</td>
<td>Specifies a color for the background when the background is opaque.</td>
</tr>
</tbody>
</table>
## Borders

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BORDER= FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>BORDERATTRS= ( line-options )</td>
<td>Specifies the properties of the border when a border is displayed.</td>
</tr>
</tbody>
</table>

## Padding

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAD= number</td>
<td>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.</td>
</tr>
<tr>
<td>PAD=(&lt;TOP=number&gt; &lt;BOTTOM=number&gt; &lt;LEFT=number&gt; &lt;RIGHT=number&gt;)</td>
<td></td>
</tr>
<tr>
<td>OUTERPAD= AUTO</td>
<td>number</td>
</tr>
<tr>
<td>OUTERPAD=(&lt;TOP=number&gt; &lt;BOTTOM=number&gt; &lt;LEFT=number&gt; &lt;RIGHT=number&gt;)</td>
<td></td>
</tr>
</tbody>
</table>

## Test Position

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HALIGN= LEFT</td>
<td>CENTER</td>
</tr>
<tr>
<td>VALIGN= TOP</td>
<td>CENTER</td>
</tr>
</tbody>
</table>

24 Chapter 2 • Summary of the Graph Template Language Statements and Features
## Legend and Layout Positions

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| HALIGN= LEFT | CENTER | RIGHT | 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 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 |
Chapter 3
Getting Started with the Graph Template Language

Creating the Graph Template

Creating the Graph Template ................................................................. 27
Compiling and Executing the Template .................................................. 30
Compiling the Template ................................................................. 30
Executing the Template to Produce the Graph ...................................... 31
Managing the Graphical Output .......................................................... 32
Directing Output to ODS Destinations .................................................. 32
Modifying Graph Appearance with Styles .......................................... 33
Controlling Physical Aspects of the Output .......................................... 34
Example: Building a Model Fit Plot ......................................................... 35
Overview of This Example .................................................................... 35
Create the Basic Scatter Plot .................................................................. 36
Add the Regression Plot ....................................................................... 38
Add the Confidence Limit Bands .......................................................... 39
Add the Legend and the Observation Count ......................................... 42
Enable the Axis Grid Lines .................................................................... 44
Examples and Resources on the Web ..................................................... 44

Creating the Graph Template

As described in “Steps for Generating a Graph” on page 8, the first step in creating a graph by using the GTL is to define the graph template. To illustrate this step, assume that you want to produce the following graph. This graph shows a linear regression fit for a set of data in which Height is an independent variable and Weight is a dependent variable.
Here is the SAS code that defines the template for this graph. As you can see, this template conforms to the basic structure described in “Overview of How a Graph Is Generated By Using the GTL” on page 8.

**Example Code 3.1  Regression Fit Plot Template**

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

The TEMPLATE procedure can produce different types of templates, such as STYLE, TABLE, COLUMN, and STATGRAPH. The type of template to be created is specified with a DEFINE block. The DEFINE STATGRAPH statement and its matching END statement indicate that a graphics template named MODELFIT is to be created.

The template name can be a simple one-level name or a multi-level name such as GRAPHS.MODELFIT or PROJECT.STUDY3.MODELFIT, indicating the folder in which the MODELFIT template is to be stored. The BEGINGRAPH statement and its matching ENDFRAPH statement define the outermost container for the graph.
The BEGINGRAPH statement supports several options that control the graph size and appearance. Within the BEGINGRAPH block, you can use statements that define the content of the graph. ENTRYTITLE and ENTRYFOOTNOTE statements can be used to specify graph title lines and graph footnote lines. The LAYOUT OVERLAY statement and its matching ENDLAYOUT statement define the type of graphical layout to be used.

The OVERLAY layout enables the contained plots to be overlaid. It manages the plot layers and queries all contained plots to decide the axis types, axis labels, and axis ranges. Statements in the layout block define the content of the graph.

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

For more information about the types of layouts and plots in GTL, see “Categories of Statements” on page 13.
Compiling and Executing the Template

Compiling the Template

When you submit the PROC TEMPLATE statement shown in Example Code 3.1 on page 28, the template syntax is checked, and the results are sent to the SAS log. Here is an example of the template being compiled in the SAS windowing environment.

```
12   proc template;
13     define statgraph modelfit;
14       begingraph;
15         entrytitle "Regression Fit Plot";
16         layout overlay;
17           scatterplot x=height y=weight;
18           regressionplot x=height y=weight;
19         endlayout;
20       endgraph;
21     end;
NOTE: STATGRAPH 'Modelfit' has been saved to: SASUSER.TEMPLAT
22   run;
NOTE: PROCEDURE TEMPLATE used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

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

Note:

- The STATGRAPH template syntax requires that you specify necessary arguments (X= and Y= arguments are required for both the SCATTERPLOT and REGRESSIONPLOT statements), but no checking for the existence of the assigned variables is performed at compile time.
- No reference to an input data set appears in the template.
- In SAS Studio, the compiled template is stored in Work.Templat.

When you compile the template, no graph is produced. Instead, a compiled template is created that you can execute to produce a graph.

To list the templates that have been stored in the Sasuser.Templat item store, for example, use the TEMPLATE procedure as shown in the following example:

```
proc template;
  list / store=sasuser.templat;
run;
```
If you want to see the source for a compiled template, you can use the TEMPLATE procedure to write the template source to the SAS log, as shown in the following example:

```sas
proc template;
    path sasuser.templat;
    source modelfit;
run;
```

Here is a partial listing of the SAS log output:

```
define statgraph Modelfit;
begingraph;
    entrytitle "Regression Fit Plot";
    layout overlay;
        scatterplot x=HEIGHT y=WEIGHT;
        regressionplot x=HEIGHT y=WEIGHT;
    endlayout;
endgraph;
end;
```

You can also use the ODST SAS windowing command or the SAS windowing environment to browse the item stores. For more information about the item stores and the TEMPLATE procedure, see *SAS Output Delivery System: Procedures Guide*.

### Executing the Template to Produce the Graph

To produce a graph, use the SGRENDER procedure as shown in the following example.

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

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

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

This information, along with the GTL definition of the graph, is then passed to a rendering module that assembles everything and produces an image. The image is
integrated into the active ODS destinations. In this example, the graph shown in Figure 3.1 on page 28 is generated.

At least one ODS destination must be open in order to generate output. In the SAS windowing environment, the ODS HTML destination is open by default. For information about the default ODS output in SAS Studio, see “SAS Studio and ODS” in SAS Output Delivery System: User’s Guide. Information about how to manage the output is provided in “Managing the Graphical Output” on page 32.

For more information about other features of the SGRENDER procedure, see Chapter 25, “Executing Graph Templates,” on page 545.

---

Managing the Graphical Output

Directing Output to ODS Destinations

All ODS graphics are generated in industry standard formats (PNG, PDF, and so on), depending on the settings for the active ODS destinations. All ODS destinations such as HTML, PDF, RTF, LATEX, and PRINTER are fully supported. The ODS destinations enable you to do the following:

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

As discussed in “Compiling the Template” on page 30, a compiled template is stored in an item store. Thus, without rewriting or resubmitting the template code, you can render the graph as often as needed during the current SAS session or a future SAS session.

To generate ODS Graphics output for use on the web, you can direct the output to an HTML destination, which generates all of the files that are needed to display the graph in a web browser.

The following code uses the ODS HTML destination to generate a graph for use on the web from the MODELFIT template that was defined in Example Code 3.1 on page 28.

```sas
/* Specify a path for the ODS output */
filename odsout "output-path";
ods _all_ close;
ods html path=odsout (url=none) file="modelfit.html";
proc sgrender data=sashelp.class template=modelfit;
run;
ods html close;
ods html; /* Not required in SAS Studio */
```

Note the following:

- Closing the currently open ODS destinations and opening only the output destination that you need helps preserve system resources.
- The FILENAME= statement and the PATH= option in the ODS HTML statement specify the directory for the output files that are created by the SAS statements, including images from ODS Graphics. You can specify any valid path on your system for the output.
• The FILE= option specifies that SAS output is written to the file `modelfit.html`, which is saved in the folder that is specified in the PATH= option.

• Each ODS destination uses a default style for graphics output ("Modifying Graph Appearance with Styles" on page 33 provides an introduction to ODS styles). The HTML destination, for example, uses the HTMLBlue style by default, which uses a gray background.

• Closing the ODS HTML destination closes the output file.

• Reopening the ODS HTML destination provides an output destination for subsequent SAS programs. You can open a different destination, if you prefer.

  Note: This step is not necessary in SAS Studio.

See Chapter 24, "Managing Your Graphics Output," on page 519 for more information about the ODS destinations and the type of output that results from each destination.

If you are using SAS Studio Basic or SAS Studio Mid-Tier, rather than use the ODS HTML destination, you can download the default output to your local machine. For information about the default ODS output in SAS Studio, see “SAS Studio and ODS” in SAS Output Delivery System: User’s Guide. For information about using SAS Studio, see SAS Studio: User's Guide.

**Modifying Graph Appearance with Styles**

GTL is designed to be totally integrated with ODS styles.

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

Every ODS destination has a style that it uses by default. For the HTML destination, for example, the default style is HTMLBlue. To modify the appearance of the graph, you can change its style by specifying the STYLE= option in the ODS destination statement before running the SGRENDER procedure. Here are some examples using the template in Example Code 3.1 on page 28 and the ODS HTML destination.

```sas
ods html style=analysis;
proc sgrender data=sashelp.class template=modelfit;
run;
```

![Regression Fit Plot](image)

```sas
ods html style=journal;
proc sgrender data=sashelp.class template=modelfit;
run;
```
Controlling Physical Aspects of the Output

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

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

The following ODS GRAPHICS statement sets a 320 pixel width for the graph and names the output image `modelfitgraph` for the template in Example Code 3.1 on page 28:

```sas
ods graphics / width=320px imagename="modelfitgraph";
proc sgrender data=sashelp.class template=modelfit;
run;
ods graphics / reset;
```
Example: Building a Model Fit Plot

Overview of This Example

This example shows you how to use the GTL to construct a model fit plot of weight by height. If you are not familiar with the GTL, review this example to get a good understanding of how to build a basic graph using the GTL. The data in Sashelp.Class is used as the data source for this example. You can view the resulting graph in a web browser window. References to the relevant topics in the GTL documentation are provided. The remainder of this book helps you build on this understanding in order to create more complex graphs.

The model fit plot that you create in this example is a composite graph that consists of a scatter plot, and a regression plot with 95% confidence limits and 95% confidence limits of the mean. Enhancements such as a graph title, axis grid lines, a legend of the confidence bands, and the total number of observations in the source data are also included. The following figure shows the final graph.
To construct this graph, use a stepwise approach:

1. Create a basic scatter plot with a graph title.
2. Add the regression plot.
3. Add the confidence limit bands to the regression plot.
4. Add the legend and observation count.
5. Enable the axis grid lines.

The remaining sections describe each step in detail.

Create the Basic Scatter Plot

To begin construction of the graph, start with a scatter plot of the data and a graph title. Start with the basic template described in “Overview of How a Graph Is Generated By Using the GTL” on page 8. Name this template WEIGHTBYHEIGHT.

```sas
proc template;
   define statgraph weightbyheight;
      begingraph;
      /* GTL statements */
      endgraph;
   end;
run;
```

In this case, you need to add GTL statements that do the following:

- Output a title for the graph.
- Specify a top-level layout for the plots.
- Output a scatter plot in the top-level layout.

The GTL provides the ENTRYTITLE statement, which generates a title from text that you provide. Several options are available that enable you to customize your title. The ENTRYTITLE statement syntax and options are described in detail in “ENTRY Statement” in *SAS Graph Template Language: Reference*. Additional information is provided in Chapter 15, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on
In the template, place the ENTRYTITLE statement after the BEGINGRAPH statement and before the top-level layout statement.

Next, you must specify a top-level layout. The GTL provides several types of layouts from simple single-cell layouts for single graphs to multicell layouts for grids, lattices, and classification panels. “Layout Statements” on page 15 provides a summary of the available layouts. In this case, because you want a single-cell graph that is a composite of multiple plots, use the LAYOUT OVERLAY statement to define the top-level layout for the graph.

```sas
proc template;
   define statgraph weightbyheight;
      begingraph;
      entrytitle "Model Weight by Height";
      layout overlay;
         /* GTL statements */
      endlayout;
      endgraph;
   end;
run;
```

The LAYOUT OVERLAY statement is described in detail in “LAYOUT OVERLAY Statement” in SAS Graph Template Language: Reference. Additional information is provided in Chapter 8, “Creating Overlay Graphs Using the OVERLAY Layout,” on page 79.

Finally, in the LAYOUT OVERLAY block, you must provide a GTL statement that generates a scatter plot. The GTL provides many plot statements that you can use to generate a variety of plot types. Chapter 4, “Gallery of the GTL Plots,” on page 49 provides a gallery of the available plots. To generate the scatter plot, use the SCATTERPLOT statement, which is described in detail in “SCATTERPLOT Statement” in SAS Graph Template Language: Reference.

```sas
proc template;
   define statgraph weightbyheight;
      begingraph;
      entrytitle "Model Weight by Height";
      layout overlay;
         scatterplot x=height y=weight;
      endlayout;
      endgraph;
   end;
run;
```

The SCATTERPLOT statement requires two arguments: an independent variable and a dependent variable. This example uses Height for the independent variable and Weight for the dependent variable. Now you can generate the basic graph. To test your work, submit the PROC TEMPLATE statement to compile the template and save it in Sasuser.Templat. If there are no syntax errors, use the following SGRENDER statement to render the graph:

```sas
proc sgrender data=sashelp.class template=weightbyheight;
run;
```
Add the Regression Plot

To add a regression plot to the graph, add a REGRESSIONPLOT statement before the SCATTERPLOT statement in the LAYOUT OVERLAY block.

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

The REGRESSIONPLOT statement is described in detail in “REGRESSIONPLOT Statement” in *SAS Graph Template Language: Reference*. Like the SCATTERPLOT statement, the REGRESSIONPLOT statement requires a category variable and a response variable. Again, use Height for the category variable and Weight for the response variable.
Add the Confidence Limit Bands

In the GTL, the MODELBAND statement plots the confidence limit bands for an associated smoother plot such as a regression plot. The MODELBAND plot is described in detail in “MODELBAND Statement” in SAS Graph Template Language: Reference.

To plot the bands for the 95% confidence limit and the 95% confidence limit mean for the regression plot:

- Add a MODELBAND statement for the 95% confidence limit band.
- Add a MODELBAND statement for the 95% confidence limit mean band.
- Associate the MODELBAND statements with the REGRESSIONPLOT statement.

The MODELBAND statement requires a confidence name. The names must be unique. Use the name "cli" for the 95% confidence limit band and the name "clm" for the 95% confidence limit mean band. To associate the MODELBAND statements with the REGRESSIONPLOT statement, add the CLI="cli" and CLM="clm" options to the REGRESSIONPLOT statement.

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

The REGRESSIONPLOT statement CLI= and CLM= options are described in “Regression Options” in SAS Graph Template Language: Reference.
The output is not correct. This is a case where overlay order is important. Because the model band plots have significant filled areas, they obscure the underlying plots if they are drawn last. You need to change the overlay order. Since the CLI band has the largest filled area, you need to draw that plot first. The CLM band has the next largest filled area, so you need to overlay that plot onto the CLI band.

```
proc template;
  define statgraph weightbyheight;
  begingraph;
    entrytitle "Model Weight by Height";
    layout overlay;
      modelband "cli";
      modelband "clm";
      regressionplot x=height y=weight / cli="cli" clm="clm";
      scatterplot x=height y=weight;
    endlayout;
  endgraph;
end;
run;
```
This plot is better, but it is still not correct. The 95% confidence limit mean band is not visible. It is actually overlaid onto the CLI band, but because both bands use the same fill color, they are indistinguishable. You need to modify the appearance of the bands to make them distinguishable. Options are available in the MODELBAND statement that enable you to change the band fill color or fill transparency, or to add an outline to the band. For a clean look, use the DATATRANSPARENCY= option to fade the fill color of the underlying CLI band in order to contrast it with the CLM band.

```sas
proc template;
  define statgraph weightbyheight;
  begingraph;
    entrytitle "Model Weight by Height";
    layout overlay;
      modelband "cli" / datatransparency=0.5;
      modelband "clm";
      regressionplot x=height y=weight / cli="cli" clm="clm";
      scatterplot x=height y=weight;
    endlayout;
  endgraph;
end;
run;
```
"Using Transparency" on page 506 provides additional information about using transparency in your graphs.

**Add the Legend and the Observation Count**

The GTL provides statements that generate legends for your graphs and text that displays additional information. The GTL provides two types of legends: discrete and continuous. Chapter 16, “Adding Legends to Your Graph,” on page 309 provides information about legends in the GTL.

For this example, you want to display information about the confidence band colors. Since you want to refer to plots in the legend, use the DISCRETELEGEND statement. The DISCRETELEGEND statement is described in detail in "DISCRETELEGEND and MERGEDLEGEND Statements" in *SAS Graph Template Language: Reference*. The DISCRETELEGEND statement requires the name of one or more plots. The plot name is specified in the plot statement with the NAME= option. The LEGENDLABEL= option can also be included in the plot statement to override the default legend entry label for that plot with a more meaningful label. To add a legend for our confidence band plots, add the NAME= and LEGENDLABEL= options to the MODELBand statements, and then add a DISCRETELEGEND statement that specifies those names. You also need to display the items in a single column, draw a border around the legend, and position it inside the graph area in the lower right hand corner. Options in the DISCRETELEGEND statement provide this output. Here is the code so far.

```sas
proc template;
define statgraph weightbyheight;
begingraph;
entrytitle "Model Weight by Height";
layout overlay;
  modelband "cli" / name="mb1" legendlabel="95% Predict" datatransparency=0.5;
  modelband "clm" / name="mb2" legendlabel="95% Mean";
  regressionplot x=height y=weight / cli="cli" clm="clm";
  scatterplot x=height y=weight;
  discretelegend "mb1" "mb2" / border=true across=1 valign=bottom halign=right location=inside;
endlayout;
end;
```

Here is the output.
In addition to the legend, you want to add text that shows the observation count. The GTL provides statements that enable you to add titles, footnotes, and text entries to your graphs. Chapter 15, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 291 provides information about these statements. To add the string Nobs=nn in the top left corner of the graph area, use the ENTRY statement. The ENTRY statement is described in detail in “ENTRY Statement” in SAS Graph Template Language: Reference. The ENTRY statement requires a quoted string to display. To include the observation count in the string, use the GTL summary statistic function N, which returns the total nonmissing values for the Weight variable in this case. Chapter 28, “Using Functions in Your Templates,” on page 573 describes how to use functions in your GTL statements.

Within an overlay-type layout, the entry text is centered inside the parent layout. Options are available that enable you to change the text position. Position the text in the top left corner of the graph area. Here is the code with the ENTRY statement added.

```sas
proc template;
define statgraph weightbyheight;
begingraph;
  entrytitle "Model Weight by Height";
  layout overlay;
    modelband "cli" / name="mb1" legendlabel="95% Predict"
      datatransparency=0.5;
    modelband "clm" / name="mb2" legendlabel="95% Mean";
    regressionplot x=height y=weight / cli="cli" clm="clm";
    scatterplot x=height y=weight;
    discretelegend "mb1" "mb2" / border=true across=1
      valign=bottom halign=right
      location=inside;
    entry halign=left "Nobs=" eval(strip(put(n(weight),12.0))) /
      valign=top;
  endlayout;
endgraph;
end;
run;
```

Here is the output.
Enable the Axis Grid Lines

The final step in building your graph is to enable grid lines on both axes to improve the graph readability. The LAYOUT OVERLAY statement provides options that enable you to manage the axes in your graphs. “Managing Axes in OVERLAY Layouts” on page 85 provides general information about managing the axes in an overlay layout. “Axis Options for LAYOUT OVERLAY” in SAS Graph Template Language: Reference provides detailed information about all of the options that you can use to manage your axes in an overlay layout. To enable the grid lines for the X and Y axes, specify the GRIDDISPLAY=ON for both axes. In the LAYOUT OVERLAY statement, axis options for the X and Y axes are specified in the XAXISOPTS= and YAXISOPTS= options respectively.

```
proc template;
  define statgraph weightbyheight;
  begingraph;
    entrytitle "Model Weight by Height";
    layout overlay /
      xaxisopts=(griddisplay=on)
      yaxisopts=(griddisplay=on);
    modelband "cli" / name="mb1" legendlabel="95% Predict"
      datatransparency=0.5;
    modelband "clm" / name="mb2" legendlabel="95% Mean";
    regressionplot x=height y=weight / cli="cli" clm="clm";
    scatterplot x=height y=weight;
    discretelegend "mb1" "mb2" / border=true across=1
      valign=bottom halign=right
      location=inside;
    entry halign=left "Nobs= I eval(strip(put(n(weight),12.0)))" /
      valign=top;
  endlayout;
  endgraph;
end;
run;
```

This code results in the final graph shown in Figure 3.2 on page 36.

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 by Sanjay Matange 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.
  
  http://blogs.sas.com/content/graphicallyspeaking/

- The Graphics Samples Output Gallery is a collection of graphs organized by SAS procedure. The graphs link to the source code in SAS Samples & Notes. The gallery is maintained by SAS Technical Support.
  
  http://support.sas.com/sassamples/graphgallery/index.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.
  
  http://support.sas.com/rnd/datavisualization/index.htm

• Samples & SAS Notes contains an abundance of searchable examples. You can browse by topic, search for a particular note, search for a particular technology such as SGPLOT, and conduct other searches.

  http://support.sas.com/notes/index.html

In addition, you can share your questions, suggestions, and experiences related to graphics on the SAS/GRAPH and ODS Graphics community site. See https://communities.sas.com/community/support-communities/sas_graph_and_ods_graphics.
Part 2

Plots

Chapter 4
Gallery of the GTL Plots ........................................ 49

Chapter 5
How the GTL Plot Statements Are Categorized .................. 61

Chapter 6
Plot Statements by Category ....................................... 65
Chapter 4
Gallery of the GTL Plots

The GTL provides a wide variety of plots. Table 4.1 on page 49 lists the basic plots that are available with the GTL and provides an example of each. The statement that generates each plot is also listed. For additional information, see Chapter 5, “How the GTL Plot Statements Are Categorized,” on page 61 and Chapter 6, “Plot Statements by Category,” on page 65.

Table 4.1 Basic Plots

<table>
<thead>
<tr>
<th>Plot Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Band Plot</td>
<td>A band plot is generated by the BANDPLOT statement. 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. For more information, see “BANDPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>Plot Example</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **Bar Chart** | A bar chart is generated by the BARCHART or BARCHARTPARM statement. These statements can draw vertical or horizontal bar charts. The BARCHART statement draws a bar chart from raw data. It computes all of the values that are required to draw the chart. The BARCHARTPARM statement draws a bar chart from pre-summarized data. It provides no data transformation. For more information, see:  
• “Plot Types” on page 61  
• “BARCHART Statement” in SAS Graph Template Language: Reference  
• “BARCHARTPARM Statement” in SAS Graph Template Language: Reference |
| **Block Plot** | A block plot is generated by the BLOCKPLOT statement. 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-value that is also provided in the plot data. For more information, see “BLOCKPLOT Statement” in SAS Graph Template Language: Reference. |
| **Box Plot** | A box plot is generated by the BOXPLOT or BOXPLOTPARM statement. 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 BOXPLOTPARM statement draws a box plot from pre-computed data. It provides no data transformation. For more information, see:  
• “Plot Types” on page 61  
• “BOXPLOT Statement” in SAS Graph Template Language: Reference  
• “BOXPLOTPARM Statement” in SAS Graph Template Language: Reference |
<table>
<thead>
<tr>
<th>Plot Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B-Spline Plot (Penalized)</strong></td>
<td>A penalized B-spline plot is generated by the PBSPLINEPLOT statement. 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. For more information, see “PBSPLINEPLOT Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
<tr>
<td><strong>Bubble Plot</strong></td>
<td>A bubble plot is generated by the BUBBLEPLOT statement. The plot data provides the X and Y values, which specify the location of the center of each bubble. The plot data also provides a size value, which specifies the radius of each bubble. For more information, see “BUBBLEPLOT Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
</tbody>
</table>
| **Confidence Ellipses** | Confidence ellipses are drawn by the ELLIPSE or ELLIPSEPARM statement. The ELLIPSE statement draws the ellipses from the plot data. It computes all of the values needed to draw the ellipses. 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. For more information, see:  
  - “Plot Types” on page 61  
  - “ELLIPSE Statement” in *SAS Graph Template Language: Reference*  
  - “ELLIPSEPARM Statement” in *SAS Graph Template Language: Reference* |
## Plot Example

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Contour Plot</strong></td>
</tr>
<tr>
<td>A contour plot is generated by the CONTOURPLOTPARM statement. 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. For more information, see “CONTOURPLOTPARM Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
</tbody>
</table>

| **Dendrogram** |
| A dendrogram is generated by the DENDROGRAM statement. 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. For more information, see “DENDROGRAM Statement” in *SAS Graph Template Language: Reference*. |

| **Density Plot (NORMAL Distribution)** |
| A univariate probability density curve is generated by the DENSITYPLOT statement. You can specify a NORMAL or KERNEL distribution. For more information, see “DENSITYPLOT Statement” in *SAS Graph Template Language: Reference*. |

| **Fringe Plot (Below Histogram)** |
| A fringe plot is generated by the FRINGEPLOT statement. In a fringe plot, each line represents the location of a raw data value on the X axis. All of the fringe lines are of equal length. For more information, see “FRINGEPLOT Statement” in *SAS Graph Template Language: Reference*. |
A heat map is generated by the HEATMAPPARM statement and, starting with the third maintenance release of SAS 9.4, by the HEATMAP statement. The heat map is a grid of rectangles in which the location of each rectangle is determined by two independent variables, and the color of each rectangle is determined by a third variable. The HEATMAPPARM statement draws a heat map from pre-computed data. The HEATMAP statement draws a heat map from raw input data. It computes all of the values needed to draw the heat map.

For more information, see:
- “Plot Types” on page 61
- “HEATMAPPARM Statement” in SAS Graph Template Language: Reference
- “HEATMAP Statement” in SAS Graph Template Language: Reference

A high-low chart is generated by the HIGHLOWPLOT statement. A high-low chart displays floating vertical or horizontal lines or bars that connect the minimum and maximum response values for each value of a categorical variable. 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.

For more information, see “HIGHLOWPLOT Statement” in SAS Graph Template Language: Reference.

A univariate histogram is generated by the HISTOGRAM or HISTOGRAMPARM statement. The HISTOGRAM statement draws a histogram from raw input data. It computes all of the values needed to draw the histogram. The HISTOGRAMPARM statement draws a histogram from pre-computed data. It provides no data transformation.

For more information, see:
- “Plot Types” on page 61
- “HISTOGRAM Statement” in SAS Graph Template Language: Reference
- “HISTOGRAMPARM Statement” in SAS Graph Template Language: Reference
### Plot Example

<table>
<thead>
<tr>
<th>Plot Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Histogram (3-D Bivariate)</strong>&lt;br&gt;<a href="image1.png">Image</a></td>
<td>A 3-D bivariate histogram is generated by the BIHISTOGRAM3DPARM statement. The plot data provides variables for the X, Y, and Z values. The plot data must be binned by both X and Y, and the Z variable values must not be negative. For more information, see “BIHISTOGRAM3DPARM Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
<tr>
<td><strong>Line Chart</strong>&lt;br&gt;<a href="image2.png">Image</a></td>
<td>A line chart is generated by the LINECHART statement. A line chart consists of straight-line segments that connect consecutive data points along an axis. A line chart is useful for showing trends over time. The area below the line can be filled, as shown in this example. For more information, see “LINECHART Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
<tr>
<td><strong>Loess Plot</strong>&lt;br&gt;<a href="image3.png">Image</a></td>
<td>A loess plot is generated by the LOESSPLOT statement. The LOESSPLOT statement supports only models of one independent and one dependent variable. For more information, see “LOESSPLOT Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
<tr>
<td><strong>Model Band Plot</strong>&lt;br&gt;<a href="image4.png">Image</a></td>
<td>A model band plot is generated by the MODELBAND statement. A model band shows the confidence limits for an associated smoother plot such as a regression plot. The MODELBAND statement must be associated with a smoother statement that specifies a fitted model and a type of confidence level to compute. For more information, see “MODELBAND Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
</tbody>
</table>
### Plot Example

<table>
<thead>
<tr>
<th>Plot Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mosaic Plot</strong></td>
<td>A mosaic plot is generated by the MOSAICPLOTPARM statement from pre-computed categorical data. 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. The MOSAICPLOTPARM statement must be placed in a REGION, GRIDDED, or LATTICE layout. For more information, see “MOSAICPLOTPARM Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
<tr>
<td><strong>Needle Plot</strong></td>
<td>A needle plot is generated by the NEEDLEPLOT statement. In a needle plot, a needle is drawn for each observation in the input data. A needle consists of a point that is connected to a baseline by a vertical line segment. For more information, see “NEEDLEPLOT Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
<tr>
<td><strong>Pie Chart</strong></td>
<td>A pie chart is generated by the PIECHART statement. 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. For more information, see “PIECHART Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
</tbody>
</table>
A polygon plot is generated by the POLYGONPLOT statement. The POLYGONPLOT statement can draw one or more polygons from data that is stored in a data set. The polygons can be filled, outlined, or both. They can also be labeled. The POLYGONPLOT statement is useful for drawing shapes on your graphs that highlight data, outline data boundaries, and so on. This example shows a filled, yellow polygon that highlights the Setosa data in a scatter plot. The polygon is labeled Setosa.

Note: This feature applies to the first maintenance release of SAS 9.4 and to later releases. See “What’s New in SAS 9.4 Graph Template Language” in SAS Graph Template Language: Reference.

For more information, see “POLYGONPLOT Statement” in SAS Graph Template Language: Reference.

A regression plot is generated by the REGRESSIONPLOT statement. The REGRESSIONPLOT statement supports only models of one independent and one dependent variable. In addition to the regression line, the REGRESSIONPLOT statement can compute confidence levels for the fitted line.

For more information, see “REGRESSIONPLOT Statement” in SAS Graph Template Language: Reference.

A scatter plot is generated by the SCATTERPLOT statement. A marker is drawn for each observation that has nonmissing X and Y values. Markers can be symbols or character strings. Symbol markers can be labeled, as shown in this example.

For more information, see “SCATTERPLOT Statement” in SAS Graph Template Language: Reference.
## Plot Example

<table>
<thead>
<tr>
<th>Scatter-Plot Pairwise Matrix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A pairwise matrix of scatter plots is generated by the SCATTERPLOTMATRIX statement.</strong> By default, 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. For more information, see “SCATTERPLOTMATRIX Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Series Plot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A series plot is generated by the SERIESPLOT statement. A series plot consists of line segments that connect observations of input data. It is typically used to show time-dependent data.</strong> For more information, see “SERIESPLOT Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step Plot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A step plot is generated by the STEPPLOT statement. A step plot is a series of horizontal and vertical line segments that connect observations of input data.</strong> For more information, see “STEPPLOT Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
<td></td>
</tr>
</tbody>
</table>
**Plot Example** | **Description**
---|---
![Straight-Line Plot (Point and Slope)](image)

A straight line of a specified slope and intercept point is drawn by the LINEPARM statement. 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.

For more information, see “LINEPARM Statement” in *SAS Graph Template Language: Reference*.

![Surface Plot](image)

A 3-D surface plot is generated by the SURFACEPLOTPARM statement. 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.

For more information, see “SURFACEPLOTPARM Statement” in *SAS Graph Template Language: Reference*.

![Text Plot](image)

A text plot is generated by the TEXTPLOT statement. A text plot displays text strings at the X and Y location in the graph. It is similar to a scatter plot. 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 the second maintenance release of SAS 9.4 and to later releases. See “What’s New in SAS 9.4 Graph Template Language” in *SAS Graph Template Language: Reference*.*

For more information, see “TEXTPLOT Statement” in *SAS Graph Template Language: Reference*.

![Vector Plot](image)

A vector plot is generated by the VECTORPLOT statement. 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.

For more information, see “VECTORPLOT Statement” in *SAS Graph Template Language: Reference*. 
A waterfall chart is generated by the WATERFALLCHART statement. A waterfall chart is typically used to show credit and debit transactions or successive changes to a given state. The bars in a waterfall chart that are calculated from the data 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.

For more information, see “WATERFALLCHART Statement” in *SAS Graph Template Language: Reference*.

The GTL provides additional plot statements that add features such as reference lines, drop lines, and axis-aligned tables to your plot. Table 4.2 on page 59 provides a list of these plots and an example of each. The statement that generates each plot is also listed.

### Table 4.2 Additional Plot Features

<table>
<thead>
<tr>
<th>Plot Feature Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Axis-Aligned Table</strong></td>
<td>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. For more information, see “AXISTABLE Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
<tr>
<td><strong>Drop Lines (At Inflection Point)</strong></td>
<td>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. For more information, see “DROPLINE Statement” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
</tbody>
</table>
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.

For more information, see “`REFERENCELINE Statement`” in *SAS Graph Template Language: Reference*.

<table>
<thead>
<tr>
<th>Plot Feature Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Reference Lines (At Inflection Point)" /></td>
<td>A reference line is drawn by the <code>REFERENCELINE</code> statement. Specifying the <code>X=</code> option creates a line perpendicular to the X-axis at an X-intercept. Specifying the <code>Y=</code> option creates a line perpendicular to the Y-axis at a Y-intercept. You must use the <code>REFERENCELINE</code> statement with another plot statement that establishes the axes. For more information, see “<code>REFERENCELINE Statement</code>” in <em>SAS Graph Template Language: Reference</em>.</td>
</tr>
</tbody>
</table>
Chapter 5
How the GTL Plot Statements Are Categorized

Plot Types

Overview of the Plot Types

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

- computed plot
- parameterized plot
- stand-alone 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 \( Y = \text{eval}(\log_{10}(\text{column})) \). 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 \( X = \text{column} \) provides the bar tick value and the \( Y = \text{column} \) 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.

**Stand-Alone Plots**

A stand-alone plot is one that can be drawn without any other accompanying plot. In general, a plot is stand-alone if its input data defines a range of values for all axes that are needed to display the plot. For example, the observations plotted in a SCATTERPLOT normally span a certain data range in both \( X \) and \( Y \) axes. This information is necessary to successfully draw the axes and the markers. It is important to know which plots are stand-alone 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 stand-alone 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 15.

**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 stand-alone, computed plots
- 2-D stand-alone, parameterized plots
- 3-D stand-alone, parameterized plots
- dependent plots

Chapter 6, “Plot Statements by Category,” on page 65 provides a list of the plot statements that are included each of these categories.

**Additional Plot Category: the Primary Plot**

For stand-alone 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 stand-alone 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 stand-alone 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 stand-alone plot in the template definition might not produce the axis properties that you want. You can include the PRIMARY=TRUE option in any stand-alone 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 stand-alone plot statement has this option, the first statement encountered that has the PRIMARY=TRUE option is considered the primary plot.
Chapter 6
Plot Statements by Category

Stand-alone, 2-D, Computed Plots .................................................. 65
Stand-alone, 2-D, Parameterized Plots ........................................... 67
Stand-alone, 3-D, Parameterized Plots .......................................... 69
Dependent Plots ................................................................. 69

Stand-alone, 2-D, Computed Plots

Table 6.1 Stand-alone, 2-D, Computed Plots

<table>
<thead>
<tr>
<th>Statement</th>
<th>Required Arguments</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARCHART</td>
<td>One column</td>
<td>Horizontal or vertical. For details, see “BARCHART Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>BOXPLOT</td>
<td>One numeric-column</td>
<td>Horizontal or vertical. For details, see “BOXPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>CONTOURPLOTPARM</td>
<td>Three numeric-columns</td>
<td>Draws contour plot from pre-gridded data. Basic &quot;gridding&quot; feature is provided using an option. For details, see “CONTOURPLOTPARM Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>DENSITYPLOT</td>
<td>One numeric-column</td>
<td>Theoretical distribution curve (for example, NORMAL or KDE). For details, see “DENSITYPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>ELLIPSE</td>
<td>Two numeric-columns</td>
<td>Confidence or prediction ellipse for a set of points. For details, see “ELLIPSE Statement” in SAS Graph Template Language: Reference.</td>
</tr>
</tbody>
</table>
## 2-D Plots: Computed

<table>
<thead>
<tr>
<th>Statement</th>
<th>Required Arguments</th>
<th>Comments</th>
</tr>
</thead>
</table>
| **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 the third maintenance release of SAS 9.4 and later releases.  
For details, see “**HEATMAP Statement**” in *SAS Graph Template Language: Reference.* |
| **HISTOGRAM**        | One numeric-column | Horizontal or vertical. For details, see “**HISTOGRAM Statement**” 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 Statement**” in *SAS Graph Template Language: Reference.* |
| **LOESSPLOT**        | Two numeric-columns| Fit plot using loess. For details, see “**LOESSPLOT Statement**” in *SAS Graph Template Language: Reference.* |
| **PBSPLINEPLOT**     | Two numeric-columns| Fit plot using Penalized B-spline. For details, see “**PBSPLINEPLOT Statement**” in *SAS Graph Template Language: Reference.* |
| **PIECHART**         | One column         | Must be used within a LAYOUT REGION block. For details, see “**PIECHART Statement**” in *SAS Graph Template Language: Reference.* |
| **REGRESSIONPLOT**   | Two numeric-columns| Fit plot using linear, quadratic, or cubic regression. For details, see “**REGRESSIONPLOT Statement**” 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 Statement**” 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 “**WATERFALLCHART Statement**” in *SAS Graph Template Language: Reference.* |
## Stand-alone, 2-D, Parameterized Plots

### Table 6.2  Stand-alone, 2-D, Parameterized Plots

<table>
<thead>
<tr>
<th>Statement</th>
<th>Required Arguments</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANDPLOT</td>
<td>Three columns, at least two numeric limits</td>
<td>Area bounded by two straight or curved lines. The input data must be sorted by the X or Y variable. For details, see “BANDPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>BARCHARTPARM</td>
<td>Two columns, Y must be numeric</td>
<td>Horizontal or vertical. Summarized data provided by user. For details, see “BARCHARTPARM Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>BLOCKPLOT</td>
<td>Two columns</td>
<td>Strip of X-axis aligned rectangular blocks containing text. The X data must be sorted. For details, see “BLOCKPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>BOXPLOTPARM</td>
<td>One numeric-column and one string-column</td>
<td>Horizontal or vertical. Needs special data format. For details, see “BOXPLOTPARM Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>BUBBLEPLOT</td>
<td>Three numeric-columns</td>
<td>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 Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>DENDROGRAM</td>
<td>Three numeric-columns</td>
<td>Tree diagram that represents the results of a hierarchical clustering analysis. For details, see “DENDROGRAM Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>ELLIPSEPARM</td>
<td>Five numbers or numeric-columns</td>
<td>Draws ellipse given center, slope, semi-major, and semi-minor axis lengths. For details, see “ELLIPSEPARM Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>FRINGE PLOT</td>
<td>One numeric-column</td>
<td>Draws a short line segment of equal length along the X or X2 axis for each observation's X value. For details, see “FRINGE PLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>HEATMAPPARM</td>
<td>Two columns</td>
<td>expressions and one numeric-column</td>
</tr>
<tr>
<td>Statement</td>
<td>Required Arguments</td>
<td>Comments</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HIGHLOWPLOT</td>
<td>Three columns. HIGH, and LOW must be numeric</td>
<td>Draws a high-low bar or line plot. For details, see “HIGHLOWPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>HISTOGRAMPARM</td>
<td>Two numeric-columns</td>
<td>Horizontal or vertical. The Y data must be nonnegative. For details, see “HISTOGRAMPARM Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>MOSAICPLOTPARM</td>
<td>List of categorical columns enclosed in parenthesis and a numeric-column.</td>
<td>Creates a mosaic plot from pre-summarized categorical data. The numeric column must contain only positive values. For details, see “MOSAICPLOTPARM Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>NEEDLEPLOT</td>
<td>Two columns, Y must be numeric</td>
<td>Draws parallel, vertical line segments connecting data points to a baseline. For details, see “NEEDLEPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>POLYGONPLOT</td>
<td>Three columns</td>
<td>Draws one or more polygons from data that is stored in a data set. Note: This statement is valid in the first maintenance release of SAS 9.4 and later releases. For details, see “POLYGONPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>SCATTERPLOT</td>
<td>Two columns</td>
<td>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 Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>SERIESPLOT</td>
<td>Two columns</td>
<td>Draws line segments to connect a set of data points. For details, see “SERIESPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>STEPPLOT</td>
<td>Two columns, Y must be numeric</td>
<td>Draws stepped line segments to connect a set of data points. For details, see “STEPPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>TEXTPLOT</td>
<td>Three columns</td>
<td>Draws text markers at data point locations. Note: This statement is valid in the second maintenance release of SAS 9.4 and later releases. The marker text is provided by a column in the plot data. For details, see “TEXTPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>VECTORPLOT</td>
<td>Four numeric-columns, X and Y origins can be numeric constants.</td>
<td>Creates directed line segment(s) based on pairs of data points. For details, see “VECTORPLOT Statement” in SAS Graph Template Language: Reference.</td>
</tr>
</tbody>
</table>
Stand-alone, 3-D, Parameterized Plots

Table 6.3  Stand-alone, 3-D, Parameterized Plots

<table>
<thead>
<tr>
<th>Statement</th>
<th>Required Arguments</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIHISTOGRAM3DPARM</td>
<td>Three numeric-columns</td>
<td>Bivariate histogram. The Z data must be nonnegative. For details, see “BIHISTOGRAM3DPARM Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>SURFACEPLOTPARM</td>
<td>Three numeric-columns</td>
<td>Smooth surface. For details, see “SURFACEPLOTPARM Statement” in SAS Graph Template Language: Reference.</td>
</tr>
</tbody>
</table>

Dependent Plots

Table 6.4  Dependent Plots

<table>
<thead>
<tr>
<th>Statement</th>
<th>Required Arguments</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROPLINE</td>
<td>(X,Y) point location, two columns, or one value and one column</td>
<td>Draws a perpendicular line from a data point to a specified axis. For details, see “DROPLINE Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>LINEPARM</td>
<td>(X,Y) point location and slope. The three values can be provided in any combination of number and numeric-column</td>
<td>Draws line(s) given a data point and the slope of the line. For details, see “LINEPARM Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>MODELBAND</td>
<td>CLM or CLI name of associated fit plot</td>
<td>Confidence bands. Used only in conjunction with a fit plot. For details, see “MODELBAND Statement” in SAS Graph Template Language: Reference.</td>
</tr>
<tr>
<td>REFERENCEDLINE</td>
<td>X or Y location, column</td>
<td>Draws line(s) perpendicular to an axis. For details, see “REFERENCEDLINE Statement” in SAS Graph Template Language: Reference.</td>
</tr>
</tbody>
</table>
Part 3

Layouts

Chapter 7
  Overview of the Graph Template Language Layouts ............... 73

Chapter 8
  Creating Overlay Graphs Using the OVERLAY Layout ............ 79

Chapter 9
  Creating Overlay Graphs with Equated Axes
  Using the OVERLAYEQUATED Layout .......................... 157

Chapter 10
  Creating Overlay 3-D Graphs Using the OVERLAY3D Layout .... 167

Chapter 11
  Creating Gridded Graphs Using the GRIDDED Layout ............ 187

Chapter 12
  Creating Lattice Graphs Using the LATTICE Layout ............ 201

Chapter 13
  Creating Classification Panels Using the
  DATALATTICE and DATAPANEL Layouts ....................... 235

Chapter 14
  Creating Graphs with No Axis Using the REGION Layout ....... 283
Chapter 7
Overview of the Graph Template Language Layouts

Layouts in the GTL

The 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.

<table>
<thead>
<tr>
<th>One-Cell Layout</th>
<th>Four-Cell Layout</th>
<th>Five-Cell Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell</td>
<td>Cell 1</td>
<td>Cell 1</td>
</tr>
<tr>
<td></td>
<td>Cell 2</td>
<td>Cell 2</td>
</tr>
<tr>
<td></td>
<td>Cell 3</td>
<td>Cell 3</td>
</tr>
<tr>
<td></td>
<td>Cell 4</td>
<td>Cell 4</td>
</tr>
<tr>
<td></td>
<td>Cell 2</td>
<td>Cell 4</td>
</tr>
<tr>
<td></td>
<td>Cell 3</td>
<td>Cell 5</td>
</tr>
</tbody>
</table>

Layout Types

The 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 74.
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 75.

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 75.

---

**Single-Cell Layout Statements**

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAYOUT OVERLAY</td>
<td>Enables you to combine 2-D plots, titles, footnotes, legends, and text into a single cell. See Chapter 8, “Creating Overlay Graphs Using the OVERLAY Layout,” on page 79.</td>
</tr>
<tr>
<td>LAYOUT OVERLAY3D</td>
<td>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 10, “Creating Overlay 3-D Graphs Using the OVERLAY3D Layout,” on page 167.</td>
</tr>
<tr>
<td>LAYOUT OVERLAYEQUATED</td>
<td>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 9, “Creating Overlay Graphs with Equated Axes Using the OVERLAYEQUATED Layout,” on page 157.</td>
</tr>
<tr>
<td>LAYOUT REGION</td>
<td>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 14, “Creating Graphs with No Axis Using the REGION Layout,” on page 283.</td>
</tr>
</tbody>
</table>
Multi-Cell, Non-Data-Driven Layout Statements

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAYOUT GRIDDED</td>
<td>Enables you to combine multiple 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 11, “Creating Gridded Graphs Using the GRIDDED Layout,” on page 187.</td>
</tr>
<tr>
<td>LAYOUT LATTICE</td>
<td>Enables you to create panel of 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 12, “Creating Lattice Graphs Using the LATTICE Layout,” on page 201.</td>
</tr>
</tbody>
</table>

Multi-Cell, Data-Driven Layout Statements

The 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.
<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAYOUT DATAPANEL</td>
<td>Enables you to specify one or more classification variables. You specify these variables in a list:</td>
</tr>
<tr>
<td></td>
<td>classvars=(var1, var2, var3, var4 ...)</td>
</tr>
<tr>
<td></td>
<td>classvars=(country, product)</td>
</tr>
<tr>
<td></td>
<td>The order in which the individual crossings are added to the grid is based on the following criteria:</td>
</tr>
<tr>
<td></td>
<td>• the order in which you specify the classification variables. The last variable that you specify is the first variable to vary.</td>
</tr>
<tr>
<td></td>
<td>• whether you specify that cells should be populated by row or by column.</td>
</tr>
<tr>
<td></td>
<td>• the order that variable values occur in your data. You might want to sort the data by the classification variable.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>LAYOUT DATALATTICE</td>
<td>Enables you to specify up to two classification variables. You specify whether each variable is a row classifier or a column classifier.</td>
</tr>
<tr>
<td></td>
<td>rowvar=country</td>
</tr>
<tr>
<td></td>
<td>columnvar=product</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

See Chapter 13, “Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts,” on page 235.

**Axis Management**

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.
<table>
<thead>
<tr>
<th>Layout Statement</th>
<th>How the Axes Are Managed</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAYOUT OVERLAY</td>
<td>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.</td>
</tr>
<tr>
<td>LAYOUT OVERLAY3D</td>
<td></td>
</tr>
<tr>
<td>LAYOUT OVERLAYEQUATED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For information, see:</td>
</tr>
<tr>
<td></td>
<td>• “Managing Axes in OVERLAY Layouts” on page 85</td>
</tr>
<tr>
<td></td>
<td>• “Managing Axes in OVERLAYEQUATED Layouts” on page 159</td>
</tr>
<tr>
<td></td>
<td>• “Managing Axes in OVERLAY3D Layouts” on page 181</td>
</tr>
<tr>
<td>LAYOUT LATTICE</td>
<td>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 210.</td>
</tr>
<tr>
<td>LAYOUT DATAPANEL</td>
<td>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 247.</td>
</tr>
<tr>
<td>LAYOUT DATALATTICE</td>
<td></td>
</tr>
</tbody>
</table>

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 bases.
Chapter 8
Creating Overlay Graphs Using the OVERLAY Layout

The LAYOUT OVERLAY Statement ........................................... 80
Statements You Can Use in an OVERLAY Block ...................... 81
Restrictions on Allowed Statements ........................................ 81
Restrictions on Statement Combinations ............................... 82
Statement Order ........................................................................ 84
Managing Axes in OVERLAY Layouts ..................................... 85
   Overview ............................................................................... 85
   Axis Terminology ............................................................... 85
   How Plot Statements Affect Axis Construction ...................... 86
   Avoiding Plot Axis Conflicts ................................................. 90
   Axis Line versus Wall Outline ............................................. 91
   Axis Appearance Features Controlled by the Current Style .......... 94
   Specifying Axis Options ..................................................... 95
   Axis Type ........................................................................... 97
   Axis Data Range ............................................................... 98
   Axis Labels ........................................................................ 99
   Axis Tick Values .............................................................. 106
   Axis Thresholds ............................................................... 106
   Axis Offsets ...................................................................... 110
   LINEAR Axes .................................................................... 113
   Discrete Axes .................................................................... 121
   TIME Axes ....................................................................... 131
   LOG Axes ......................................................................... 137
Avoiding Plot Data Conflicts .................................................. 144
Overlay Examples ..................................................................... 145
   Vertical and Horizontal Bar-Line Charts ............................... 145
   Plot with Multiple Axes ..................................................... 147
   Plot with Fit Line ............................................................... 149
   Plot with Fit Line with Confidence Bands ............................ 150
   Plot of Grouped Data ........................................................ 152
   Using Overlays to Graph Multiple Response Variables .......... 153
   Plot with Insets .................................................................. 154
   Plot Appearance ............................................................... 154
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)
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.

However, the following restrictions apply:

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

Restrictions on Allowed Statements

Even among the statements that are valid within an OVERLAY layout, some restrictions apply to their use. For example, some dependent statements must be accompanied by at least one stand-alone plot statement, such as SCATTERPLOT or SERIESPLOT, in order to produce a usable graph. See “Categories of Statements” on page 13 for lists of stand-alone 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 stand-alone plot statement. (See “REFERENCELINE Statement” in SAS Graph Template Language: Reference.) The stand-alone plot statement must provide an axis data range that is sufficient for determining the reference line axis extents. In this example, a stand-alone 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 8.1  Statement Combination Example**

```sas
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.

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

When the template in Example Code 8.1 on page 82 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:

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

When the template in Example Code 8.1 on page 82 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 must use a discrete
axis, and so 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.

```sas
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.

```sas
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.

![Model Weight by Height](image)

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. This is demonstrated in “Example: Building a Model Fit Plot” on page 35.
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=)
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 8.2  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.

![Graph showing X and Y Axes](image)

Explicitly, the layout block means the following:

```plaintext
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.

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

Here is example output.

![Graph showing X2 and Y2 Axes](image)

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.

```plaintext
layout overlay;
```
This layout block can be more compactly written as follows:

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

Here is example output.

![Graph](image)

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 95.

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.

```plaintext
layout overlay / yaxisopts=(griddisplay=on);
  scatterplot x=City y=Fahrenheit;
  scatterplot x=City y=Celsius / datatransparency=1 yaxis=y2;
endlayout;
```
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 106 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.

```sas
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.

![Graph showing Y and Y2 Axes Use Different Data](image)

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:

```gtl
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.

```gtl
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 95.

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.

```sas
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.

```sas
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:

```sas
/* 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 8.2 on page 86.
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.

<table>
<thead>
<tr>
<th>Style Element</th>
<th>Style Attributes</th>
<th>Values</th>
<th>Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphAxisLines</td>
<td>TickDisplay</td>
<td>&quot;ACROSS&quot; &quot;INSIDE&quot; &quot;OUTSIDE&quot;</td>
<td>Tick mark location</td>
</tr>
<tr>
<td></td>
<td>LineStyle</td>
<td>Integer: 1 to 49</td>
<td>Axis line pattern</td>
</tr>
<tr>
<td></td>
<td>LineThickness</td>
<td>Dimension</td>
<td>Axis line and tick thickness</td>
</tr>
<tr>
<td></td>
<td>ContrastColor</td>
<td>Color</td>
<td>Axis line and tick color</td>
</tr>
<tr>
<td>GraphGridlines</td>
<td>DisplayOpts</td>
<td>&quot;AUTO&quot; &quot;ON&quot; &quot;OFF&quot;</td>
<td>When to display grid lines</td>
</tr>
<tr>
<td></td>
<td>LineStyle</td>
<td>Integer: 1 to 49</td>
<td>Grid line pattern</td>
</tr>
<tr>
<td></td>
<td>LineThickness</td>
<td>Dimension</td>
<td>Grid line thickness</td>
</tr>
<tr>
<td></td>
<td>ContrastColor</td>
<td>Color</td>
<td>Grid line color</td>
</tr>
<tr>
<td>GraphLabelText</td>
<td>Color</td>
<td>Color</td>
<td>Axis label text color</td>
</tr>
<tr>
<td></td>
<td>Font</td>
<td>font-specification*</td>
<td>Axis label font</td>
</tr>
<tr>
<td>GraphValueText</td>
<td>Color</td>
<td>Color</td>
<td>Axis tick value text color</td>
</tr>
<tr>
<td></td>
<td>Font</td>
<td>font-specification*</td>
<td>Axis tick value font</td>
</tr>
</tbody>
</table>

* A style font-specification includes attributes for FONTFAMILY, FONTFAMILY, FONTWEIGHT, FONTFAMILY, 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Overrides</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRIDDISPLAY=</td>
<td>DisplayOpts attribute of GraphGridLines</td>
</tr>
<tr>
<td>GRIDATTRS=</td>
<td>GraphGridLines</td>
</tr>
</tbody>
</table>
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;
```
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:

```sas
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.

```sas
layout overlay / xaxisopts=( display=( line label tickvalues ) )
yaxisopts=( displaysecondary=( ticks tickvalues line ) griddisplay=on )
  barchartparm category=city response=fahrenheit;
endlayout;
```
Here is example output.

![Graph example](image)

**Axis Type**

To determine axis types, the OVERLAY container examines all of the stand-alone plot statements that are specified. It also examines whether an axis type has been specified with the TYPE= setting on an axis option (for example, on XAXISOPTS=). If there is only one stand-alone plot, or a plot is designated as PRIMARY, the rules are 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 131 for examples. Otherwise, the numeric axis type is LINEAR, the general numeric axis type. See “LINEAR Axes” on page 113 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 137 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 the 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 stand-alone plot that is specified in the layout and no other plot has been set as the primary plot.

```plaintext
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.

```plaintext
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.

```plaintext
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.

```plaintext
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 113, “TIME Axes” on page 131, and “LOG
Axes” on page 137.

Axis Labels

Default 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 94. 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 8.3 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, we can use a
simple overlay of series plots. Whenever two or more columns are mapped to the same
axis, the primary plot determines the axis label. In the following example, the first
SERIESPLOT is primary by default, so its columns determine the axis labels. In this
case, the Y-axis label is determined by the Bacteria column.

  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.

![Graph](image)

If we 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.

```plaintext
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 8.1  Primary Plot Determines Default Axis Label**

![Graph](image)

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.

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

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.
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 a new column that can be used as a required argument as shown in the following example.

```sas
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 Label**

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

```sas
layout overlay / xaxisopts=(label="Product of Weight and Height");
   histogram eval(weight*height);
endlayout;
```

![Default Axis Label for a Computed Column](image)
Here is example output.

![Custom Axis Label for a Computed Column](image)

In this example, the manufactured label BIN(WEIGHT*HEIGHT) is replaced with 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.

![Axis Label Truncation](image)

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 8.4  Short Axis Label Example**

```text
proc template;
  define statgraph shortaxislabel;
  begingraph;
    entrytitle "Short Label Substitution";
    layout overlay / cycleattrs=true
      yaxisopts=(label="Growth of Virus and Bacteria Cultures"
```
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 8.4 on page 102 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 Labels

By default, axis labels are centered in the axis area as shown in Figure 8.1 on page 100. 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=:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CENTER</td>
<td>centers the axis label in the axis area (default).</td>
</tr>
<tr>
<td>DATACENTER</td>
<td>for OVERLAY layouts, centers the axis label in the axis tick display area.</td>
</tr>
<tr>
<td>TOP or BOTTOM</td>
<td>orients the label horizontally at the top or bottom of the axis area.</td>
</tr>
<tr>
<td>LEFT or RIGHT</td>
<td>positions the label to the left or right of the axis area.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CENTER</td>
<td>centers the axis label in the axis area (default).</td>
</tr>
<tr>
<td>DATACENTER</td>
<td>centers each row or column axis label in its axis tick display area.</td>
</tr>
</tbody>
</table>

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.

```sas
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;
```

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

**Note:** See Example Code 8.3 on page 99.
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 113, “Discrete Axes” on page 121, and “LOG Axes” on page 137.

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 94.

*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 116.

**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 8.2 on page 86.

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)
```

Here is example output.

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

```sas
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.

```plaintext
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 8.5  TICKVALUELIST Example**

```sas
proc template;
define statgraph heightchart;
begingraph;
entrytitle "Specifying Tick Values for Linear Axes";
layout overlay /
xaxisopts=(type=linear griddisplay=on griddisplayattrs=(pattern=dot color=lightgray) linearopts=(minorticks=true))
yaxisopts=(type=linear tickvaluehalign=left griddisplay=on griddisplayattrs=(pattern=dot color=lightgray) linearopts=(viewmin=48 viewmax=78 tickvalueclist=(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;
```

**Managing Axes in OVERLAY Layouts**

113
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 8.5 on page 113 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.

```
Specifying Tick Display Values for Linear Axes

<table>
<thead>
<tr>
<th>Height</th>
<th>4 ft</th>
<th>5 ft</th>
<th>5.5 ft</th>
<th>6 ft</th>
<th>6.5 ft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>60</td>
<td>80</td>
<td>100</td>
<td>120</td>
<td>140</td>
</tr>
</tbody>
</table>
```

**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 631.

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 /
      xaxiosopts=(griddisplay=on gridattrs=|pattern=dot)
      timeopts=(tickvalueformat=monname3. viewmax='01DEC2001'd)
      yaxiosopts=(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;
  endgraph;
end;
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 102 for 15 hours of growth data.

```sas
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;
```

Here is the output.
end;
run;

proc template;
define statgraph axisdefaultscale;
begingroup;
  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:

<table>
<thead>
<tr>
<th>Linear Axis</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
</tr>
</tbody>
</table>

![Graph with tick value default scaling](image)
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. \texttt{LINEAROPTS = (TICKVALUEFITPOLICY=THIN)} is the default action for linear axes:

![Linear Axis: TickvalueFitPolicy = THIN](image)

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

![Linear Axis: TickvalueFitPolicy = ROTATE](image)

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

![Linear Axis: TickvalueFitPolicy = ROTATE, TickValueRotation = VERTICAL](image)

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

\textit{Note:} The \texttt{ROTATE} and \texttt{ROTATEALWAYS} policies, and the \texttt{TICKVALUEROTATION=} option apply to the X and X2 axes only.

You can set \texttt{TICKVALUEFITPOLICY=STAGGER}, which creates alternating tick values on two rows:

![Linear Axis: TickvalueFitPolicy = STAGGER](image)

For the X and X2 axes, you can set \texttt{TICKVALUEFITPOLICY} to a compound policy \texttt{ROTATETHIN, STAGGERTHIN,} or \texttt{STAGGERROTATE}. The compound policies attempt the second policy if the first policy does not work. For the Y and Y2 axes, you can set \texttt{TICKVALUEFITPOLICY} to \texttt{NONE} or \texttt{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 627.
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 the first maintenance release of SAS 9.4, 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.

```sas
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 the third maintenance release of SAS 9.4, 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 8.2 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 8.2 on page 120.

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.

<table>
<thead>
<tr>
<th>BRACKET</th>
<th>NOTCH</th>
<th>SLANTEDLEFT</th>
<th>SLANTEDRIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3 6 8 10</td>
<td>1 3 6 8 10</td>
<td>1 3 6 8 10</td>
<td>1 3 6 8 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPARK</th>
<th>SQUIGGLE</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3 6 8 10</td>
<td>1 3 6 8 10</td>
<td>1 3 6 8 10</td>
</tr>
</tbody>
</table>

For more information about the BEGINGRAPH statement AXISBREAKTYPE= and AXISBREAKSYMBOL= options, see “BEGINGRAPH Statement” in SAS Graph Template Language: Reference.
**Discrete Axes**

**Setting the Tick Values on a Discrete Axis**

On a discrete axis, by default, the data values that are represented in the graph are displayed as tick values on the axis. Along the axis, numeric tick values are arranged in ascending order and character values are arranged in the order in which they are used in the graph. 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 use 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.

```sas
/* Create a test data set of annual failure data */
data failrate;
  do month=1 to 12 by 1;
    failures=2*ranuni(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.

```
/* 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
```

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 the third maintenance release of SAS 9.4, 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 TICKVALUETYPE= option. Also starting with the third maintenance release of SAS 9.4, 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 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 '2265'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\textsubscript{n} and No\textsubscript{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\textsubscript{n} category values into tick
value Y, and all of the No\textsubscript{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.

\textbf{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.
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 117. A DISCRETE axis supports additional fit policies that enable you to do the following:

- 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 124.
- 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 126.
- Truncate the axis values with the TRUNCATE fit policy. See “Truncating the Discrete Axis Tick Values” on page 127.

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

<table>
<thead>
<tr>
<th>Compound Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROTATEALWAYSDROP</td>
</tr>
<tr>
<td>SPLITALWAYSTHIN</td>
</tr>
<tr>
<td>SPLITROTATE</td>
</tr>
<tr>
<td>RATETHIN</td>
</tr>
<tr>
<td>STAGGERTRUNCATE</td>
</tr>
<tr>
<td>TRUNCATETHIN</td>
</tr>
<tr>
<td>TRUNCATEROTATE</td>
</tr>
</tbody>
</table>

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;
begin graph / 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 627.

Splitting the Discrete Axis Tick Values
For long tick values, you can set the TICKVALUEFITPOLICY= option to SPLIT to split the tick values, when necessary, 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, we 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 TICKVALUESPLITITCHARDROP=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, suppose that we decide to truncate the values from the previous example rather than split them. If we 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.

```sas
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.

![MPG City - European Makes](attachment:image.png)
### 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
- BOXPLOT
- 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.

![Product Sales by Country](image)

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.

```sas
/* 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, the Date column has a SAS date format associated with it. By default, the TIME axis decides an appropriate tick value format and an interval to display. Notice that, in the default case, when the X or X2 axis is a TIME axis, the space that is used for the tick values is conserved by splitting the values at appropriate date or time intervals and extracting larger intervals. In this example, the column format for the Date 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.

```plaintext
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, say 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.

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

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.

<table>
<thead>
<tr>
<th>Original Form</th>
<th>Split Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY-MONTH-YEAR</td>
<td>DAY-MONTH</td>
</tr>
<tr>
<td></td>
<td>YEAR</td>
</tr>
<tr>
<td>QUARTER-YEAR</td>
<td>QUARTER</td>
</tr>
<tr>
<td></td>
<td>YEAR</td>
</tr>
<tr>
<td>MONTH-YEAR</td>
<td>MONTH</td>
</tr>
<tr>
<td></td>
<td>YEAR</td>
</tr>
<tr>
<td>TIME-DATE</td>
<td>TIME</td>
</tr>
<tr>
<td></td>
<td>DATE</td>
</tr>
<tr>
<td>TIME only</td>
<td>original form</td>
</tr>
<tr>
<td>YEAR, QUARTER, MONTH, or DAY only</td>
<td>original form</td>
</tr>
<tr>
<td>International format</td>
<td>original form</td>
</tr>
</tbody>
</table>

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

```plaintext
layout overlay / xaxisopts=(timeopts=(interval=quarter
splittickvalue=false));
seriesplot x=date y=close;
endlayout;
```
Here is example output.

```
    close
```

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.

```
    close
```

**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 631.

**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));
```
Here is example output.

```plaintext
seriesplot x=date y=close;
endlayout;
```

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 627.

### 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.

```plaintext
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;
```

```plaintext
proc sgrender data=sashelp.stocks template=timeaxis1;
where stock="IBM" and date between "1dec2002"d and "31dec2005"d;
run;
```
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 the first maintenance release of SAS 9.4, 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.

```sas
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.

![Broken Time Axis Example](image-url)
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 the third maintenance release of SAS 9.4, 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 119.

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.

```sas
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 8.3 on page 99.

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 TICKINTERVALSTYLE=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;
```
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.

```plaintext
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 142.

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.

```plaintext
layout overlay;
seriesplot x=Hours y=eval(log10(growth));
endlayout;
```
Do not use a LOG axis in this case. Here is example output.

**Linear Axis Displaying Existing Log Data**

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.

```plaintext
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.

**Log Axis With Tick Value List**
You can use the VALUETYPES= option to specify the TICKVALUELIST= option values in a scale and format other than that specified by the TICKINVERV ALSTYLE= 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.

```plaintext
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 the third maintenance release of SAS 9.4, 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 TICKINVERV ALSTYLE= 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 631.

When TICKINVERV ALSTYLE=LOGEXPAND, the specified format is honored for base 2 and base 10 logarithmic scales. When TICKINVERV ALSTYLE=LINEAR, the specified format is honored for base 10, base 2, and base E logarithmic scales. Here is an example.

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

```sas
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**

The 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 REFERENCENAME, 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 REFERENCE LINE and SERIES PLOT reflect this directly by changing the variables that are mapped to the X and Y axes. The variable mapping for BARCHART PARM remains how it was, but we add the ORIENT=HORIZONTAL option to swap the axis mappings. The data set up and SGRENDER step are unchanged. Here is the output.

![Overlay of REFERENCE LINE, BARCHART PARM and SERIES PLOT](image)

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.

```plaintext
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. The following is a complete program that generates this type of graph. In this example, we want to display Fahrenheit temperatures on a separate Y2 axis from the Y axis used to display Celsius temperatures 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.

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

```plaintext
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 149, the REGRESSIONPLOT statement can also generate sets of points for the upper and lower confidence limits of the mean (CLM), and for the upper and lower confidence limits of individual predicted values (CLI) for each observation. The CLM="name" and CLI="name" options cause the extra computation. However, the confidence limits are not displayed by the regression plot. Instead, you must use the dependent plot statement MODELBAND, with the unique name as its required argument as shown in the following layout block.

```plaintext
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, stand-alone plots to build the fit plot. First, choose a procedure to do the analysis. The following example uses the REG procedure.

```sas
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.

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

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.

```sas
proc template;
  define statgraph fit;
  begingraph;
    entrytitle "Regression Fit Plot";
    layout overlay;
      bandplot x=height
          limitupper=uclm
          limitlower=lclm
    endgraph;
  enddefine;
run;
```
Here is the output.

![Regression Fit Plot](image)

**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.

```sas
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:

![Overlay of SERIESPLOTs with GROUP=](image)

**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.

```sas
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.

![Overlay of Multiple SERIESPLOTs](image)

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. The GTL provides several ways in which can inset supplemental information in your graphs. For more information, see Chapter 17, “Adding Insets to Your Graph,” on page 351.

**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 153, you can use the LINEATTRS= and CURVELABELATTRS= options as shown in the following layout block:

``` Sas
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.

``` Sas
layout overlay / yaxisopts=(label="IBM Stock Price") cycleattrs=true;
  seriesplot x=date y=high / curvelabel="High";
```
Either coding produces the following graph:

For additional information about how set the appearance features of plots, see Chapter 23, “Managing Your Graph’s Appearance,” on page 445.
Chapter 9
Creating Overlay Graphs with Equated Axes Using the OVERLAYEQUATED Layout

The LAYOUT OVERLAYEQUATED Statement ........................................ 157
Managing Axes in OVERLAYEQUATED Layouts ................................. 159
  Types of Equated Axes ................................................................. 159
  Defining Axes for Equated Layouts ............................................. 162
Equated Overlay Layout Examples ............................................... 162

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 9.1 Sample Plot from PROC PRINQUAL

![Multidimensional Preference Analysis](image)
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 159.)

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 OVERLAYEQUATED 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:

```sas
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 85 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, we will look at a simplified version of the PPLOT template that is supplied with PROC UNIVARIATE, which is delivered with Base SAS. The following code shows a SAS program that can be used to run PROC UNIVARIATE:

```sas
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 PPLOT 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 PPLOT 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.

```sas
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:

```sas
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.

![P-P Plot for Weight](image)

The following OVERLAYEQUATED statement modifies the equated axes:

```plaintext
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](image)

*Derived from PPLOT template*
Chapter 10
Creating Overlay 3-D Graphs
Using the OVERLAY3D Layout

The LAYOUT OVERLAY3D Statement

Data Requirements for 3-D Plots

Managing Axes in OVERLAY3D Layouts

Display Features of 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 graph:
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 Chapter 5, “How the GTL Plot Statements Are Categorized,” on page 61). 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).

```sas
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.

<table>
<thead>
<tr>
<th>Obs</th>
<th>value1</th>
<th>value2</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51.5000</td>
<td>67.000</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>51.5000</td>
<td>92.889</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>51.5000</td>
<td>118.778</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>51.5000</td>
<td>144.667</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>51.5000</td>
<td>170.556</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>51.5000</td>
<td>196.444</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>51.5000</td>
<td>222.333</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>51.5000</td>
<td>248.222</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>51.5000</td>
<td>274.111</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>51.5000</td>
<td>300.000</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>55.0714</td>
<td>67.000</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>55.0714</td>
<td>92.889</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>55.0714</td>
<td>118.778</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>55.0714</td>
<td>144.667</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>55.0714</td>
<td>170.556</td>
<td>1</td>
</tr>
</tbody>
</table>

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 10.1 KDE Data Plot Template**

```sas
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.

```sas
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 10.1 on page 170 for only nonzero values:

```sas
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:

```sas
proc kde data=sashelp.heart;
```
Setting Bin Width. Another technique for binning data is to set a bin width and compute the number of observations in each bin. With this technique that you do not know the exact number of bins, but you can assure 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;
Notice that the COMPLETETYPES option forces all possible combinations of the two variables to be output, even if no data exists for a particular crossing. The template can be simplified as shown in the following because we know that the bin midpoints are equally spaced integers.

```sas
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;
```

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 than 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:

```sas
proc template;
  define statgraph bihistogram2a;
  begingraph;
```
entrytitle "Distribution of Height and Weight";
entryfootnote halign=right "SASHHELP.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 "SASHHELP.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.

```sas
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;
end;
run;
ods graphics / antialiasmax=5700;
proc sgrender data=sashelp.lake template=surfaceplotparm;
run;
```
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.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Width</th>
<th>Length</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0.0</td>
<td>0.00000</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.5</td>
<td>0.00000</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1.0</td>
<td>0.00000</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1.5</td>
<td>-0.00598</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2.0</td>
<td>0.00000</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2.5</td>
<td>0.00000</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>3.0</td>
<td>0.00000</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>3.5</td>
<td>-0.21287</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>4.0</td>
<td>0.00000</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>4.5</td>
<td>0.00000</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>5.0</td>
<td>0.00000</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>5.5</td>
<td>-0.00299</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>6.0</td>
<td>0.00000</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>6.5</td>
<td>0.00000</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>7.0</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

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.

```sas
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 16, “Adding Legends to Your Graph,” on page 309.

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, LOG, or TIME. A DISCRETE axis is not supported on OVERLAY3D layouts.
- For a LOG axis, the LOGOPTS= option 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;
```
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.

```sas
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;
```
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.

```plaintext
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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value Range</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROTATE=</td>
<td>-360 to 360</td>
<td>54</td>
<td>Specifies the angle of rotation. Rotation is measured in a clockwise direction about a virtual axis, parallel to the Z axis (vertical) and passing through the center of the bounding cube. A counterclockwise rotation can be specified with a negative value.</td>
</tr>
<tr>
<td>TILT=</td>
<td>-360 to 360</td>
<td>20</td>
<td>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.</td>
</tr>
<tr>
<td>ZOOM=</td>
<td>&gt; 0</td>
<td>1</td>
<td>Specifies a zoom factor. Factors greater than 1 move closer to the bounding cube (zoom in), less than 1 move farther away (zoom out).</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Display Features of the OVERLAY3D Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using ROTATE=</td>
</tr>
<tr>
<td>DEFAULT</td>
</tr>
<tr>
<td>ROTATE=90</td>
</tr>
<tr>
<td>ROTATE=.36</td>
</tr>
</tbody>
</table>

| Using TILT=  |
| ROTATE=00 TILT=5  |
| TILT=90  |
| TILT=45  |

| Using ZOOM=  |
| TILT=60 ZOOM=.5  |
| TILT=60 ZOOM=1.5  |
| TILT=60 ZOOM=3.0  |
Chapter 11
Creating Gridded Graphs Using the GRIDDED Layout

The LAYOUT GRIDDED Statement ............................................. 187
Defining a Basic Grid ............................................................... 188
  Setting Grid Dimensions ....................................................... 188
  Setting Gutter .................................................................. 190
  Defining Cells ................................................................ 190
Building a Table of Text ......................................................... 193
Sizing Issues ........................................................................ 194
  Row and Column Sizes ........................................................ 194
  Adjusting Graph Size ........................................................... 198

The LAYOUT GRIDDED Statement

The GTL provides several layout types to organize your graph into smaller regions (cells). The GRIDDED and LATTICE layouts support a regular grid of cells with a fixed number of rows and columns. The DATALATTICE and DATAPANEL layouts generate classification panels, which are graphs where the number of cells and the cell content are determined by the values of one or more classification variables.

The GRIDDED layout differs from the classification panel layouts in that the number of cells must be predefined and that you must define 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 12, “Creating Lattice Graphs Using the LATTICE Layout,” on page 201.

Typical applications of GRIDDED layouts are to create:

• a table of text, such as an inset (discussed in detail in Chapter 17, “Adding Insets to Your Graph,” on page 351)
• a simple grid of plots (discussed in this chapter)

In a GRIDDED layout, each cell is independent. Contents of the cell can be specified by a stand-alone plot statement or a nested layout. The following example shows a very simple GRIDDED layout:

  proc template;
    define statgraph intro;
In this case, each plot statement is considered independent and is placed in a separate cell. When no grid size is provided, the default layout creates a graph with one column of cells, and it allots each cell the same amount of space. The number of rows in the grid is determined by the number and arrangement of stand-alone plot statements and nested layouts in the GRIDDED layout block.

**Defining a Basic Grid**

Although you can generate a nice looking graph in a default GRIDDED layout, in most cases you will want more control over the grid, how it is populated, and the complexity of the cell contents.

**Setting Grid Dimensions**

Assume you want a grid of five plots. Before starting to write code, you must first decide what grid dimensions you want to set (how many columns and rows) and whether you want to permit an empty cell in the grid. If do not want an empty cell, you must limit the grid to five cells, which gives you two choices for the grid dimensions: five columns by one row (5x1), or one column by five rows (1x5).
To specify the grid size, you use the COLUMNS= or ROWS= option in the LAYOUT GRIDDED statement. To use ROWS=, you must also specify ORDER=COLUMNMAJOR.

Two explicit specifications could be used to create the following grid, which contains one row and five columns:

```
layout gridded / columns=5;
   /* plot definitions */
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:

```plaintext
layout gridded / rows=2 order=columnmajor;
  /* plot1 definition */
  /* plot2 definition */
  /* plot3 definition */
  /* plot4 definition */
  /* plot5 definition */
endlayout;
```

![Grid with 2 rows and 3 columns](image)

**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.

```plaintext
layout gridded / columns=3 columngutter=5 rowgutter=5;
  /* plot1 definition */
  /* plot2 definition */
  /* plot3 definition */
  /* plot4 definition */
  /* plot5 definition */
endlayout;
```

![Grid with 3 columns and 2 rows](image)

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 194.

**Defining Cells**

Two valid techniques are available for indicating the contents of a cell:
### Technique | Example | Advantages | Disadvantages
---|---|---|---
stand-alone plot statement or text statement | `scatterplot x= y=;` | simplicity | 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.

```sas
proc template;
  define statgraph celldfendefine;
  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;
```

```sas
proc sgrender data=sashelp.class template=celldfendefine;
run;
```
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.

<table>
<thead>
<tr>
<th>N</th>
<th>5203</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>119.96</td>
</tr>
<tr>
<td>Std Dev</td>
<td>19.98</td>
</tr>
</tbody>
</table>

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.

```plaintext
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.

![Auto-positioning the Inset Within the Plot Wall](image)

Tables like this can be organized many different ways. For more information about these techniques, see Chapter 15, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 291 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 17, “Adding Insets to Your Graph,” on page 351 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.

```plaintext
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;
```
Sizing Issues

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.

In this example, the legend for the scatterplot with sex as the grouping variable is larger than the default cell size. As a result, the cell size is adjusted to accommodate 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 11.1 Three-Cell Gridded Layout Example**

```plaintext
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 */
  endgraph;
enddefine;

discreteLegend / displayclipped=clip;
```
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 12.1 on page 218.
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 11.1 on page 196 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 11.1 on page 196. 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;
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.
Chapter 12
Creating Lattice Graphs Using the LATTICE Layout

The LAYOUT LATTICE Statement ........................................ 201
Defining a Basic Lattice .................................................. 204
  Setting Grid Dimensions ............................................. 204
  Setting Gutter ............................................................ 206
  Defining Cells ............................................................ 206
  Adding Cell Headers ................................................... 208
Managing Axes in LATTICE Layouts .................................... 210
  Internal Axes .............................................................. 210
  Uniform Axis Ranges .................................................. 211
  External Axes ............................................................. 212
Adjusting the Sizes of Rows and Columns .............................. 216
Adjusting the Graph Size .................................................. 218
Examples: Lattice Layout .................................................. 221
  Example 1: Basic Lattice with Cell Axes ........................... 221
  Example 2: Lattice with Side Bars .................................. 225
  Example 3: Lattice with External Axes ............................. 227
  Example 4: Lattice with Row Headers ............................... 228
  Example 5: Lattice with Custom Row Sizing ........................ 230

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 12.1 on page 202 shows a four-cell grid (two rows and two columns). It was produced with a LATTICE layout to illustrate the features of this layout type. The figure contains definitions of four plots, which by default are treated independently.

A mixture of plot types or nested layouts could be used in the cells of the lattice. By default, each plot manages its own axes internal to the lattice boundaries. In the figure, a light gray border has been added to each plot to show its boundaries within the lattice. The shaded areas represent the optional features that you can add to the lattice definition. By default, these shaded areas are not used in the lattice and space is not reserved for them. Thus, in the default case, the plot areas would expand to replace the shaded areas in the cells.

Figure 12.1  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 12.2 on page 203 shows how the lattice would look if you used additional options to externalize the axes. The figure externalizes both the row and column axes, but you could externalize the axes only for the rows, or only for the columns. When axes are external to the cells, the scale of the data ranges that are displayed for the plots are always unified in some form. Unifying the scale of the data ranges means taking the minimum of all data minima and the maximum of all data maxima from a set of plots.

The following variations are available for unifying the axes:

• The scale of the data ranges of all X-axes in a column can be unified on a per-column basis or unified across all columns. (See "Column 1 Axis" and "Column 2 Axis" in Figure 12.2 on page 203.)

• 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 12.2 on page 203.)

By default, external axes are displayed only on the primary axes (bottom and left). They are not displayed on the secondary axes (top and right) unless requested. Notice that external axes use less space and result in larger plot areas than internal axes. (Compare Figure 12.2 on page 203 with Figure 12.1 on page 202, which is the same size.)

Figure 12.2  LATTICE Layout with External Axes
The following example shows a very simple LATTICE layout:

```sas
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 stand-alone plot statements in the layout block.

---

**Defining a Basic Lattice**

**Setting Grid Dimensions**

Assume you want a grid of five plots. Before starting to write code, you must first decide what grid dimensions you want to set (how many columns and rows) and whether you want to permit an empty cell in the grid. If you do not want an empty cell, you must limit the grid to five cells, which gives you two choices for the grid dimensions: five columns by one row (5x1), or one column by five rows (1x5).

To specify grid size, you use the ROWS= and COLUMNS= options in the LAYOUT LATTICE statement. These options can be used in three ways to create the following grid, which contains one row and five columns:
Defining a Basic Lattice

layout lattice / columns=5 rows=1;
    /* plot definitions */
endlayout;

This makes the grid size explicit.

layout lattice / order=columnmajor rows=1;
    /* plot definitions */
endlayout;

To specify only one grid row, also specify ORDER=COLUMNMAJOR. In this case, there 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 definitions */
endlayout;

When only the number of columns is specified, you place a limit on how many plots can appear in one row. If you were to include more than five plot definitions, additional rows (with five columns) would be added automatically because ORDER=ROWMAJOR by default.

If you are willing to have an empty cell in the grid, you could use a 2x3 or a 3x2 grid:

layout lattice / columns=3 rows=2;
endlayout;

If you are willing to have an empty cell in the grid, you could use a 2x3 or a 3x2 grid:

layout lattice / columns=3 rows=2;
endlayout;

Note: The LAYOUT LATTICE statement honors the full specification of columns and rows, unlike the LAYOUT GRIDDED statement, which honors only COLUMNS= or ROWS=, depending on the ORDER= setting.

By default, the layout uses the ORDER=ROWMAJOR setting to populate grid cells. This specification essentially means "fill in all cells in the top row (starting at the top left) and then continue to the next row below":

layout lattice / columns=3 rows=2;
    /* plot1 definition */
    /* plot2 definition */
    /* plot3 definition */
    /* plot4 definition */
    /* plot5 definition */
endlayout;

Alternatively, you can specify ORDER=COLUMNMAJOR, which means "fill in all cells in the left column and then continue to the next column to the right":

layout lattice / columns=3 rows=2 order=columnmajor;

Plot 1  |  Plot 2  |  Plot 3
---|---|---
Plot 4 | Plot 5 | Empty
Setting Gutter

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.

```plaintext
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 218.

Defining Cells

Several valid techniques are available for indicating the contents of a cell:

<table>
<thead>
<tr>
<th>Technique</th>
<th>Example</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>stand-alone plot statement</td>
<td><code>scatterplot x= y=</code>;</td>
<td>simplicity</td>
<td>cannot adjust axes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cannot have overlays</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cannot have cell headers</td>
</tr>
<tr>
<td>Technique</td>
<td>Example</td>
<td>Advantages</td>
<td>Disadvantages</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>layout block</td>
<td>layout overlay; scatterplot x= y=; seriesplot x= y=; endlayout;</td>
<td>cell can contain a complex plot</td>
<td>cannot have cell headers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axes can be adjusted</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>other layout types can be used</td>
<td></td>
</tr>
<tr>
<td>cell block</td>
<td>cell; layout overlay; scatterplot x= y=; seriesplot x= y=; endcell;</td>
<td>makes it easy to see cell boundary in code</td>
<td>adds to program length when no cell header is desired</td>
</tr>
<tr>
<td></td>
<td></td>
<td>required if a cell header is desired</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Here is a simple example:

```plaintext
proc template;
define statgraph lattice;
begingraph;
  entrytitle "Simple 3x2 Lattice with Five Cells Populated";
  layout lattice / columns=3 rows=2 columngutter=10 rowgutter=10;
   /* stand-alone plot statements define cells 1-3 */
  boxplot x=sex y=age;
  boxplot x=sex y=height;
  boxplot x=sex y=weight;
   /* overlay blocks define cells 4-5 */
  layout overlay;
    scatterplot y=weight x=height;
    pbsplineplot y=weight x=height;
    entry halign=right "Spline" / valign=bottom;
  endlayout;
  layout overlay;
    scatterplot y=weight x=height;
    loessplot y=weight x=height;
    entry halign=right "Loess " / valign=bottom;
  endlayout;
  endlayout;
endgraph;
end;
run;
```

```plaintext
proc sgrender data=sashelp.class template=lattice;
run;
```
In the examples shown to this point, a LATTICE layout produces the same result as a GRIDDED layout. We can now look at features that are not available with the GRIDDED layout.

Adding Cell Headers

To add cell headers to the grid, you must specify a CELL block that contains a nested CELLHEADER block. The CELLHEADER block can contain one or more ENTRY statements, or it can contain other statements (DISCRETELEGEND, for example). Here is an example.

```sas
proc template;
define statgraph lattice2;
begingraph / designwidth=495px designheight=230px;
entrytitle "Simple 3x2 Lattice with Five Cells Populated";
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;
  endlayout;
endcell;
cell;
  cellheader;
    entry "Spline Fit";
  endcellheader;
  layout overlay;
    scatterplot y=weight x=height;
  endlayout;
endcell;
cell;
  cellheader;
    entry "Loess Fit";
  endcellheader;
  layout overlay;
    scatterplot y=weight x=height;
  endlayout;
endcell;
cell;
  cellheader;
    entry "Spline Fit";
  endcellheader;
  layout overlay;
    scatterplot y=weight x=height;
  endlayout;
endcell;
cell;
endgraph;
end;
```

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.
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 like LISTING and JOURNAL, the background is white.

As a final enhancement, you could coordinate the text color for the cell headers with a visual attribute in the plot. For example, if you are displaying a fit plot in the cell, you could set the text color to match the color of the fit line. The TEXTATTRA$S= option in the ENTRY statement can be used to set the text properties. The default settings for TEXTATTRA$S= are derived from the GraphValueText style element. For more information about ENTRY statements, see Chapter 15, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 291.
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;
```
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 block specifies UNIONALL for the X axis and UNION for the Y axis.

```plaintext
layout lattice / columns=2 columngutter=5px
columndatarange=unionall
rowdatarange=union;
histogram mpg_city;
histogram mpg_highway;
endlayout;
```
Here is example output.

![Histogram Example](image)

**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 type axis so that the axis tick values can be uniform, even though they might not be at bin midpoints or boundaries for all histograms.

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, the 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.

**External Axes**

**Specifying External Axes**

Whenever axis scales have been unified for a row or a column, you can replace the individual cell axes in that row or column with a single axis that is external to the cells.

To externalize the X axis, use the following syntax:

```
COLUMNAXES;
    COLUMNAXIS / options;
    <COLUMNAXIS / options;>
ENDCOLUMNAXES;
```

To externalize the Y axis, use the following syntax:
ROWAXES;
   ROWAXIS / options;
   <ROWAXIS / options;>
ENDROWAXES;

Within the axes blocks, you should specify as many COLUMNAXIS or ROWAXIS statements as there are columns or rows in the grid. The options that are available to each statement are similar to those that are available for the XAXISOPTS= and YAXISOPTS= options of a LAYOUT OVERLAY statement. The options that you specify can differ from statement to statement.

Note: When a row or column external axis is used, all axis options on the internal axes in that same dimension 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.

![External Axis](image)

**Displaying External 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 external secondary axes are duplicates of the external 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 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 on the DISPLAYSECONDARY= option.

```
layout lattice / columns=2 columngutter=5px
columndatarange=unionall
```
External Axes and Empty Cells

If a LATTICE layout generates empty cells and there are external axes, a row or column axis might be displayed near one or more of those empty cells. The following example shows the default case:

```
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;
   endgraph;
endproc;
```
Adding the `SKIPEMPTYCELLS=TRUE` setting to the LAYOUT LATTICE statement as shown in the following layout block eliminates the space that is normally reserved for the empty cells.

```sas
layout lattice / columns=2 rows=2
  rowgutter=5px columngutter=5px
  rowdatarange=unionall columndatarange=unionall
  skipemptycells=true;
```

Adding the `SKIPEMPTYCELLS=TRUE` setting to the LAYOUT LATTICE statement as shown in the following layout block eliminates the space that is normally reserved for the empty cells.
In this case, an external axis that might be displayed near an empty cell is displayed near a populated cell instead. Here is example output.

**Adjusting the Sizes of Rows and Columns**

By default, the rows and columns of the lattice are of the same depth and width. You can use the ROWWEIGHTS= and COLUMNWEIGHTS= options on the LAYOUT LATTICE statement to designate different row depths or column widths or both. Consider the following settings:

```
LAYOUT LATTICE / ROW=2 COLUMNS=2
   ROWWEIGHTS=(.6 .4) COLUMNWEIGHTS=(.45 .65);
```

Figure 12.3 on page 217 uses these settings. The ROWWEIGHTS= setting specifies that the first row gets 60% of available row space, and the second row gets 40%. The COLUMNWEIGHTS= setting specifies that the first column gets 45% of available column space, and the second column gets 65%. Potentially, the settings on these options affect the space that is allocated to cell headers and to row and column headers.
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 depths for the graph in Figure 12.8 on page 230:

```latex
layout lattice / columns=2 rows=2 rowweights=(.6 .4)
rowdatarange=union columndatarange=union
rowgutter=3px columngutter=3px;
```
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 12.1  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;
```
The graph has a default aspect ratio of 4:3. It would look something like this:

The graph would look better if the graph's height were smaller in relation to its width. You can establish a good default graph size in the template definition by setting the DESIGNWIDTH= and DESIGNHEIGHT= options in the BEGINGRAPH statement. After some experimentation, you might decide that a 2:1 aspect ratio looks good:

begingraph / designwidth=460px designheight=230px;
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.

```plaintext
ods graphics / reset width=430px;
proc sgrender data=sashelp.cars template=fitcompare;
run;
```

If you provide both the HEIGHT= and WIDTH= options in the ODS GRAPHICS statement, you completely override the design aspect ratio. If the WIDTH= or HEIGHT= options are not specified, the design size is in effect.

Setting the DESIGNHEIGHT= and DESIGNWIDTH= options is highly recommended for all multi-cell layouts that contain plots. This recommendation applies to the GRIDDED, LATTICE, DATAPANEL, and DATALATTICE layouts.
Examples: Lattice Layout

Example 1: Basic Lattice with Cell Axes

About This Example
This example demonstrates how to generate the basic lattice shown in Figure 12.5 on page 221. To generate the basic lattice, you must complete the following steps:

- transform the input data
- create the basic lattice
- customize the cell axes

Here is the output.

Figure 12.5 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 do any or all of the following tasks:

- summarizing the data
- transposing the data
- scaling the data values
creating new variables that represent subsets of the data

The graph that is shown in Figure 12.5 on page 221 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:

```sas
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;
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.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01DEC05</td>
<td>.</td>
<td>$26</td>
<td>.</td>
<td>62.8924</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>01NOV05</td>
<td>.</td>
<td>$27</td>
<td>.</td>
<td>71.4692</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>03OCT05</td>
<td>.</td>
<td>$25</td>
<td>.</td>
<td>72.1325</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>01SEP05</td>
<td>.</td>
<td>$25</td>
<td>.</td>
<td>66.9765</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>01AUG05</td>
<td>.</td>
<td>$27</td>
<td>.</td>
<td>65.5300</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>01JUL05</td>
<td>.</td>
<td>$25</td>
<td>.</td>
<td>69.0466</td>
<td>.</td>
</tr>
<tr>
<td>7</td>
<td>01JUN05</td>
<td>.</td>
<td>$25</td>
<td>.</td>
<td>62.9567</td>
<td>.</td>
</tr>
<tr>
<td>8</td>
<td>02MAY05</td>
<td>.</td>
<td>$25</td>
<td>.</td>
<td>62.6998</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>01APR05</td>
<td>.</td>
<td>$25</td>
<td>.</td>
<td>77.0902</td>
<td>.</td>
</tr>
<tr>
<td>10</td>
<td>01MAR05</td>
<td>.</td>
<td>$24</td>
<td>.</td>
<td>72.8997</td>
<td>.</td>
</tr>
<tr>
<td>11</td>
<td>01FEB05</td>
<td>.</td>
<td>$25</td>
<td>.</td>
<td>75.9923</td>
<td>.</td>
</tr>
<tr>
<td>12</td>
<td>03JAN05</td>
<td>.</td>
<td>$26</td>
<td>.</td>
<td>79.6428</td>
<td>.</td>
</tr>
<tr>
<td>13</td>
<td>01DEC04</td>
<td>.</td>
<td>$26</td>
<td>.</td>
<td>84.4881</td>
<td>.</td>
</tr>
<tr>
<td>14</td>
<td>01NOV04</td>
<td>.</td>
<td>$26</td>
<td>.</td>
<td>86.4461</td>
<td>.</td>
</tr>
<tr>
<td>15</td>
<td>01OCT04</td>
<td>.</td>
<td>$25</td>
<td>.</td>
<td>65.7429</td>
<td>.</td>
</tr>
<tr>
<td>16</td>
<td>01SEP04</td>
<td>.</td>
<td>$24</td>
<td>.</td>
<td>57.7253</td>
<td>.</td>
</tr>
<tr>
<td>17</td>
<td>02AUG04</td>
<td>.</td>
<td>$24</td>
<td>.</td>
<td>52.1046</td>
<td>.</td>
</tr>
<tr>
<td>18</td>
<td>01JUL04</td>
<td>.</td>
<td>$25</td>
<td>.</td>
<td>76.6667</td>
<td>.</td>
</tr>
<tr>
<td>19</td>
<td>01JUN04</td>
<td>.</td>
<td>$25</td>
<td>.</td>
<td>77.0683</td>
<td>.</td>
</tr>
<tr>
<td>20</td>
<td>03MAY04</td>
<td>.</td>
<td>$23</td>
<td>.</td>
<td>58.9425</td>
<td>.</td>
</tr>
<tr>
<td>21</td>
<td>01APR04</td>
<td>.</td>
<td>$23</td>
<td>.</td>
<td>77.3867</td>
<td>.</td>
</tr>
<tr>
<td>22</td>
<td>01MAR04</td>
<td>.</td>
<td>$22</td>
<td>.</td>
<td>77.1139</td>
<td>.</td>
</tr>
<tr>
<td>23</td>
<td>02FEB04</td>
<td>.</td>
<td>$23</td>
<td>.</td>
<td>57.3859</td>
<td>.</td>
</tr>
<tr>
<td>24</td>
<td>03JAN04</td>
<td>.</td>
<td>$24</td>
<td>.</td>
<td>63.6359</td>
<td>.</td>
</tr>
</tbody>
</table>

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:

```plaintext
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;
```
Figure 12.6  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 cell basis instead of using external 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 external 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:

```sas
proc template;
define statgraph latticebasic;
begingroup;
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 */
```
Example 2: Lattice with Side Bars

This example demonstrates how to use side bars to enhance the graph in Figure 12.7 on page 228. 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. We should also explain 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, we must define legend text on two of the plot statements, and add a DISCRETELEGEND statement to a side bar along the bottom. Here is the modified template code.

```plaintext
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)
endgraph;
run;
```

Figure 12.5 on page 221 shows the output for the final basic lattice.
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 we 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 External Axes**

Figure 12.6 on page 224 would benefit from externalizing the X and Y axes because the external axes reduces the redundant X axis information and unify the data ranges in the Y axes. We would also like to add grid lines to all axes. To conserve space along the X axes, the automatic formatting of each TIME axis is turned off in the following template code. The `TICKVALUEFORMAT=MONNAME1` setting indicates how to format the time axis tick values.

```sas
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;
run;
```

```sas
proc sgrender data=stock template=latticeexternalaxes;
run;
```
Here is the output.

**Figure 12.7  Lattice with External Axes**

Example 4: Lattice with Row Headers

In “Example 2: Lattice with Side Bars” on page 225, we used a legend in a sidebar 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 GRIDDED layouts, we can have multiple lines of text split exactly where we want and in any text style that we 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, we can rotate the row header text to make it appear to be a row axis label as shown in the following modified ROWHEADERS block:

```plaintext
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 we must specify COLUMNS=2 for the GRIDDED layouts. Here is the final graph.
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 how to use the ROWWEIGHTS= and COLUMNWEIGHTS= options to size the rows. Figure 12.9 on page 231 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 ROWWEIGHTS= 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 12.9  Graph with ROWEIGHTS=PREFERRED

![Graph showing the distribution of cholesterol from the Framingham Heart Study](image)

```
proc template;
define statgraph distribution;
begingraph;
  entrytitle "Distribution of Cholesterol";
  entryfootnote halign=left "From Framingham Heart Study (SASHELP.HEART)";
  layout lattice / rowweights=PREFERRED
columndata=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 26, “Using Dynamics and Macro Variables in Your Templates,” on page 555.
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 Statement” in SAS Graph Template Language: Reference.
# Chapter 13

## Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Classification Panels</td>
<td>236</td>
</tr>
<tr>
<td>The LAYOUT DATALATTICE Statement</td>
<td>236</td>
</tr>
<tr>
<td>The LAYOUT DATAPANEL Statement</td>
<td>237</td>
</tr>
<tr>
<td>The LAYOUT PROTOTYPE Statement</td>
<td>238</td>
</tr>
<tr>
<td>Distinction between DATAPANEL and DATALATTICE</td>
<td>239</td>
</tr>
<tr>
<td>Organizing Panel Contents</td>
<td>239</td>
</tr>
<tr>
<td>Overview of What to Consider When Planning a Classification Panel</td>
<td>239</td>
</tr>
<tr>
<td>Grid Dimensions and Cell Population Order</td>
<td>240</td>
</tr>
<tr>
<td>Gutters</td>
<td>242</td>
</tr>
<tr>
<td>Graph Aspect Ratio</td>
<td>243</td>
</tr>
<tr>
<td>Cell Size</td>
<td>244</td>
</tr>
<tr>
<td>Prototype Orientation</td>
<td>246</td>
</tr>
<tr>
<td>Managing Axes in DATALATTICE and DATAPANEL Layouts</td>
<td>247</td>
</tr>
<tr>
<td>Controlling Data Ranges of Rows or Columns</td>
<td>247</td>
</tr>
<tr>
<td>Setting Axis Options</td>
<td>249</td>
</tr>
<tr>
<td>Controlling the Classification Headers</td>
<td>251</td>
</tr>
<tr>
<td>Using Sidebars</td>
<td>255</td>
</tr>
<tr>
<td>Controlling the Interactions of Classifiers</td>
<td>256</td>
</tr>
<tr>
<td>Appearance of the Last Panel</td>
<td>256</td>
</tr>
<tr>
<td>User Control of Panel Generation</td>
<td>260</td>
</tr>
<tr>
<td>Sparse Data</td>
<td>264</td>
</tr>
<tr>
<td>Missing Class Values</td>
<td>267</td>
</tr>
<tr>
<td>Using Non-computed Plots in Classification Panels</td>
<td>269</td>
</tr>
<tr>
<td>Adding an Inset to Each Cell</td>
<td>271</td>
</tr>
<tr>
<td>Using PROC SGPANEL to Create Classification Panels</td>
<td>271</td>
</tr>
<tr>
<td>Examples: Data Lattice Layout and Data Panel Layout</td>
<td>273</td>
</tr>
<tr>
<td>Example 1: A Basic Data Lattice</td>
<td>273</td>
</tr>
<tr>
<td>Example 2: A Basic Data Panel</td>
<td>275</td>
</tr>
<tr>
<td>Example 3: A Data Panel with Sidebars</td>
<td>277</td>
</tr>
<tr>
<td>Example 4: A Data Panel with an Inset in Each Cell</td>
<td>280</td>
</tr>
</tbody>
</table>
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 13.1 on page 237 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.
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 Statement” 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).
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 Statement” 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 269 for examples of how to work around this limitation.

- DISCRETELEGEND, CONTINUOUSLEGEND, ENTRY, ENTRYTITLE, and ENTRYFOOTNOTE 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 277.

- 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 247.

**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.

```plaintext
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.

```plaintext
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 produce 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
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 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). 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.

<table>
<thead>
<tr>
<th>var = value 1</th>
<th>var = value 2</th>
<th>var = value 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plot 1</td>
<td>Plot 2</td>
<td>Plot 3</td>
</tr>
<tr>
<td>var = value 4</td>
<td>var = value 5</td>
<td></td>
</tr>
<tr>
<td>Plot 4</td>
<td>Plot 5</td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>var = value 1</th>
<th>var = value 3</th>
<th>var = value 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plot 1</td>
<td>Plot 3</td>
<td>Plot 5</td>
</tr>
<tr>
<td>var = value 2</td>
<td>var = value 4</td>
<td></td>
</tr>
<tr>
<td>Plot 2</td>
<td>Plot 4</td>
<td></td>
</tr>
</tbody>
</table>

One last variation is to specify START=BOTTOMLEFT which produces the following grids, depending on the setting for the ORDER= option:

```plaintext
layout datapanel classvars=(var) / columns=3 rows=2 start=bottomleft;
layout prototype;
  ... plot statements ...
endlayout;
endlayout;
```

Here is the resulting grid layout.

<table>
<thead>
<tr>
<th>var = value 4</th>
<th>var = value 5</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Plot 4</td>
<td>Plot 5</td>
<td></td>
</tr>
<tr>
<td>var = value 1</td>
<td>var = value 2</td>
<td>var = value 3</td>
</tr>
<tr>
<td>Plot 1</td>
<td>Plot 2</td>
<td>Plot 3</td>
</tr>
</tbody>
</table>

```plaintext
layout datapanel classvars=(var) / columns=3 rows=2 order=columnmajor start=bottomleft;
layout prototype;
  ... plot statements ...
endlayout;
endlayout;
```

Here is the resulting grid layout.

<table>
<thead>
<tr>
<th>var = value 2</th>
<th>var = value 4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Plot 2</td>
<td>Plot 4</td>
<td></td>
</tr>
<tr>
<td>var = value 1</td>
<td>var = value 3</td>
<td>var = value 5</td>
</tr>
<tr>
<td>Plot 1</td>
<td>Plot 3</td>
<td>Plot 5</td>
</tr>
</tbody>
</table>

*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 256.

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:

```plaintext
layout datapanel classvars=(A B) / columns=3 rows=2;
```

Here is the resulting grid layout.

<table>
<thead>
<tr>
<th>A = a1</th>
<th>A = a1</th>
<th>A = a1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B = b1</td>
<td>B = b2</td>
<td>B = b3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Plot 1</th>
<th>Plot 2</th>
<th>Plot 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = a2</td>
<td>A = a2</td>
<td>A = a2</td>
</tr>
<tr>
<td>B = b1</td>
<td>B = b2</td>
<td>B = b3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Plot 4</th>
<th>Plot 5</th>
<th>Plot 6</th>
</tr>
</thead>
</table>

**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.

```plaintext
layout lattice classvars=(var) / columns=3 rows=2
columngutter=5 rowgutter=5;
layout prototype;
... plot statements ...
endlayout;
endlayout;
```

Here is the resulting grid layout.

<table>
<thead>
<tr>
<th>var = value 1</th>
<th>var = value 2</th>
<th>var = value 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plot 1</td>
<td>Plot 2</td>
<td>Plot 3</td>
</tr>
<tr>
<td>var = value 4</td>
<td>var = value 5</td>
<td></td>
</tr>
<tr>
<td>Plot 4</td>
<td>Plot 5</td>
<td></td>
</tr>
</tbody>
</table>

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 13.1 on page 243 defines a three column by one row grid.

**Example Code 13.1 3x1 Panel Template**

```sas
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.

```sas
proc sgrender data=sashelp.orsales template=onerow;
  where product_group in {"Golf" "Tennis" "Soccer"};
run;
```

![Yearly Profit for Sports Products](image)

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 13.1 on page 243 with a 2:1 aspect ratio, modify the BEGINGRAPH statement as follows:

```begingroup / designwidth=640px designheight=320px;
```

Here is the result.

![Yearly Profit for Sports Products](image)

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 13.1 on page 243 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 251.)

For example, to render the panel in Example Code 13.1 on page 243 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.

![Yearly Profit for Sports Products](image)

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. The GTL has an internal restriction on how small a cell in the panel can be: 100 pixels by 100 pixels. Cell size is computed after all titles, footnotes, and sidebar contents have been established. Thus, if we had additional titles in the panel design, log messages similar to the one just shown would be issued, even with a larger panel size.

The CELLWIDTHMIN= and CELLHEIGHTMIN= options on the LAYOUT DATAPANEL or LAYOUT DATALATTICE statements can be used to specify 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;
```
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;
   endgraph;
end;
```
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 13.2 Classification Panel Template

```sas
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;
```
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 COLUMNNDATARANGE=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.

![Yearly Profit for Sports Products](image)

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.

```plaintext
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.

```plaintext
layout datalattice rowvar=product_group / 
  rowdatarange=union 
  rowgutter=5px 
  columnaxisopts=(display=(tickvalues)) 
  rowaxisopts=(display=(tickvalues) 
    linearopts= (tickvalueformat=dollar12.) 
    griddisplay=on offsetmin=0);
```
Controlling the Classification Headers

In many cases, it is not necessary to display the classification-variable name in the classification headers. Often, just the classification value is sufficient. Both the DATALATTICE and DATAPANEL layouts support the HEADERLABELDISPLAY= option. By default, HEADERLABELDISPLAY=NAMEVALUE, which 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;
begin graph / designwidth=350px designheight=400px;
entrytitle "Yearly Profit for Sports Products";
layout datalattice rowvar=product_group /
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 HEADERBACKGROUND COLOR= 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.

```plaintext
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.

![Yearly Profit for Sports Products](image)

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.

```plaintext
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;
```
Starting with the first maintenance release of SAS 9.4, 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) /
      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.
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 Statement” in SAS Graph Template Language: Reference or “LAYOUT DATAPANEL Statement” 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 13.1 on page 237 and Figure 13.2 on page 238. 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:
Controlling the Interactions of Classifiers

Whenever you have classifiers with a large number of unique levels, the potential exists for generating a large number of cells in the panel. If you do not want to see all classification levels, you can limit the crossings by using a WHERE expression when creating the input data. Or, you can use a WHERE expression as part of the PROC SGRENDER step that renders the graph.

Appearance of the Last Panel

If you set the ROWS= and COLUMNS= options to define a relatively small grid, PROC SGRENDER automatically generates as many separate panels as it takes to exhaust all the classification levels. Depending on the grid size and total number of classification levels, one or more empty cells might be created on the last panel to complete the grid. For example, if there are seven classification levels and you define a 2x2 grid, two panels are created (with different names), and the last panel contains one empty cell as shown in the following example.

```sas
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.

```plaintext
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.

![Profit for Selected Sports Items](image)

The SKIPEMPTYCELLS= option also applies to a DATALATTICE layout. The following output shows the last panel when Division has two levels and Product has three levels, while ROWS=2 and COLUMNS=2. When SKIPEMPTYCELLS=FALSE, the last panel will have a column of empty cells. Entire rows or columns of empty cells can be removed by setting SKIPEMPTYCELLS=TRUE.
**User Control of Panel Generation**

It is possible to control the generation of panels. Consider the following output, in which each panel displays in its upper right corner the current panel number and the total number of panels:
Controlling the Interactions of Classifiers

Panel 1 of 3

Office Furniture Sales for 1994

<table>
<thead>
<tr>
<th>Product</th>
<th>Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOFA</td>
<td>EDUCATION</td>
</tr>
<tr>
<td>SOFA</td>
<td>CONSUMER</td>
</tr>
</tbody>
</table>

Actual Sales

Month

Panel 2 of 3

Office Furniture Sales for 1994

<table>
<thead>
<tr>
<th>Product</th>
<th>Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>BED</td>
<td>EDUCATION</td>
</tr>
<tr>
<td>BED</td>
<td>CONSUMER</td>
</tr>
<tr>
<td>TABLE</td>
<td>EDUCATION</td>
</tr>
<tr>
<td>TABLE</td>
<td>CONSUMER</td>
</tr>
</tbody>
</table>

Actual Sales

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.

```plaintext
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)
panganese=PANELNUM;
  layout prototype;
    seriesplot x=month y=actual / lineattrs=GraphData1;
  endlayout;
  sidebar / align=top;
    entry "Office Furniture Sales for " YEAR /
      textattrs=GraphTitleText;
  endsidebar;
endgraph;
enddefine;
endproc;
```

262 Chapter 13 • Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts
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 26, “Using Dynamics and Macro Variables in Your Templates,” on page 555.

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:

```plaintext
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;
```
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;
```
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.
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.

```plaintext
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 13.3 on page 268 when these formats are applied to the classification variables.
Note: In the second maintenance release of SAS 9.4 and in earlier releases, ODS Graphics does not support Unicode values in user-defined formats. Starting with the third maintenance release of SAS 9.4, 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 122.

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.

```sas
proc template;
define statgraph dosepanel;
begingroup /
designwidth=490px
designheight=350px;
layout datpanel classvars=(dose) / rows=1;
```

Predicted Response to Dosage (mg) over Time

<table>
<thead>
<tr>
<th>Dose = 100</th>
<th>Dose = 200</th>
<th>Dose = 300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Days</td>
<td>Days</td>
<td>Days</td>
</tr>
<tr>
<td>0 5 10 15 20 25 30</td>
<td>0 5 10 15 20 25 30</td>
<td>0 5 10 15 20 25 30</td>
</tr>
</tbody>
</table>

Response

50 55 60

Days

0 5 10 15 20 25 30

Fit  95% Confidence Limits  95% Prediction Limits
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 the first maintenance release of SAS 9.4, 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 280.

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 269, 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;
```
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:

```sas
title "Cholesterol Distribution by Gender and Weight";
proc sgpanel data=sashelp.heart;
panelby sex weight_status / layout=lattice onepanel novarnames;
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.

![Graph showing cholesterol distribution by gender and weight](image-url)
Here is the output.

![Cholesterol Distribution by Gender and Weight](image)

**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.

```plaintext
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;
```
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.

```plaintext
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;
dendatastyle;
dendatastoplet;
dendatastyle;
endgraph;
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 13.5** 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
```

---

Here is the output.
proc sgrender data=sashelp.prdsale template=sidebar;
  where country="U.S.A." and region="EAST" and
  product in ("CHAIR" "DESK" "TABLE");
run;
Office Furniture Sales

Examples: Data Lattice Layout and Data Panel Layout
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:

![Predicted Response to Dosage (mg) over Time](image)

Here is the template code.

```sas
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;
```
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 369 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.
Chapter 14
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 18, “Adding Code-Driven Graphics Elements to Your Graph,” on page 393.

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.

```sas
/* 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
    endcell;
  endlayout;
  endgraph;
enddefine;
run;
```
/* Generate a pie chart of percent of total sales. */
cell;
  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.

```sas
/* 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;
```
Part 4

Text, Legends, and Insets

Chapter 15
Adding Titles, Footnotes, and Text Entries to Your Graph ........ 291

Chapter 16
Adding Legends to Your Graph ........................................... 309

Chapter 17
Adding Insets to Your Graph ................................................. 351
Chapter 15
Adding Titles, Footnotes, and Text Entries to Your Graph

Text Strings in Graphs ......................................................... 291
Text Properties and Syntax Conventions .................................. 293
Text Statement Basics ............................................................ 295
  Using Titles and Footnotes ................................................... 295
  Using Text Entries in the Graphical Area .................................. 296
Managing the String on Text Statements .................................. 297
  Text Statement Syntax ......................................................... 297
  Using Rich Text ................................................................. 297
  Horizontally Aligning Text Items ........................................... 298
  Generating Text Items with Dynamic Variables, Macro
    Variables, and Expressions ............................................... 298
  Adding Subscripts, Superscripts, and Unicode Rendering ................. 299
  Using Unicode Values in Labels ............................................ 300
Using Options on Text Statements .......................................... 301
  Options Available on All Text Statements ................................. 301
  Setting Text Background, Borders, and Padding ....................... 301
  Managing Long Text in Titles and Footnotes ............................. 303
ENTRY Statements: Additional Control .................................... 304
  Features Available for ENTRY Text ......................................... 304
  Positioning ENTRY Text ..................................................... 305
  Rotating ENTRY Text ....................................................... 306

Text Strings in Graphs

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>label data points</td>
<td>plot statements that display markers</td>
<td>DATALABEL=column</td>
</tr>
<tr>
<td>label a curve or a reference line</td>
<td>plot statements that display lines</td>
<td>CURVELABEL=&quot;string&quot;</td>
</tr>
<tr>
<td>describe a plot in a legend</td>
<td>most plot statements</td>
<td>LEGENDLABEL=&quot;string&quot;</td>
</tr>
<tr>
<td>add title to a legend</td>
<td>legend statements</td>
<td>TITLE=&quot;string&quot;</td>
</tr>
<tr>
<td>label an axis</td>
<td>axis statement or layout axis option</td>
<td>LABEL=&quot;string&quot;</td>
</tr>
</tbody>
</table>

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

ENTRYTITLE "string"  Defines title text for the entire graph.

ENTRYFOOTNOTE "string"  Defines footnote text for the entire graph.

ENTRY "string"  Defines text that is displayed in the graphical area.
This chapter focuses primarily on how to set text properties for any text. Additional information 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 85.
- For managing the text in legends, see Chapter 16, “Adding Legends to Your Graph,” on page 309.
- For managing the text in insets, see Chapter 17, “Adding Insets to Your Graph,” on page 351.
- For managing the text in multi-cell layouts, see Chapter 12, “Creating Lattice Graphs Using the LATTICE Layout,” on page 201 and Chapter 13, “Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts,” on page 235.

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:

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Option</th>
<th>Supporting Text Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>plot statements</td>
<td>DATALABEL=</td>
<td>DATALABELATTRS=</td>
</tr>
<tr>
<td></td>
<td>CURVELABEL=</td>
<td>CURVELABELATTRS=</td>
</tr>
<tr>
<td>legend statements</td>
<td>TITLE=</td>
<td>TITLEATTRS=</td>
</tr>
<tr>
<td>layout or axis statements</td>
<td>LABEL=</td>
<td>LABELATTRS=</td>
</tr>
<tr>
<td>text statements</td>
<td>TEXTATTRS=</td>
<td></td>
</tr>
</tbody>
</table>

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 457 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`:

<table>
<thead>
<tr>
<th>Text Option</th>
<th>Value</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR=</td>
<td>color</td>
<td>style-reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For more information about text-options, see “Text Options” on page 611.

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:

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

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

```plaintext
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:

```plaintext
layout overlay / xaxisopts=( labelattrs=GraphValueText )
               yaxisopts=( labelattrs=GraphValueText );
```
scatterplot x=height y=weight / group=age datalabel=name
dataxlabelatts=( color=GraphDataText:Color );

To ensure that a footnote is displayed in bold italics, you could specify the following:
entryfootnote "Study conducted in 2007" /
textatts=( weight=bold style=italic );

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

---

**Text Statement Basics**

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

**Using Titles and Footnotes**

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

```sas
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;
```

---

![Graph](image-url)
However, the statement placement in the following BEGINGRAPH block yields the same result:

```plaintext
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 531.

### 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:

```plaintext
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;
```
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.

```plaintext
elementfootnote 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.

```plaintext
elementfootnote 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:

```plaintext
elementfootnote "Summary for " eval(put(today(),mmdydyd,));
```

Here is the output.

```
Summary for 02-02-12
```

For more information about dynamic variables, macro variables, and EVAL expressions in GTL, see Chapter 26, “Using Dynamics and Macro Variables in Your Templates,” on page 555 and Chapter 27, “Using Conditional Logic and Expressions in Your Templates,” on page 565.
**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.

\[ R^2 = .457 \]

...for the \( H_2O \) Regression...

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

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

In the code, the \( 03B1 \) 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 \( 03B1 \) is

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

Here is the output.

\[ \alpha = .05 \]

For a complete list of keywords that can be used with the \{unicode keyword\} notation, see “Reserved Keywords and Unicode Values” on page 593. 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.

\[ \bar{X}=6.78 \]
Using Unicode Values in Labels

The {UNICODE}, {SUB}, and {SUP} text commands apply only to the ENTRY, ENTRYTITLE, and ENTRYFOOTNOTE statements. However, strings that are assigned to axis labels, curve labels, legend labels, and so on, can present Unicode characters using what is called "in-line formatting." To use this special formatting, you embed within the string an ODS escape sequence followed by a text command. Specifically, whenever you 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:

```sas
ods escapechar="^"; /* Define an escape character */
proc template;
define statgraph fit;
begingroup;
  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;
endgroup;
end;
run;

proc sgrender data=sashelp.class template=fit;
run;
```
Using Options on Text Statements

Options Available on All Text Statements

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

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

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

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

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

BORDERATTRS= style-element | style-element (line-options) | (line-options)
   Specifies the properties of the border line. For information about line-options, see “Line Options” on page 609.

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 611. 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);
  endgraph;
end;
```

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

Here is the output.
entryfootnote halign=right
"Prepared with SAS® 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/or its font properties cause the text line to exceed the space available for it. The font properties include the font family, font size, and font weight (BOLD or NORMAL). Thus, adjusting the length of the text and/or changing its font properties are adjustments 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:

```sas
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, we know that 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.

```plaintext
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.

![Microsoft Stock Performance](image)

The complete code for this example is shown in “Example 4: Lattice with Row Headers” on page 228.
## 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 309.

For details, see “General Legend Features” on page 315 and “Adding a Discrete Legend” on page 323.

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 309.

For details, see “General Legend Features” on page 315 and “Adding a Continuous Legend” on page 343.

**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;
```
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 326. 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.

```sas
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;
```
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 sort data=sashelp.class out=class;
    by age;
run;
proc sgrender data=class template=order;
run;
```

Introduction to Legend Management
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).

```sas
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 11, “Creating Gridded Graphs Using the GRIDDED Layout,” on page 187 (GRIDDED)
- Chapter 12, “Creating Lattice Graphs Using the LATTICE Layout,” on page 201 (LATTICE)
- Chapter 13, “Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts,” on page 235 (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=CENTE R VALIGN=BOT TOM";
    layout overlay;
      scatterplot X=Height Y=Weight / name="sp" group=sex;
      discretelegend "sp" / location=outside halign=cente r valign=bottom;
    endlayout;
  endgraph;
```
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 328 and “When Discrete Legends Get Too Large” on page 340.

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

```ods html
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, we know that the best position for an inside legend is TOPRIGHT or TOLEFT.

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

```plaintext
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.

BACKGROUND COLOR = 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 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.

```plaintext
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"
```
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;
```
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.

![Example Output](image)

## 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.

```plaintext
proc template;
define statgraph order;
dynamic TITLE;
begingraph;
entrytitle TITLE;
```
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 323 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;
```
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 492.

**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.

```plaintext
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;
endgraph;
end;
```
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.

![Example Graph]

**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, LINEPARAM, REFERENCELINE, and DROPLINE. Additional options are available to control curve label location, position, and text properties. For examples, see Chapter 23, “Managing Your Graph’s Appearance,” on page 445 and Chapter 15, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 291.

### 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.

![Legend with no Wrapping]

**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 | COLUMNMAJOR**

- Determines whether legend entries are wrapped on a column or row basis. Default is ROWMAJOR.

**ACROSS = number**

- Determines the number of columns. Only used with ORDER=ROWMAJOR

**DOWN = number**

- Determines the number of rows. Use only with ORDER=COLUMNMAJOR

**DISPLAYCLIPPED = TRUE | FALSE**

- Determines whether to show a legend when there are too many entries to fit in the available space

### Organizing Legend Entries in a Fixed Number of Columns

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

```proc template;
define statgraph order;
begingraph;
entrytitle "ORDER=ROWMAJOR ACROSS=1";
layout overlay;
```
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;
endgraph;
end;
run;
```

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

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

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."
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=COLUNMMAJOR` 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.

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

**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" </option(s)>

You must place the LEGENDITEM statement in the global definition area of the template between the BEGINGRAPH statement and the first layout 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 Statement” 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.

```sas
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 the third maintenance release of SAS 9.4, 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 <option(s)>
```

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.
- 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 Statement” 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.

```sas
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.

```sas
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;
```

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.

```sas
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;
```
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.

```plaintext
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.

![Legend Drops out with GROUP=NAME](image)

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.

![Legend Cliped](image)

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:

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

Here is example output.

![Legend With Reduced Font Sizes](image)

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.

<table>
<thead>
<tr>
<th>Plot Statement</th>
<th>Enabling Plot Option</th>
<th>Related Plot Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUBBLEPLOT</td>
<td>COLORRESPONSE=</td>
<td>COLORMODEL=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REVERSECOLORMODEL=</td>
</tr>
<tr>
<td>CONTOURPLOTYPARMS</td>
<td>CONTOURTYPE=</td>
<td>COLORMODEL=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REVERSECOLORMODEL=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NLEVELS=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NHINT=</td>
</tr>
<tr>
<td>HEATMAPPARMS</td>
<td>COLORRESPONSE=</td>
<td>COLORMODEL=</td>
</tr>
<tr>
<td>SCATTERPLOT</td>
<td>MARKERCOLORGRADIENT=</td>
<td>COLORMODEL=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REVERSECOLORMODEL=</td>
</tr>
<tr>
<td>SURFACEPLOTYPARMS</td>
<td>SURFACECOLORGRADIENT=</td>
<td>COLORMODEL=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REVERSECOLORMODEL=</td>
</tr>
<tr>
<td>WATERFALLCHART</td>
<td>COLORRESPONSE=</td>
<td>COLORMODEL=</td>
</tr>
</tbody>
</table>

Starting with the third maintenance release of SAS 9.4, the MARKERCOLORGRADIENT= and SURFACECOLORGRADIENT= options are deprecated and replaced with the COLORRESPONSE= option. The MARKERCOLORGRADIENT= and SURFACECOLORGRADIENT= options are still honored, but SAS recommends that you use the COLORRESPONSE= option instead.

Starting with the third maintenance release of SAS 9.4, continuous legends can be used with the following additional plot statements when the enabling plot option is also specified.

<table>
<thead>
<tr>
<th>Plot Statement</th>
<th>Enabling Plot Option</th>
<th>Related Plot Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARCHART</td>
<td>COLORRESPONSE=</td>
<td>COLORMODEL=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COLORBYFREQ=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COLORSTAT=</td>
</tr>
</tbody>
</table>

* In the second maintenance release of SAS 9.4, the MARKERCOLORGRADIENT= and SURFACECOLORGRADIENT= options are deprecated and replaced with the COLORRESPONSE= option. The MARKERCOLORGRADIENT= and SURFACECOLORGRADIENT= options are still honored, but SAS recommends that you use the COLORRESPONSE= option instead.
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.

```sas
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.

![Contour Plot](image1)

Here is the output when CONTOURTYPE=GRADIENT.

![Contour Plot](image2)

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.

```sas
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.

```sas
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" valuescounthint=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 297.

Here is the output.

For more information about surface plots, see Chapter 10, “Creating Overlay 3-D Graphs Using the OVERLAY3D Layout,” on page 167.
When you use V ALIGN=BOTTOM or V ALIGN=TOP instead of the HALIGN= option, the default orientation of the legend automatically becomes ORIENT=HORIZONTAL. Here is the modified CONTINUOUSLEGEND statement:

```plaintext
continuouslegend "surf" /
  title="Temperature (\textdegree F)"
  valign=bottom;
```

Here is the output.

![Surface Color Gradient Legend](image)

**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.

![Conditional Density](image)
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.
Chapter 17
Adding Insets to Your Graph

Uses for Insets in a Graph .................................................. 351
Creating a Simple Inset with an ENTRY Statement .................. 352
Creating an Inset as a Table of Text ................................. 353
Positioning an Inset ......................................................... 356
Creating an Inset with Values That Are Computed in the Template 359
Creating an Inset from Values That Are Passed to the Template . 361
Overview of Importing Data into a Template .......................... 361
Creating a Template That Uses Macro Variables ..................... 362
Defining a Macro to Initialize the Variables and Generate the Graph 363
Executing the Macro .......................................................... 365
Adding Insets to a SCATTERPLOTMATRIX Graph ................... 366
Adding Insets to Classification Panels ................................. 369
About Cell Insets in Classification Panels ............................. 369
Adding Insets By Using the Match-Merging Data Scheme ............ 369
Adding Insets By Using the One-To-One-Merging Data Scheme .... 373
Creating Axis-Aligned Insets ................................................ 377
Creating an Axis-Aligned Inset with a Block Plot .................... 377
Creating an Axis-Aligned Inset with an Axis Table .................. 385
Creating an Axis-Aligned Inset in a Classification Panel .......... 387
Creating an Axis-Aligned Inset with the GTL Annotation Facility ... 389

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 15, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 291 and Chapter 11, “Creating Gridded Graphs Using the GRIDDED Layout,” on page 187.

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.

```plaintext
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;
```
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:

<table>
<thead>
<tr>
<th>N</th>
<th>5203</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>119.96</td>
</tr>
<tr>
<td>Std Dev</td>
<td>19.98</td>
</tr>
</tbody>
</table>

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:

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.
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.

```
<table>
<thead>
<tr>
<th>N</th>
<th>5203</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>119.96</td>
</tr>
<tr>
<td>Std Dev</td>
<td>19.98</td>
</tr>
</tbody>
</table>
```

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.

```plaintext
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.

<table>
<thead>
<tr>
<th>N</th>
<th>5203</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>119.96</td>
</tr>
<tr>
<td>Std Dev</td>
<td>19.98</td>
</tr>
</tbody>
</table>

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.

```plaintext
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.

<table>
<thead>
<tr>
<th>Stat Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>N 5203</td>
</tr>
<tr>
<td>Mean 119.96</td>
</tr>
<tr>
<td>Std Dev 19.98</td>
</tr>
</tbody>
</table>

### 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;
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.

```sas
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;
endgraph;
```
By default, the background of ENTRY statements and a GRIDDED layout are transparent. So if the current style defines a background color and the inset does not appear in the plot wall, the style’s background color is seen through the inset. You can make the background of the inset 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
```

Making the Inset Background Opaque

---

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;
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.

![Two Column Inset with Computed Values](image)

See Chapter 27, “Using Conditional Logic and Expressions in Your Templates,” on page 565 for more information about the functions that can be used in the EVAL function. See Chapter 26, “Using Dynamics and Macro Variables in Your Templates,” on page 555 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 discussion explains how to create and call a macro that can pass a data set name and variable name to a previously compiled GTL template. To follow this discussion, you should understand the topics that are discussed in Chapter 26, “Using Dynamics and Macro Variables in Your Templates,” on page 555.

For this example, we create a macro named HISTOGRAM that takes two arguments.

- The first argument, DSN, passes a data set name.
- The second argument, VAR, passes a variable name.

When invoked, the macro generates a histogram and model fit plot for the analysis variable VAR. The graph also displays two insets that show the related statistics. The following figure shows an example.
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 17.1 on page 362. 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 we name GINSET:

```gtl
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;
endgraph;
enddefine;
```

For more information about macros, see the SAS Macro Language: Reference.
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 we will see later when we create the macro, either of two normality tests will be used, depending on the number of observations that are read from the data. Macro variable TESTLABEL provides either of two test labels, depending on which test is used at 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 362 we created template GINSET, which declared the following macro variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST</td>
<td>Identifies the normality test that is applied to the data.</td>
</tr>
<tr>
<td>TESTLABEL</td>
<td>Provides the label that is associated with the applied normality test.</td>
</tr>
</tbody>
</table>
STAT Provides the normality statistic that is calculated by the applied normality test.

PTYPE and PVALUE Provide the probability (type and value) for the applied normality test.

NOBS, MEAN, and STD Provide the number of observations, mean, and standard deviation for the analysis variable.

To initialize these macro variables, we will now create a macro that calculates values for them and also specifies an SGRENDER procedure that uses template GINSET. The macro needs two parameters: one for passing a SAS data set name, and a second for passing the name of a column in that data set.

The following macro code uses PROC UNIVARIATE to create two output data sets. A DATA step then reads the output data sets, creates the required macro variables, and assigns values to those macro variables in a local symbol table. When the macro runs the SGRENDER procedure, the values of the macro variables are imported into the GINSET template to produce a graph with insets, similar to the graph shown in Figure 17.1 on page 362. 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.

```sas
%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, we must pass it a data set name and the name of a numeric column in the data.

The following macro call passes the data set name Sashelp.Heart and the column name MRW. Because column MRW has more than 2000 observations, the Kolmogorov-Smirnov test is used in the analysis.

```sas
%histogram(sashelp.heart, mrw)
```

The output is shown in Figure 17.1 on page 362.

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.

```sas
%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.)

`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
- `PEARSONPV` 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).

BACKGROUND_COLOR=  
specifies the color of the inset background. Accepts a style reference or a color specification.

BORDER=  
specifies whether a border is displayed around the inset. Accepts TRUE or FALSE (the default).

HALIGN=  
specifies the horizontal alignment of the inset. Accepts keywords LEFT (the default), CENTER, or RIGHT.

OPAQUE=  
specifies whether the inset background is opaque (TRUE) or transparent (FALSE, the default).

TEXTATTRS=  
specifies the text properties of the entire inset.

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

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 13, “Creating Classification Panels Using the DATALATTICE and DATAPANEL Layouts,” on page 235.

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 366. 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 the first maintenance release of SAS 9.4, 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 387. 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.

```sas
/* 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.

```sas
/* 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.

```sas
/* 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;
```
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.

<table>
<thead>
<tr>
<th>classification variables</th>
<th>columnvar=origin rowvar=cylinders</th>
</tr>
</thead>
<tbody>
<tr>
<td>inset information</td>
<td>inset=(nobs mean)</td>
</tr>
<tr>
<td>bar chart data</td>
<td>category=type response=mean</td>
</tr>
</tbody>
</table>

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 DATASCIHEME=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.

```sas
/* Create the graph template for the classification panel. */
proctemplate;
definestatgraphpanel;
begingraph;
entrytitle "Average City MPG for Vehicles";
entrytitle "by Origin, Cylinders and VehicleType";
layoutdatalatticecolumnvar=originrowvar=cylinders/
columndatarange=unionallrowdatarange=unionall
headerlabeldisplay=value
headerbackgroundcolor=GraphAltpBlock:color
inset=(nobs mean)
insetopts=(border=true datascHEME=mATCHed
opaque=true backgroundcolor=GraphAltpBlock:color)
rowaxisopts=(OFFSETMAX=.4 offsetmin=.1
```
Finally, here is the SAS code that renders the graph by using the data in the AvgMileage data set.

```sas
/* 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;
```

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 369.
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.

```sas
/* 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 17.1 Partial Listing of the Mileage Data Set**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Cylinders</th>
<th>Origin</th>
<th>Type</th>
<th><em>TYPE</em></th>
<th>Mean</th>
<th>Nobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>Asia</td>
<td></td>
<td>6</td>
<td>25.8421</td>
<td>57</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Europe</td>
<td></td>
<td>6</td>
<td>23.4444</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>USA</td>
<td></td>
<td>6</td>
<td>24.0000</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>Asia</td>
<td></td>
<td>6</td>
<td>18.3000</td>
<td>60</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>Europe</td>
<td></td>
<td>6</td>
<td>18.7632</td>
<td>38</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>USA</td>
<td></td>
<td>6</td>
<td>18.4754</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>Asia</td>
<td></td>
<td>6</td>
<td>15.2000</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>Europe</td>
<td></td>
<td>6</td>
<td>16.1667</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>USA</td>
<td></td>
<td>6</td>
<td>15.5143</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>Asia</td>
<td>SUV</td>
<td>7</td>
<td>21.4000</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>Asia</td>
<td>Sedan</td>
<td>7</td>
<td>26.5102</td>
<td>49</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>Asia</td>
<td>Truck</td>
<td>7</td>
<td>22.3333</td>
<td>3</td>
</tr>
</tbody>
</table>

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.

```sas
/* 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).

```sas
/* 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 17.2  Partial Listing of the Summary Data Set**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Cylinders</th>
<th>Origin</th>
<th>Type</th>
<th>Mean</th>
<th>Nobs</th>
<th>Cyl</th>
<th>Origin</th>
<th>Mean</th>
<th>Nobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>Asia</td>
<td>SUV</td>
<td>21.4</td>
<td>5</td>
<td>4</td>
<td>Asia</td>
<td>25.8</td>
<td>57</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Asia</td>
<td>Sedan</td>
<td>26.5</td>
<td>49</td>
<td>4</td>
<td>Europe</td>
<td>23.4</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>Asia</td>
<td>Truck</td>
<td>22.3</td>
<td>3</td>
<td>4</td>
<td>USA</td>
<td>24.0</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Europe</td>
<td>SUV</td>
<td>.</td>
<td>0</td>
<td>6</td>
<td>Asia</td>
<td>18.3</td>
<td>60</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Europe</td>
<td>Sedan</td>
<td>23.4</td>
<td>18</td>
<td>6</td>
<td>Europe</td>
<td>18.8</td>
<td>38</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>Europe</td>
<td>Truck</td>
<td>.</td>
<td>0</td>
<td>6</td>
<td>USA</td>
<td>18.5</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>USA</td>
<td>SUV</td>
<td>20.5</td>
<td>2</td>
<td>8</td>
<td>Asia</td>
<td>15.2</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>USA</td>
<td>Sedan</td>
<td>24.7</td>
<td>29</td>
<td>8</td>
<td>Europe</td>
<td>16.2</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>USA</td>
<td>Truck</td>
<td>20.0</td>
<td>3</td>
<td>8</td>
<td>USA</td>
<td>15.5</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>Asia</td>
<td>SUV</td>
<td>17.2</td>
<td>15</td>
<td>.</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>Asia</td>
<td>Sedan</td>
<td>19.0</td>
<td>41</td>
<td>.</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>Asia</td>
<td>Truck</td>
<td>15.5</td>
<td>4</td>
<td>.</td>
<td>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

- **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.

```plaintext
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))
  endgraph;
end;
```
The output is shown in Figure 17.2 on page 373.

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.
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 17.1 SAS Code That Generates the Events Data Set**

```sas
/* 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;
run;
```
by date;
run;

**Output 17.3  Partial Listing of the Events Data Set**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Stock</th>
<th>Date</th>
<th>Close</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Microsoft</td>
<td>01OCT87</td>
<td>$0.30</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Microsoft</td>
<td>02NOV87</td>
<td>$0.27</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Microsoft</td>
<td>01DEC87</td>
<td>$0.33</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>09DEC87</td>
<td></td>
<td>2.0</td>
</tr>
<tr>
<td>19</td>
<td>Microsoft</td>
<td>04JAN88</td>
<td>$0.34</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Microsoft</td>
<td>01FEB88</td>
<td>$0.36</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Microsoft</td>
<td>01MAR88</td>
<td>$0.34</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Microsoft</td>
<td>04APR88</td>
<td>$0.33</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Microsoft</td>
<td>02MAY88</td>
<td>$0.35</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Microsoft</td>
<td>01JUN88</td>
<td>$0.40</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

```sas
/* 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

- **VALUEALIGN=** 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 Statement” 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 377. 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 17.1 on page 378) is used to generate the graph. Here is the output for this example.
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.

```sas
/* 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.;
```

Figure 17.4  Displaying Axis-Aligned Filled Segments Using a Block Plot
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 17.5  Axis-Aligned Multi-row Table Created Using a Block Plot**

To create this graph, some data set up is necessary. First, we use PROC SUMMARY to summarize the data in Sashelp.Cars and write it to data set Stats.

**Example Code 17.2  SAS Code That Generates the Stats Data Set**

```sas
/* 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 17.4  Listing of Stats Data Set

<table>
<thead>
<tr>
<th>Obs</th>
<th>Type</th>
<th>N</th>
<th>Mean</th>
<th>UCLM</th>
<th>LCLM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hybrid</td>
<td>3</td>
<td>56.0000</td>
<td>77.5133</td>
<td>34.4867</td>
</tr>
<tr>
<td>2</td>
<td>SUV</td>
<td>60</td>
<td>20.5000</td>
<td>21.3620</td>
<td>19.6380</td>
</tr>
<tr>
<td>3</td>
<td>Sedan</td>
<td>262</td>
<td>28.6298</td>
<td>29.1732</td>
<td>28.0863</td>
</tr>
<tr>
<td>4</td>
<td>Sports</td>
<td>49</td>
<td>25.4898</td>
<td>26.3234</td>
<td>24.6562</td>
</tr>
<tr>
<td>5</td>
<td>Truck</td>
<td>24</td>
<td>21.0000</td>
<td>22.6378</td>
<td>19.3622</td>
</tr>
<tr>
<td>6</td>
<td>Wagon</td>
<td>30</td>
<td>27.9000</td>
<td>29.5478</td>
<td>26.2522</td>
</tr>
</tbody>
</table>

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.

```sas
/* 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 17.5  Partial Listing of the Blockstats Data Set

<table>
<thead>
<tr>
<th>Obs</th>
<th>type2</th>
<th>statname</th>
<th>stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hybrid</td>
<td>N</td>
<td>3.000</td>
</tr>
<tr>
<td>2</td>
<td>Hybrid</td>
<td>Mean</td>
<td>56.000</td>
</tr>
<tr>
<td>3</td>
<td>Hybrid</td>
<td>UCLM</td>
<td>77.513</td>
</tr>
<tr>
<td>4</td>
<td>Hybrid</td>
<td>LCLM</td>
<td>34.487</td>
</tr>
<tr>
<td>5</td>
<td>SUV</td>
<td>N</td>
<td>60.000</td>
</tr>
<tr>
<td>6</td>
<td>SUV</td>
<td>Mean</td>
<td>20.500</td>
</tr>
<tr>
<td>7</td>
<td>SUV</td>
<td>UCLM</td>
<td>21.362</td>
</tr>
<tr>
<td>8</td>
<td>SUV</td>
<td>LCLM</td>
<td>19.638</td>
</tr>
<tr>
<td>9</td>
<td>Sedan</td>
<td>N</td>
<td>262.000</td>
</tr>
<tr>
<td>10</td>
<td>Sedan</td>
<td>Mean</td>
<td>28.630</td>
</tr>
<tr>
<td>11</td>
<td>Sedan</td>
<td>UCLM</td>
<td>29.173</td>
</tr>
<tr>
<td>12</td>
<td>Sedan</td>
<td>LCLM</td>
<td>28.086</td>
</tr>
</tbody>
</table>

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.

```sas
/* Combine summary data for BARCHARTPARM with tabular data for BLOCKPLOT */
data all;
  merge stats blockstats;
run;
```
Output 17.6  Partial Listing of the ALL Data Set

<table>
<thead>
<tr>
<th>Obs</th>
<th>Type</th>
<th>N</th>
<th>Mean</th>
<th>UCLM</th>
<th>LCLM</th>
<th>type2</th>
<th>statname</th>
<th>stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hybrid</td>
<td>3</td>
<td>56.0000</td>
<td>77.5133</td>
<td>34.4867</td>
<td>Hybrid</td>
<td>N</td>
<td>3.000</td>
</tr>
<tr>
<td>2</td>
<td>SUV</td>
<td>60</td>
<td>20.5000</td>
<td>21.3620</td>
<td>19.6380</td>
<td>Hybrid</td>
<td>Mean</td>
<td>56.000</td>
</tr>
<tr>
<td>3</td>
<td>Sedan</td>
<td>262</td>
<td>28.6298</td>
<td>29.1732</td>
<td>28.0863</td>
<td>Hybrid</td>
<td>UCLM</td>
<td>77.513</td>
</tr>
<tr>
<td>4</td>
<td>Sports</td>
<td>49</td>
<td>25.4898</td>
<td>26.3234</td>
<td>24.6562</td>
<td>Hybrid</td>
<td>LCLM</td>
<td>34.487</td>
</tr>
<tr>
<td>5</td>
<td>Truck</td>
<td>24</td>
<td>21.0000</td>
<td>22.6378</td>
<td>19.3622</td>
<td>SUV</td>
<td>N</td>
<td>60.000</td>
</tr>
<tr>
<td>6</td>
<td>Wagon</td>
<td>30</td>
<td>27.9000</td>
<td>29.5478</td>
<td>26.2522</td>
<td>SUV</td>
<td>Mean</td>
<td>20.500</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SUV</td>
<td>UCLM</td>
<td>21.362</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SUV</td>
<td>LCLM</td>
<td>19.638</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sedan</td>
<td>N</td>
<td>262.000</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sedan</td>
<td>Mean</td>
<td>28.630</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sedan</td>
<td>UCLM</td>
<td>29.173</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sedan</td>
<td>LCLM</td>
<td>28.086</td>
</tr>
</tbody>
</table>

...
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 17.3 on page 378. In this example, an axis table is used instead of a block plot. The same data (Example Code 17.1 on page 378) is used to generate the plot.

Here is the output for this example.

Figure 17.6  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;```
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 Statement” 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 17.5 on page 382. The example uses an axis table instead of a block plot. The same data (Example Code 17.2 on page 382) is used to generate the plot. Here is the output for this example.

Figure 17.7  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 17.2 on page 382.

Here is the code that defines the template for this example and then renders the graph.

```sas
/* Create the template for the graph. */
proc template;
   define statgraph axistable2;
end;
run;

/* Render the graph. */
proc sgrender data=events template=axistable1;
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-AlignedInset in a Classification Panel**

Starting with the first maintenance release of SAS 9.4, 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 17.2 on page 373. 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.

```sas
/* 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
```
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 INNERMARGIN 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 421. For information about the GTL annotation facility, see “About the GTL Annotation Facility” in *SAS Graph Template Language: Reference*. 
Part 5

Custom Graphical Elements

Chapter 18
   Adding Code-Driven Graphics Elements to Your Graph ............ 393

Chapter 19
   Adding Data-Driven Annotations to Your Graph ................. 405
Chapter 18
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 394. 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 396.

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.

<table>
<thead>
<tr>
<th>Drawing Space</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| 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. | ![Data Area](image) |

| 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. | ![Wall Area](image) |
### Drawing Space and Description

<table>
<thead>
<tr>
<th>Drawing Space</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layout</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Graph</td>
<td>The area in which the entire graph is displayed.</td>
<td></td>
</tr>
</tbody>
</table>

**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.

```
TOPLEFT  TOP  TOPRIGHT
      ●        ●        ●
LEFT    CENTER  RIGHT
      ●        ●        ●
BOTTOMLEFT  BOTTOM  BOTTOMRIGHT
```

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.

<table>
<thead>
<tr>
<th>To Draw this Type of Graphics Element</th>
<th>Use this GTL Statement or Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>DRAWTEXT</td>
</tr>
<tr>
<td>An arrow</td>
<td>DRAWARROW</td>
</tr>
<tr>
<td>A line</td>
<td>DRAWLINE</td>
</tr>
</tbody>
</table>
To Draw this Type of Graphics Element  Use this GTL Statement or Block

<table>
<thead>
<tr>
<th>Graphics Element</th>
<th>Statement or Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>An oval or circle</td>
<td>DRAWOVAL</td>
</tr>
<tr>
<td>A square or rectangle</td>
<td>DRAWRECTANGLE</td>
</tr>
<tr>
<td>A polyline</td>
<td>DRAW statements within a</td>
</tr>
<tr>
<td></td>
<td>BEGINPOLYLNE block</td>
</tr>
<tr>
<td>A polygon</td>
<td>DRAW statements within a</td>
</tr>
<tr>
<td></td>
<td>BEGINPOLYGON block</td>
</tr>
<tr>
<td>An image</td>
<td>DRAWIMAGE</td>
</tr>
</tbody>
</table>

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 DRAWTTEXT statement to add a text element to your graph. The basic syntax is as follows:

```plaintext
DRAWTTEXT <TEXTATTRS=(text-options)> "text" / X=x Y=y <options>
```

In your DRAWTTEXT 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 DRAWTTEXT statement.

*Note:* The TEXTATTRS= option must be placed before "text" in the DRAWTTEXT statement.

Use the X= and Y= options in the DRAWTTEXT 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 DRAWTTEXT statement that adds a 120-pixel-wide block of text with a gray border.

```plaintext
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:

```plaintext
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.

```plaintext
drawarrow x1=60 y1=70 x2=190 y2=70 / drawspace=graphpixel
   lineattrs=(pattern=3 thickness=1px)
   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.
Adding Graphics Elements to Your Graph

The LINEATTRS= option specifies a dashed line (PATTERN=3) that is one pixel 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=blue pattern=shortdash thickness=1px);
```
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 blue dashed outline.

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=graphtable
   widthunit=percent heightunit=percent
   anchor=bottomleft
display=all
   fillattrs=(color=lightgray
   outlineattrs=(color=blue pattern=shortdash thickness=1px);
```
The \texttt{ANCHOR=} option specifies the rectangle anchor point as \texttt{BOTTOMLEFT}, which positions the lower left corner at coordinates \(80, 80\). The \texttt{DISPLAY=} \texttt{ALL} option displays the outline and fill. The \texttt{FILLATTRS=} option specifies a light gray fill, and the \texttt{OUTLINEATTRS=} option specifies a blue dashed outline.

### Polylines

Use a \texttt{BEGINPOLYLINE} block to add a polyline to your graph. The basic syntax is as follows:

```plaintext
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 \texttt{BEGINPOLYLINE} statement to open the block. In the \texttt{BEGINPOLYLINE} statement, use the \texttt{X=} and \texttt{Y=} options to specify the coordinates of the beginning point of the polyline. By default, the coordinate units are \texttt{GRAPHPERCENT}. You can include the \texttt{DRAWSPACE=} option, or the \texttt{XSPACE=} and \texttt{YSPACE=} options in \texttt{options} to specify different units. If you specify the units as \texttt{DATAVALUE} and you want to scale the polygon to the secondary axis, you must also include the \texttt{XAXIS=X2} and \texttt{YAXIS=Y2} options. To change the line color, pattern, or thickness, include the \texttt{LINEATTRS=} option.

Following the \texttt{BEGINPOLYLINE} statement are the individual \texttt{DRAW} statements. Each \texttt{DRAW} statement draws a straight line from the previous point to the endpoint that is specified in the \texttt{DRAW} statement's \texttt{X=} and \texttt{Y=} options. For the first \texttt{DRAW} statement, the previous point is the starting point that is specified in the \texttt{BEGINPOLYLINE} statement. For subsequent \texttt{DRAW} statements, the previous point is the endpoint that is specified in the previous \texttt{DRAW} statement. Add a \texttt{DRAW} statement for each segment in your polyline. You can add as many segments as you need. After the \texttt{DRAW} statements, add an \texttt{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 \texttt{GRAPHPIXEL} units.

```plaintext
beginpolyline x=30 y=150 / xspace=graphpixel yspace=graphpixel
  lineattrs=(thickness=1px);
  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 */
ENDPOLYLINE;
```
The following figure shows how this polyline is drawn.

![Polyline Diagram](image)

The X= and Y= options in the BEGINPOLYLINE statement specify the starting point of the polyline, (30, 150), in GRAPHPixel units. 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 DATAVALE 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)
outlinenattrs=(thickness=1px pattern=shortdash color=blue);
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 */
derpolygon;

The following figure shows how the polygon is drawn.

![Polygon Diagram]

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 dashed blue outline. 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.

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=y <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.
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/GRAF 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 394. 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 the 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.
• 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, you must, at a minimum, specify TEXT in the Function column and the annotation text in the Label column. You can also use the TEXTCONT function to continue the text for a text annotation by using different text attributes. You can use the ODS escape sequence to include superscripts, subscripts, and Unicode characters in the label as you would in quoted strings for a text command. For more information, see “Using Text Commands” in SAS Graph Template Language: Reference.

By default, the text 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. The width of the text box is determined by the system. You can add the Width column and specify a new width. You can add other columns to specify other text attributes such as font, size, color, and so on. For more information, see “TEXT Function” in SAS ODS Graphics: Procedures Guide and “TEXTCONT Function” in SAS ODS Graphics: Procedures Guide.
Here is an example of a DATA step that creates three observations for a text annotation that contains normal text, blue text, and red text that expresses 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";
output; /* Write normal-text observation. */
function="textcont";
textcolor="blue";
label=" rich text with Unicode and superscript characters:";
output; /* Write blue-text observation. */
textcolor="red";
label=" A = (*ESC*){unicode pi}R(*ESC*){sup 2}";
output; /* Write red equation observation. */
run;

Note: Starting with the first maintenance release of SAS 9.4, 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 416.

Here is a listing of TEXTDATA.

<table>
<thead>
<tr>
<th>Obs</th>
<th>function</th>
<th>label</th>
<th>drawspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>text</td>
<td>A text string that contains normal text and</td>
<td>graphpixel</td>
</tr>
<tr>
<td>2</td>
<td>textcont</td>
<td>rich text with Unicode and superscript characters:</td>
<td>graphpixel</td>
</tr>
<tr>
<td>3</td>
<td>textcont</td>
<td>A = (<em>ESC</em>){unicode pi}R(<em>ESC</em>){sup 2}</td>
<td>graphpixel</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>x1</th>
<th>y1</th>
<th>width</th>
<th>widthunit</th>
<th>anchor</th>
<th>border</th>
<th>linecolor</th>
<th>linepattern</th>
<th>textcolor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>70</td>
<td>190</td>
<td>pixel</td>
<td>left</td>
<td>true</td>
<td>gray</td>
<td>solid</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>70</td>
<td>190</td>
<td>pixel</td>
<td>left</td>
<td>true</td>
<td>gray</td>
<td>solid</td>
<td>blue</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>70</td>
<td>190</td>
<td>pixel</td>
<td>left</td>
<td>true</td>
<td>gray</td>
<td>solid</td>
<td>blue</td>
</tr>
</tbody>
</table>

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.

![Diagram of an arrow annotation](image)

**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 an observation for an arrow annotation.

```sas
data arrowdata;
  function="arrow";
  drawspace="graphpixel";
  x1=60; y1=70; x2=190; y2=70;
  shape="barbed"; scale=2; direction="both";
  linepattern="shortdash"; linethickness=1;
run;
```

**Note:** Starting with the first maintenance release of SAS 9.4, 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 416.

Here is a listing of the observation.

<table>
<thead>
<tr>
<th>Obs</th>
<th>function</th>
<th>drawspace</th>
<th>x1</th>
<th>y1</th>
<th>x2</th>
<th>y2</th>
<th>shape</th>
<th>scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>arrow</td>
<td>graphpixel</td>
<td>60</td>
<td>70</td>
<td>190</td>
<td>70</td>
<td>barbed</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>direction</th>
<th>linepattern</th>
<th>linethickness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>both</td>
<td>shortdash</td>
<td>1</td>
</tr>
</tbody>
</table>

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 and LineThickness columns specify the arrow line as a 2-pixel-wide short-dashed line.
Here is how this annotation is placed in a graph.

```
Arrowhead Direction is BOTH

(IN) ←¬¬¬¬→ (OUT)

(X1=60, Y1=70) (X2=190, Y2=70)
```

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.

```sas
data linedata;
  function="line";
  drawspace="graphpixel";
  x1=60; y1=70; x2=190; y2=70;
  linepattern="shortdash"; linethickness=1;
run;
```

Here is a listing of the observation.

```
Obs function drawspace  x1  y1  x2  y2  linepattern linethickness
1   line   graphpixel 60  70 190  70  shortdash        1
```

**Adding Observations for Polylines and Polygons**

Multiple observations are required to construct a polyline or polygon. The first observation establishes the starting point of the polyline or polygon. The subsequent observations specify the individual segments of the polyline or polygon. A polyline must have at least one segment. 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 an observation for a five-segment polyline annotation.

```sas
data polylinedata;
  function="polyline";
  drawspace="graphpixel";
  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 the first maintenance release of SAS 9.4, 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 416.

Here is a listing of the observation.

<table>
<thead>
<tr>
<th>Obs</th>
<th>function</th>
<th>drawspace</th>
<th>x1</th>
<th>y1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>polyline</td>
<td>graphpixel</td>
<td>30</td>
<td>150</td>
</tr>
<tr>
<td>2</td>
<td>polycont</td>
<td>graphpixel</td>
<td>110</td>
<td>150</td>
</tr>
<tr>
<td>3</td>
<td>polycont</td>
<td>graphpixel</td>
<td>145</td>
<td>250</td>
</tr>
<tr>
<td>4</td>
<td>polycont</td>
<td>graphpixel</td>
<td>170</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>polycont</td>
<td>graphpixel</td>
<td>205</td>
<td>150</td>
</tr>
<tr>
<td>6</td>
<td>polycont</td>
<td>graphpixel</td>
<td>285</td>
<td>150</td>
</tr>
</tbody>
</table>

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 an observation for a four-segment polyline annotation.

```plaintext
data polygondata;
  length function $8;
  function="polygon";
  drawspace="graphpixel";
  x1=40; y1=100; /* Begin polygon */
  display="all"; fillcolor="lightgray";
  linecolor="blue"; linepattern="shortdash";
  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 observation.

<table>
<thead>
<tr>
<th>Obs</th>
<th>function</th>
<th>drawspace</th>
<th>x1</th>
<th>y1</th>
<th>display</th>
<th>fillcolor</th>
<th>linecolor</th>
<th>linepattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>polygon</td>
<td>graphpixel</td>
<td>40</td>
<td>100</td>
<td>all</td>
<td>lightgray</td>
<td>blue</td>
<td>shortdash</td>
</tr>
<tr>
<td>2</td>
<td>polycont</td>
<td>graphpixel</td>
<td>30</td>
<td>220</td>
<td>all</td>
<td>lightgray</td>
<td>blue</td>
<td>shortdash</td>
</tr>
<tr>
<td>3</td>
<td>polycont</td>
<td>graphpixel</td>
<td>160</td>
<td>200</td>
<td>all</td>
<td>lightgray</td>
<td>blue</td>
<td>shortdash</td>
</tr>
<tr>
<td>4</td>
<td>polycont</td>
<td>graphpixel</td>
<td>180</td>
<td>80</td>
<td>all</td>
<td>lightgray</td>
<td>blue</td>
<td>shortdash</td>
</tr>
<tr>
<td>5</td>
<td>polycont</td>
<td>graphpixel</td>
<td>40</td>
<td>100</td>
<td>all</td>
<td>lightgray</td>
<td>blue</td>
<td>shortdash</td>
</tr>
</tbody>
</table>

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). 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 an observation for a 90-pixel by 110-pixel oval annotation.

```sas
data ovaldata;
  function='oval';
  drawspace='graphpixel';
  x1=115; y1=110;
  width=90; widthunit='pixel'; height=110; heightunit='pixel';
  display='all'; fillcolor='lightgray';
```

...
Note: Starting with the first maintenance release of SAS 9.4, 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 416.

Here is a listing of the observation.

<table>
<thead>
<tr>
<th>Obs</th>
<th>function</th>
<th>drawspace</th>
<th>x1</th>
<th>y1</th>
<th>width</th>
<th>widthunit</th>
<th>height</th>
<th>heightunit</th>
<th>display</th>
<th>fillcolor</th>
<th>linecolor</th>
<th>linepattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>oval</td>
<td>graphpixel</td>
<td>115</td>
<td>110</td>
<td>90</td>
<td>pixel</td>
<td>110</td>
<td>pixel</td>
<td>all</td>
<td>lightgray</td>
<td>blue</td>
<td>shortdash</td>
</tr>
</tbody>
</table>

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 an observation for 60%-by-40% rectangle annotation.

```plaintext
data rectangledata;
```
function="rectangle";
drawspace="graphpixel";
x1=80; y1=80; width=60; height=40; anchor="bottomleft";
fillcolor="lightgray"; display="all";
linecolor="blue"; linepattern="shortdash";
run;

**Note:** Starting with the first maintenance release of SAS 9.4, 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 416.

Here is a listing of the observation.

<table>
<thead>
<tr>
<th>Obs</th>
<th>function</th>
<th>drawspace</th>
<th>x1</th>
<th>y1</th>
<th>width</th>
<th>height</th>
<th>anchor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>rectangle</td>
<td>graphpixel</td>
<td>80</td>
<td>80</td>
<td>60</td>
<td>40</td>
<td>bottomleft</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>fillcolor</th>
<th>display</th>
<th>linecolor</th>
<th>linepattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>lightgray</td>
<td>all</td>
<td>blue</td>
<td>shortdash</td>
</tr>
</tbody>
</table>

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 LinePattern columns specify a blue short-dashed outline.

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 an observation for a SAS logo.

```sas
data imagedata;
  function="image";
  drawspace="graphpixel";
  x1=70; y1=60;
  anchor="bottomleft";
  border="true";
  image="."\saslogo.png";
run;
```

**Note:** Starting with the first maintenance release of SAS 9.4, 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 416.

Here is a listing of the observation.

<table>
<thead>
<tr>
<th>Obs</th>
<th>function</th>
<th>drawspace</th>
<th>x1</th>
<th>y1</th>
<th>anchor</th>
<th>border</th>
<th>image</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>image</td>
<td>graphpixel</td>
<td>70</td>
<td>60</td>
<td>bottomleft</td>
<td>true</td>
<td>.\saslogo.png</td>
</tr>
</tbody>
</table>

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

Starting with the first maintenance release of SAS 9.4, macros are available that you can run in a DATA step to simplify the process of creating your SG annotation data sets. The following table lists the available macros.

Table 19.1 SG Annotation Macros

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| %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. |
| %SGTEXTCONT  | Creates an observation that draws a continuing line of text in a multiline annotation. |

For more information about these macros, see *SAS ODS Graphics: Procedures Guide*. 

416 Chapter 19 • Adding Data-Driven Annotations to Your Graph
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 417 and “Example 2: Displaying Subsets of Annotations in Axis-Aligned Insets” on page 421.

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 Statement” in SAS Graph Template Language: Reference. For an example, see “Example 2: Displaying Subsets of Annotations in Axis-Aligned Insets” on page 421.

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.

![Average Highway Mileage by Vehicle Type](image)

**Program**

```sas
/* 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);
    endgraph;
end;
```

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 421.

/* Summarize the highway mileage data in SASHHELP.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 421.

/* 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 the first maintenance release of SAS 9.4, 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;
%sgtest(label=type,
x1=2,y1=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 421.
Listing of the Mileage Data Set

Here is a listing of the Mileage data set.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Type</th>
<th><em>TYPE</em></th>
<th><em>FREQ</em></th>
<th>mpg_city</th>
<th>highway</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hybrid</td>
<td>1</td>
<td>3</td>
<td>55.0000</td>
<td>56.0000</td>
</tr>
<tr>
<td>2</td>
<td>SUV</td>
<td>1</td>
<td>60</td>
<td>16.1000</td>
<td>20.5000</td>
</tr>
<tr>
<td>3</td>
<td>Sedan</td>
<td>1</td>
<td>262</td>
<td>21.0840</td>
<td>28.6298</td>
</tr>
<tr>
<td>4</td>
<td>Sports</td>
<td>1</td>
<td>49</td>
<td>18.4082</td>
<td>25.4898</td>
</tr>
<tr>
<td>5</td>
<td>Truck</td>
<td>1</td>
<td>24</td>
<td>16.5000</td>
<td>21.0000</td>
</tr>
<tr>
<td>6</td>
<td>Wagon</td>
<td>1</td>
<td>30</td>
<td>21.1000</td>
<td>27.9000</td>
</tr>
</tbody>
</table>

Listing of the Anno Data Set

Here is a listing of the Anno data set.

<table>
<thead>
<tr>
<th>Obs</th>
<th>function</th>
<th>drawspace</th>
<th>textfont</th>
<th>textweight</th>
<th>textcolor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>text</td>
<td>datavalue</td>
<td>Arial</td>
<td>bold</td>
<td>GraphData1:contrastColor</td>
</tr>
<tr>
<td>2</td>
<td>text</td>
<td>datavalue</td>
<td>Arial</td>
<td>bold</td>
<td>GraphData1:contrastColor</td>
</tr>
<tr>
<td>3</td>
<td>text</td>
<td>datavalue</td>
<td>Arial</td>
<td>bold</td>
<td>GraphData1:contrastColor</td>
</tr>
<tr>
<td>4</td>
<td>text</td>
<td>datavalue</td>
<td>Arial</td>
<td>bold</td>
<td>GraphData1:contrastColor</td>
</tr>
<tr>
<td>5</td>
<td>text</td>
<td>datavalue</td>
<td>Arial</td>
<td>bold</td>
<td>GraphData1:contrastColor</td>
</tr>
<tr>
<td>6</td>
<td>text</td>
<td>datavalue</td>
<td>Arial</td>
<td>bold</td>
<td>GraphData1:contrastColor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>discreteoffset</th>
<th>width</th>
<th>widthunit</th>
<th>anchor</th>
<th>x1</th>
<th>y1</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>100</td>
<td>pixel</td>
<td>left</td>
<td>2</td>
<td>Hybrid</td>
<td>Hybrid</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>100</td>
<td>pixel</td>
<td>left</td>
<td>2</td>
<td>SUV</td>
<td>SUV</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
<td>100</td>
<td>pixel</td>
<td>left</td>
<td>2</td>
<td>Sedan</td>
<td>Sedan</td>
</tr>
<tr>
<td>4</td>
<td>0.1</td>
<td>100</td>
<td>pixel</td>
<td>left</td>
<td>2</td>
<td>Sports</td>
<td>Sports</td>
</tr>
<tr>
<td>5</td>
<td>0.1</td>
<td>100</td>
<td>pixel</td>
<td>left</td>
<td>2</td>
<td>Truck</td>
<td>Truck</td>
</tr>
<tr>
<td>6</td>
<td>0.1</td>
<td>100</td>
<td>pixel</td>
<td>left</td>
<td>2</td>
<td>Wagon</td>
<td>Wagon</td>
</tr>
</tbody>
</table>

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 the first maintenance release of SAS 9.4 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. */
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 426.

```sas
/* 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.

```sas
/* 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.

```sas
/* Compile the annotation macros */
%sганно;
```

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.

```sas
/* 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),
                x1=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.

```sas
/* 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",
```
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.

```sas
/* 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.

```sas
/* 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.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Origin</th>
<th>Type</th>
<th>mpg_city</th>
<th>highway</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Asia</td>
<td>Hybrid</td>
<td>55.0000</td>
<td>56.0000</td>
</tr>
<tr>
<td>2</td>
<td>Asia</td>
<td>SUV</td>
<td>17.3200</td>
<td>21.6800</td>
</tr>
<tr>
<td>3</td>
<td>Asia</td>
<td>Sedan</td>
<td>22.8404</td>
<td>29.9681</td>
</tr>
<tr>
<td>4</td>
<td>Asia</td>
<td>Sports</td>
<td>20.2353</td>
<td>26.6471</td>
</tr>
<tr>
<td>5</td>
<td>Asia</td>
<td>Truck</td>
<td>17.8750</td>
<td>22.0000</td>
</tr>
<tr>
<td>6</td>
<td>Asia</td>
<td>Wagon</td>
<td>22.3636</td>
<td>28.1818</td>
</tr>
<tr>
<td>7</td>
<td>Europe</td>
<td>SUV</td>
<td>14.5000</td>
<td>18.7000</td>
</tr>
<tr>
<td>8</td>
<td>Europe</td>
<td>Sedan</td>
<td>19.5128</td>
<td>27.1154</td>
</tr>
<tr>
<td>9</td>
<td>Europe</td>
<td>Sports</td>
<td>17.6522</td>
<td>25.1304</td>
</tr>
<tr>
<td>10</td>
<td>Europe</td>
<td>Wagon</td>
<td>19.2500</td>
<td>26.5833</td>
</tr>
<tr>
<td>11</td>
<td>USA</td>
<td>SUV</td>
<td>15.5200</td>
<td>20.0400</td>
</tr>
<tr>
<td>12</td>
<td>USA</td>
<td>Sedan</td>
<td>20.6111</td>
<td>28.5444</td>
</tr>
<tr>
<td>13</td>
<td>USA</td>
<td>Sports</td>
<td>16.8889</td>
<td>24.2222</td>
</tr>
<tr>
<td>14</td>
<td>USA</td>
<td>Truck</td>
<td>15.8125</td>
<td>20.5000</td>
</tr>
<tr>
<td>15</td>
<td>USA</td>
<td>Wagon</td>
<td>22.2857</td>
<td>29.7143</td>
</tr>
</tbody>
</table>

Partial Listing of the Anno Data Set

Here are the last seven observations from the Anno data set.

<table>
<thead>
<tr>
<th>Obs</th>
<th>ID</th>
<th>FUNCTION</th>
<th>LABEL</th>
<th>BORDER</th>
<th>ANCHOR</th>
<th>JUSTIFY</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>USA</td>
<td>TEXT</td>
<td>16 / 20</td>
<td>true</td>
<td>center</td>
<td>center</td>
</tr>
<tr>
<td>12</td>
<td>USA</td>
<td>TEXT</td>
<td>21 / 29</td>
<td>true</td>
<td>center</td>
<td>center</td>
</tr>
<tr>
<td>13</td>
<td>USA</td>
<td>TEXT</td>
<td>17 / 24</td>
<td>true</td>
<td>center</td>
<td>center</td>
</tr>
<tr>
<td>14</td>
<td>USA</td>
<td>TEXT</td>
<td>16 / 21</td>
<td>true</td>
<td>center</td>
<td>center</td>
</tr>
<tr>
<td>15</td>
<td>USA</td>
<td>TEXT</td>
<td>22 / 30</td>
<td>true</td>
<td>center</td>
<td>center</td>
</tr>
<tr>
<td>16</td>
<td>INSETLABEL</td>
<td>TEXT</td>
<td>Avg. MPG</td>
<td>false</td>
<td>right</td>
<td>left</td>
</tr>
<tr>
<td>17</td>
<td>INSETLABEL</td>
<td>TEXT</td>
<td>(City/Hwy)</td>
<td>false</td>
<td>right</td>
<td>left</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>WIDTH</th>
<th>X1</th>
<th>XCl</th>
<th>X1SPACE</th>
<th>Y1</th>
<th>Y1SPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>90</td>
<td>.</td>
<td>SUV</td>
<td>datavalue</td>
<td>80</td>
<td>graphpercent</td>
</tr>
<tr>
<td>12</td>
<td>90</td>
<td>.</td>
<td>Sedan</td>
<td>datavalue</td>
<td>80</td>
<td>graphpercent</td>
</tr>
<tr>
<td>13</td>
<td>90</td>
<td>.</td>
<td>Sports</td>
<td>datavalue</td>
<td>80</td>
<td>graphpercent</td>
</tr>
<tr>
<td>14</td>
<td>90</td>
<td>.</td>
<td>Truck</td>
<td>datavalue</td>
<td>80</td>
<td>graphpercent</td>
</tr>
<tr>
<td>15</td>
<td>90</td>
<td>.</td>
<td>Wagon</td>
<td>datavalue</td>
<td>80</td>
<td>graphpercent</td>
</tr>
<tr>
<td>16</td>
<td>90</td>
<td>16</td>
<td></td>
<td>graphpercent</td>
<td>81</td>
<td>graphpercent</td>
</tr>
<tr>
<td>17</td>
<td>90</td>
<td>16</td>
<td></td>
<td>graphpercent</td>
<td>77</td>
<td>graphpercent</td>
</tr>
</tbody>
</table>

Note: The actual column order is different from that presented here.
Part 6

Interactive Graphs

Chapter 20
Adding Data Tips to Your Graph ................................. 429

Chapter 21
Adding Drill-Down Links to Your Graph ..................... 433

Chapter 22
Creating Animated Graphs ........................................ 439
Chapter 20
Adding Data Tips to Your Graph

Creating a Graph with Data Tips in an HTML Page

Creating a Graph with Custom Data Tips in an HTML Page

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 "mouse over" parts of a graph, and text balloons open to show information (typically data values) that is associated with the area where the mouse pointer rests. Nearly all plot statements in GTL create default data tip information. However, this information is not generated unless you request it with the IMAGEMAP= option in the ODS GRAPHICS statement 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.

![Scatter Plot with Default Data Tips](image)

ODS Graphics disables data tips when a preset upper threshold is reached. Prior to the third maintenance release of SAS 9.4, 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 the third maintenance release of SAS 9.4, 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 Statement” in SAS ODS Graphics: Procedures Guide.

---

**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.

```sas
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")
```
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, we want the Name and Age column values to show in the tip. Notice that the choice of role names is somewhat arbitrary. The TIP1 and TIP2 role names are added to the default role names X, Y, and GROUP.

The TIP= option defines a list of roles to be displayed, and it also determines their order 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 following TIPLABEL= option:

```
tiplabel=(tip1="Student Name" group="Group")
```

For any role, you can assign a format to the display of tip values.
Chapter 20 • Adding Data Tips 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
- BARCHARTPARM
- BUBBLEPLOT
- HEATMAPPARM
- LINECHART
- MOSAICPLOTPARM
- NEEDLEPLOT
- PIECHART
- POLYGONPLOT
- SERIESPLOT
- STEPPLOT
- WATERFALLCHART
- SCATTERPLOT

Starting with the first maintenance release of SAS 9.4, the following support is available:

- The following draw statements support the URL= option:
  - BEGINPOLYGON
  - BEGINPOLYLINE
  - DRAWTXT
  - DRAWLINE
  - DRAWARROW
  - DRAWIMAGE
  - DRAWOVAL
  - 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 21.1 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. The drill-down link target is a new browser window by default. You can use the DRILLTARGET= option in your ODS GRAPHICS statement to specify a different target, such as _blank or _self.

_Note:_ You must specify _blank or _self in lowercase.

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 destination.

For example, to create the drill-down pie chart in Figure 21.1 on page 434, 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=ur-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;

6. Generate the pie chart by using the ODS HTML destination.
   
   Note: In order to support the drill-down links, you must use 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.

```sas
/* 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";
run;
```
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
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.
Chapter 22
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 22.1 System Options That Control Animated Graphs

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Valid Printers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANIMATION=START</td>
<td>GIF and SVG</td>
<td>Starts or stops the creation of an animation file.</td>
</tr>
<tr>
<td>ANIMDURATION=MIN</td>
<td>GIF and SVG</td>
<td>Specifies the length of time, in seconds, that each frame in an animation is held in view. The default is MIN (0.1 seconds).</td>
</tr>
<tr>
<td>ANIMLOOP=YES</td>
<td>GIF and SVG</td>
<td>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.</td>
</tr>
<tr>
<td>Option Name</td>
<td>Valid Printers</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ANIMOVERLAY</td>
<td>NOANIMOVERLAY</td>
<td>GIF and SVG</td>
</tr>
<tr>
<td>SVGAUTOPLAY</td>
<td>NOSVGAUTOPLAY</td>
<td>SVG</td>
</tr>
<tr>
<td>SVGFADEIN=number</td>
<td>SVG</td>
<td>Specifies the number of seconds for an SVG frame to fade into view. The default is 0.</td>
</tr>
<tr>
<td>SVGFADEMODE=OVERLAP</td>
<td>SEQUENTIAL</td>
<td>SVG</td>
</tr>
<tr>
<td>SVGFADEOUT=number</td>
<td>SVG</td>
<td>Specifies the number of seconds that it takes for an SVG frame to fade out of view. The default is 0.</td>
</tr>
</tbody>
</table>

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 22.1 on page 439.
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 “About Animated GIF Images and SVG Documents” in SAS Language Reference: Concepts.

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 562.

Here is the SAS code for this example.

```sas
/* 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_;
begingroup;
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
endpoints=missing));
```

```sas
/* 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_;
begingroup;
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
endpoints=missing));
```

```sas
```
```sas
/* 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_;
begingroup;
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
endpoints=missing));
```
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 7

Graphical Output

Chapter 23
Managing Your Graph’s Appearance ......................... 445

Chapter 24
Managing Your Graphics Output .......................... 519
Chapter 23
Managing Your Graph’s Appearance

Default Appearance Features in Graphs ........................................... 446

Methods for Changing the Appearance of Your Plots .................... 448

Using ODS Styles to Control Graph Appearance .......................... 448
   Evaluating Supplied Styles ........................................... 448
   Changing Attributes in a Style Template .............................. 451
   Controlling ODS Search Paths ...................................... 455

Using Options to Override Style Attributes ................................. 457
   Options That Override Attributes for Individual Plots .............. 457
   Options That Override Style Attributes for All of the Plots in a Template .............................. 460

Color-Naming Schemes ............................................................. 461
   Overview of Color-Naming Schemes .................................. 461
   RGB Color Codes ...................................................... 462
   CMYK Color Codes ................................................... 463
   HLS Color Codes ..................................................... 464
   HSV (or HSB) Color Codes ............................................. 464
   Gray-Scale Color Codes ................................................ 465
   Color Naming System (CNS) Values .................................. 466
   SAS Color Names and RGB Values in the SAS Registry .............. 467
   Converting Color Values between Color-Naming Schemes .......... 468
   Converting Numeric Color Component Values to Color Names .... 468
   Converting Color Values from Other Applications .................. 468

Controlling the Appearance of Non-grouped Data ......................... 470

Controlling the Appearance of Grouped Data ............................... 473
   Plots That Support Grouped Data ...................................... 473
   Using the Default Appearance for Grouped Data ...................... 473
   Using Custom Styles to Control the Appearance of Grouped Data .. 475
   Using a Discrete Attribute Map to Control the Appearance of Grouped Data .............................. 477
   Controlling the Appearance of Grouped Data for All Graphs in a Template .. 477
   Changing the Grouped Data Display .................................. 481
   Including Missing Group Values .................................... 485
   Changing the Grouped Data Order .................................... 487
   Making the Appearance of Grouped Data Independent of Data Order .............................. 489

Using Attribute Maps ............................................................... 493
   About Attribute Maps .................................................. 493
   Using a Discrete Attribute Map ....................................... 494
   Using a Range Attribute Map ........................................... 499
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 23.1  ods listing style=default;
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 448.

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.

<table>
<thead>
<tr>
<th>To Change...</th>
<th>Method</th>
<th>For Information, See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the attributes of all plots that use a specific ODS style</td>
<td>Use one of the other supplied ODS styles or create a custom ODS style.</td>
<td>“Using ODS Styles to Control Graph Appearance” on page 448</td>
</tr>
<tr>
<td>the attributes of individual plots to make them independent of the current style</td>
<td>Use the appropriate attribute options in the plot statements.</td>
<td>“Options That Override Attributes for Individual Plots” on page 457</td>
</tr>
<tr>
<td>the attributes of all of the plots in a specific template to make them independent of the current style</td>
<td>Use the appropriate attribute options in the BEGINGRAPH statement for the template.</td>
<td>“Controlling the Appearance of Grouped Data for All Graphs in a Template” on page 477</td>
</tr>
<tr>
<td>the attributes of group values to make them independent of the data order</td>
<td>Use a discrete attribute map or the INDEX= option.</td>
<td>“Making the Appearance of Grouped Data Independent of Data Order” on page 489</td>
</tr>
<tr>
<td>the attributes of numeric values based on numeric ranges</td>
<td>Use a range attribute map.</td>
<td>“Using a Range Attribute Map” on page 499</td>
</tr>
</tbody>
</table>

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:

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

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

<table>
<thead>
<tr>
<th>Obs</th>
<th>Path</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Styles</td>
<td>Dir</td>
</tr>
<tr>
<td>2</td>
<td>Styles.Analysis</td>
<td>Style</td>
</tr>
<tr>
<td>3</td>
<td>Styles.Astronomy</td>
<td>Style</td>
</tr>
<tr>
<td>4</td>
<td>Styles.Banker</td>
<td>Style</td>
</tr>
</tbody>
</table>

(more observations)

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.

**Table 23.1 Recommended ODS Styles**

<table>
<thead>
<tr>
<th>Style</th>
<th>Example</th>
</tr>
</thead>
</table>
| LISTING | • white background  
|         | • white wall       
|         | • sans-serif fonts 
|         | • color used for lines, markers, and filled areas  
|         | • other colors the same as DEFAULT style |

| DEFAULT | • gray background  
|         | • white wall       
<p>|         | • sans-serif fonts |</p>
<table>
<thead>
<tr>
<th>Style</th>
<th>Example</th>
<th>Details</th>
</tr>
</thead>
</table>
| STATISTICAL   | ![Statistical Style](image1.png) | - white background  
- white wall  
- sans-serif fonts  
- contrasting color scheme of blues, reds, greens for markers, lines, and filled areas |
| HTMLBLUE      | ![HTMLBlue Style](image2.png) | - white background  
- white wall  
- sans-serif fonts  
- table colors match the graph colors  
- group distinctions based on color rather than marker or line styles  
- a lighter color scheme for HTML content |
| ANALYSIS      | ![Analysis Style](image3.png) | - light tan background  
- white wall  
- sans-serif fonts  
- muted color scheme of tans, greens, yellows, oranges, and browns for lines, markers, and filled areas |
| JOURNAL and JOURNAL3 | ![Journal3 Style](image4.png) | - white background  
- white wall  
- sans-serif fonts  
- gray-scale color scheme for markers, lines, and filled areas  
- gray-scale pattern and color scheme for bar fill patterns (JOURNAL3 only) |
Changing Attributes in a Style Template

You can create graphs with specific visual characteristics that do not have to be hard coded into the GTL for every graph that you create. For example, you might want to modify settings for the following graph features:

- font or font sizes
- line or marker properties
- colors
- display features for box plots, histograms, 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. 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.

Suppose you want to create a custom style that uses serif fonts. Notice that the
recommended styles listed in Table 23.1 on page 449 use sans-serif fonts. To create the
custom style, select the STATISTICAL style as a starting point (parent) for the custom
style. The following template code is a partial listing of the style template for the
STATISTICAL style, which is delivered with the SAS System. Only the code that needs
to be modified is shown.

```
proc template;
  define style Styles.Statistical;
  parent = styles.default;
  style fonts /
    'TitleFont2'=('<sans-serif>,<MTsans-serif>,Helvetica,Helv",2,bold)
    'TitleFont'=('<sans-serif>,<MTsans-serif>,Helvetica,Helv",3,bold)
    'StrongFont'=('<sans-serif>,<MTsans-serif>,Helvetica,Helv",2,bold)
    'EmphasisFont'=('<sans-serif>,<MTsans-serif>,Helvetica,Helv",2,italic)
    'FixedFont'=('<monospace>,Courier",2)
    'BatchFixedFont'=('SAS Monospace,<monospace>,Courier,monospace",2)
    'FixedHeadingFont'=('<monospace>,Courier,monospace",2)
    'FixedStrongFont'=('<monospace>,Courier,monospace",2,bold)
    'FixedEmphasisFont'=('<monospace>,Courier,monospace",2,italic)
    'headingEmphasisFont'=('<sans-serif>,<MTsans-serif>,Helvetica,Helv",2,bold italic)
    'headingFont'=('<sans-serif>,<MTsans-serif>,Helvetica,Helv",2,bold italic)
    'docFont'=('<sans-serif>,<MTsans-serif>,Helvetica,Helv",2,italic)
  style GraphFonts /
    'GraphDataFont'=('<sans-serif>,<MTsans-serif",7pt)
    'GraphUnicodeFont'=('<MTsans-serif-unicode">,9pt)
```

Make the following changes:

- Assign a name to a new style that identifies STATISTICAL as its parent style. The name of the new style must be different from its parent style so that access to the parent style is not blocked. See discussion under “Controlling ODS Search Paths” on page 455.

- Change the Fonts style element (affects tables) so that it uses a serif font.

- Change the GraphFonts style element (affects graphs) so that it uses a serif font.

Two style elements govern all fonts in a style: the Fonts element governs tables, and the GraphFonts element governs graphs. When you change fonts in a style, be sure to make consistent changes to both elements. In this case, you want to change from a sans-serif font to a serif font. You can also change font size, weight, and style.

In style templates, the name of a font family normally appears as a quoted string. However, ODS also supports an indirect reference to a font family. When a font name appears between less than and greater than symbols, such as <sans-serif>, 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. Here is an example.

```sas
proc registry list startat='ods/fonts';
run;
```

Here is a partial listing of the output in the SAS log.

```
NOTE: Contents of SASHELP REGISTRY starting at subkey [ods/fonts]
[ ods/fonts]
dings="Wingdings"
monospace="Courier New"
MTdings="Monotype Sorts"
MTmonospace="Cumberland AMT"
MTsans-serif="Albany AMT"
MTsans-serif-unicode="Arial Unicode MS"
MTSerif="Thorndale AMT"
MTserif-unicode="Times New Roman Uni"
MTsymbol="Symbol MT"
sans-serif="Arial"
serif="Times New Roman"
symbol="Symbol"
```

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. Fonts are normally listed in a most-
specific to most generic order so that a reasonable substitution can be made when a font cannot be located on the current computer.

Several of the graph fonts affect different features of a graph. The following table shows some features, but not all, that are affected by the graph fonts.

<table>
<thead>
<tr>
<th>Graph Font</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GraphTitleFont</strong></td>
<td>Used for all titles of the graph. Typically, this is the largest font.</td>
</tr>
<tr>
<td><strong>GraphFootnoteFont</strong></td>
<td>Used for all footnotes. Typically, this font is smaller than the title font. Sometimes footnotes are italicized.</td>
</tr>
<tr>
<td><strong>GraphLabelFont</strong></td>
<td>Used for axis labels and legend titles. Generally, this font is smaller than the title font.</td>
</tr>
<tr>
<td><strong>GraphValueFont</strong></td>
<td>Used for axis tick values and legend entries. Generally, this font is smaller than the label font.</td>
</tr>
<tr>
<td><strong>GraphDataFont</strong></td>
<td>Used for text where minimum size is necessary (such as for point labels).</td>
</tr>
<tr>
<td><strong>GraphUnicodeFont</strong></td>
<td>Used for adding special glyphs (for example, $\alpha$, $\pm$, €) to text in the graph (see Chapter 15, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 291).</td>
</tr>
<tr>
<td><strong>GraphAnnoFont</strong></td>
<td>Default font for text added as an annotation in the ODS Graphics Editor.</td>
</tr>
</tbody>
</table>

For a complete list of the style elements that affect ODS Graphics, see Appendix 2, “Graph Style Elements Used by ODS Graphics,” on page 597. For additional information about ODS styles, see *SAS Output Delivery System: User's Guide*.

For this example, name your modified style template *serifStatistical*, change all occurrences of sans-serif to serif, change all occurrences of MTsans-serif to MTserif, and change Helvetica and Helv (sans-serif fonts) to Times (a serif font):

```sas
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 /
    'GraphTitleFont'=('<serif',<MTserif>,11pt,bold)
    'GraphFootnoteFont'=('<serif',<MTserif>,10pt,italic)
    'GraphLabelFont'=('<serif',<MTserif>,10pt)
```

Controlling ODS Search Paths

Before you submit your modified style template, you should consider whether this style is for your use only or whether you want to share it with others. Your decision determines where you store the style.

The ODS PATH statement determines the read and write locations for SAS item store templates. Here is an example.

```sas
ods path show;

Current ODS PATH list is:
1. SASUSER.TEMPLAT(UPDATE)
2. SASHELP.TMPLMST(READ)
```

By default, modified templates are stored in the first path that has Write permission (UPDATE), which is Sasuser.Templat in this example. This path is appropriate for your personal use. To store a modified template in this default location, you will see the following note in the SAS Log after submitting the PROC TEMPLATE code:

```
NOTE: STYLE 'Styles.SerifStatistical' has been saved to:
SASUSER.TEMPLAT
```

You can then run your program with the new style:

```sas
ods rtf style=serifStatistical;
ods graphics on;
proc reg data=sashelp.class;
   model weight=height;
quit;
ods rtf close;
```

To save a modified template to a location where others can access it, you cannot use the default Sasuser.Templat location. Rather, store the template in a different library, using the ODS PATH statement to set the search path:

```sas
libname common "u:\ODS_templates";
ods path common.dept(update)
    sasuser.templat(update)
    sashelp,tmpmst(read);
```

This ODS PATH statement establishes a new search path. The first item store (common.dept) can be updated and will contain the new template (Styles.SerifStatistical). It is important to include Sashelp.Tmplmst in the path because the inherited parent style (Styles.Default) is in Sashelp.

After setting this new search path, you will see the following note in the SAS Log when you submit the PROC TEMPLATE code:
NOTE: STYLE 'Styles.SerifStatistical' has been saved to:
COMMON.DEPT

For others to access this style template, everyone needs to precede their programs with the following code:

```sas
libname common "u:\ODS_templates" access=readonly;
ods path sasuser.templat(update)
  common.dept(read)
  sashelp.tmpmst(read);
```

They can then run their program with the new style:

```sas
ods rtf style=serifStatistical;
ods graphics on;
proc reg data=sashelp.class;
  model weight=height;
quit;
ods rtf close;
```

The following figure shows a table and graph from the output.

<table>
<thead>
<tr>
<th>Model: MODEL1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependent Variable: Weight</td>
</tr>
</tbody>
</table>

| Parameter Estimates | Parameter Estimate | Standard Error | t Value | Pr > |t |
|---------------------|--------------------|----------------|---------|-------|
| Intercept           | 143.02692          | 32.27459       | -4.43   | 0.0004|
| Height              | 3.89903            | 0.51609        | 7.55    | <0.0001|

The following figure shows a table and graph from the output.
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:

```plaintext
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:

```plaintext
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:

```plaintext
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:

```plaintext
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:

```plaintext
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:

```plaintext
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 609.

By default, a style-element is used for the LINEATTRS= setting. For the REFERENCELINE and DROPLINE statements, the default style element is the GraphReference element. What exactly does this mean?
If we look up the GraphReference element in the DEFAULT style (see Appendix 2, “Graph Style Elements Used by ODS Graphics,” on page 597 for a complete list of all elements and attributes and their defaults), we find the following:

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

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

<table>
<thead>
<tr>
<th>Style Attribute</th>
<th>Description</th>
<th>GTL Suboption</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINETHICKNESS</td>
<td>dimension, most often pixels</td>
<td>THICKNESS</td>
<td>dimension, most often pixels</td>
</tr>
<tr>
<td>LINESTYLE</td>
<td>numeric; 1 to 46, 1 being a solid line</td>
<td>PATTERN</td>
<td>either 1 to 46 or a pattern name, such as SOLID, DASH, DOT (see “Available Line Patterns” on page 612 for examples of available line patterns)</td>
</tr>
<tr>
<td>CONTRASTCOLOR</td>
<td>color specification</td>
<td>COLOR</td>
<td>color specification</td>
</tr>
</tbody>
</table>

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

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

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

First, we will look at what it means to use a different style element for the `LINEATTRS=` option.

When selecting a different style element, you should make sure that the style element does set line properties (graph style elements do not necessarily define all possible attributes). Some reasonable choices might be GraphDataDefault, GraphAxisLines, GraphGridLines, and GraphBorderLines. You might choose GraphGridLines to force a reference line to match the properties of grid lines (if displayed). When you make this 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.

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

- 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:

   \texttt{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:

   \texttt{LINEATTRS=GraphReference(COLOR=GraphAxisLines:ContrastColor)}

• The following example is a shorter form of the previous example:

   \texttt{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:

   \texttt{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.

**MARKERATTRS Option**

Much of what is said about line properties in “LINEATTRS Option” on page 457 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 \texttt{DISPLAY=(MARKERS)} option to turn on the marker display, and also use the \texttt{MARKERATTRS=} option to control the appearance of markers. (The BOXPLOT statement uses \texttt{OUTLIERATTRS=} and \texttt{MEANATTRS=} options).

The following syntax is the complete syntax for the \texttt{MARKERATTRS=} option:

\texttt{MARKERATTRS=style-element | style-element (marker-options) | (marker-options)}

For information about the attributes that you can specify with \texttt{marker-options}, see “Marker Options” on page 610.

The following table shows how the \texttt{MARKERATTRS=} style attributes map to GTL options.

<table>
<thead>
<tr>
<th>Style Attribute</th>
<th>Description</th>
<th>GTL Suboption</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTRASTCOLOR</td>
<td>color specification</td>
<td>COLOR</td>
<td>color specification</td>
</tr>
<tr>
<td>MARKERSIZE</td>
<td>dimension, most often pixels</td>
<td>SIZE</td>
<td>dimension, most often pixels</td>
</tr>
<tr>
<td>MARKERSYMBOL</td>
<td>string (for example, &quot;circle&quot; or &quot;square&quot;)</td>
<td>SYMBOL</td>
<td>predefined keywords such a CIRCLE, SQUARE, TRIANGLE</td>
</tr>
<tr>
<td>none</td>
<td>TRANSTRANSPARENCY</td>
<td>SPECIFICATION</td>
<td>transparency specification, which ranges from 0 (opaque) to 1 (entirely transparent)</td>
</tr>
</tbody>
</table>
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:

\[ \text{TEXTATTRS=style-element | style-element (text-options) | (text-options)} \]

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:

\[ \text{LABELATTRS=style-element | style-element (text-options) | (text-options)} \]

Changing text attributes is fully discussed in Chapter 15, “Adding Titles, Footnotes, and Text Entries to Your Graph,” on page 291.

### Options That Override Style Attributes for All of the Plots in a Template

The 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTPRIORITY=AUTO</td>
<td>COLOR</td>
</tr>
<tr>
<td>DATACOLORS=(color-list)</td>
<td>Specifies the list of fill colors that replace the graph data colors from the GraphData1–GraphDataN style elements.</td>
</tr>
<tr>
<td>DATACONTRASTCOLORS=(color-list)</td>
<td>Specifies the list of contrast colors that replace the graph data contrast colors from the GraphData1–GraphDataN style elements.</td>
</tr>
</tbody>
</table>
Option | Description
---|---
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 508.
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 BEGINGLENTH statement options, see “BEGINGLENTH Statement” in SAS Graph Template Language: Reference.

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 23.2 on page 461 shows examples of each color-naming scheme.

<table>
<thead>
<tr>
<th>Color-Naming Scheme</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>COLOR=(#98FB98)</td>
</tr>
</tbody>
</table>
| CMYK | COLOR=(#FF00FF00)  
COLOR=(#CMYK00FFFF00) |
| HLS | COLOR=(#H14055FF) |
| HSV | COLOR=(#V0F055FF) |
| Gray Scale | COLOR=(GRAY4F) |
Color-Naming Scheme | Example
--- | ---
SAS Registry Colors | COLOR=(palegreen)
CNS Color Names | COLOR=(VeryLightPurplishBlue)

You can also mix color-naming schemes in the same statement. For example:

```plaintext
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.


**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 `CXrrggb`, 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 23.3 Examples of RGB Color Values**

<table>
<thead>
<tr>
<th>Color</th>
<th>RGB Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>CXFF0000</td>
</tr>
<tr>
<td>Green</td>
<td>CX00FF00</td>
</tr>
<tr>
<td>Blue</td>
<td>CX0000FF</td>
</tr>
<tr>
<td>White</td>
<td>CXXFFFFF</td>
</tr>
</tbody>
</table>
Any combination of the color components is valid. Some combinations match the colors produced by predefined SAS color names. See “Predefined Colors” on page 613.

**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 Look-Up 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 CMYKccmmyykk or Kccmmyykk, 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 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.

**Note:** When the color value starts with a number instead of a letter, you must use the CMYK or K prefix.

**Table 23.4 Examples of CMYK Color Values**

<table>
<thead>
<tr>
<th>Color</th>
<th>CMYK Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>CMYK00FFFF00</td>
</tr>
<tr>
<td>Green</td>
<td>FF00FF00</td>
</tr>
<tr>
<td>Blue</td>
<td>CMYKFFFF0000</td>
</tr>
<tr>
<td>White</td>
<td>K00000000</td>
</tr>
<tr>
<td>Process black (using cyan, magenta, and yellow ink)</td>
<td>FFFFFF00</td>
</tr>
<tr>
<td>Pure black (using only black ink)</td>
<td>K0000000FF</td>
</tr>
</tbody>
</table>

**Note:** You can specify a CMY value by specifying zero (00) for kk, the color's black component.
CMYK color specifications are for devices that support four colors. If a CMYK color is used on a three-color device, the device processes the color specification. The resulting colors might not be as expected. Different CMYK colors might map to the same device color because a four-color space supports more colors than a three-color space.

**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 Hhhhllss, 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>
<thead>
<tr>
<th>Color</th>
<th>HLS Color Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>H07880FF</td>
</tr>
<tr>
<td>Green</td>
<td>H0F080FF</td>
</tr>
<tr>
<td>Blue</td>
<td>H00080FF</td>
</tr>
<tr>
<td>Light gray</td>
<td>H000BB00</td>
</tr>
<tr>
<td>White*</td>
<td>HhxxFF00, such as H000FF00</td>
</tr>
<tr>
<td>Black*</td>
<td>Hhxx0000 such as H0000000</td>
</tr>
</tbody>
</table>

* 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 HhxxFF00 and black as Hhxx0000, 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 Vhhhssvv, 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 23.6 Examples of HSV (or HSB) Color Codes

<table>
<thead>
<tr>
<th>Color</th>
<th>HSV Color Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>V000FFFF</td>
</tr>
<tr>
<td>Green</td>
<td>V078FFFF</td>
</tr>
<tr>
<td>Blue</td>
<td>V0F0FFFF</td>
</tr>
<tr>
<td>Light gray*</td>
<td>V.xxx00BB such as V07900BB</td>
</tr>
<tr>
<td>White*</td>
<td>V.xxx00FF such as V07900FF</td>
</tr>
<tr>
<td>Black*</td>
<td>V.xxx0000 such as V0790000</td>
</tr>
</tbody>
</table>

* 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 GRAYhh. 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 467.) 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 23.7  Color Naming System Values

<table>
<thead>
<tr>
<th>Lightness</th>
<th>Saturation</th>
<th>Hue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>Gray</td>
<td>Blue</td>
</tr>
<tr>
<td>Very Dark</td>
<td>Grayish</td>
<td>Purple</td>
</tr>
<tr>
<td>Dark</td>
<td>Moderate</td>
<td>Red</td>
</tr>
<tr>
<td>Medium</td>
<td>Strong</td>
<td>Orange/Brown</td>
</tr>
<tr>
<td>Light</td>
<td>Vivid</td>
<td>Yellow</td>
</tr>
<tr>
<td>Very Light</td>
<td></td>
<td>Green</td>
</tr>
<tr>
<td>White</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
• with a space to separate words, enclosed in quotation marks

For example, all the following are valid color specifications:
• verylightmoderatepurplishblue
• very_light_moderate_purplish_blue
• "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. To use a CNS color value when the color name is also in the SAS Registry, do the following:
• Include a space to separate the words.
• Enclose the entire color name in quotation marks.

**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:

```sas
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 613.

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 *SAS Language Reference: Concepts*.

**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.

<table>
<thead>
<tr>
<th>From Color Value</th>
<th>To Color Value</th>
<th>Conversion Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMY</td>
<td>RGB</td>
<td>%CMY</td>
</tr>
<tr>
<td>CNS</td>
<td>HLS</td>
<td>%CNS</td>
</tr>
<tr>
<td>RGB</td>
<td>HLS</td>
<td>%RGB2HLS</td>
</tr>
<tr>
<td>HLS</td>
<td>RGB</td>
<td>%HLS2RGB</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>From Numeric Color Component Values</th>
<th>To Color Name</th>
<th>Conversion Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>cyan, magenta, yellow, black</td>
<td>CMYK</td>
<td>%CMYK</td>
</tr>
<tr>
<td>hue, lightness, saturation</td>
<td>HLS</td>
<td>%HLS</td>
</tr>
<tr>
<td>hue, saturation, value</td>
<td>HSV</td>
<td>%HSV</td>
</tr>
<tr>
<td>red, green, blue</td>
<td>RGB</td>
<td>%RGB</td>
</tr>
</tbody>
</table>

**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 23.4 on page 469 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 23.4 on page 469, 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 639.) 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:

\[
\left(\frac{\text{Value}}{100}\right) \times 255
\]

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 HSB (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 23.4 on page 469 to an HSV color name.

```sas
%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 23.4 on page 469 to an RGB color name.

```sas
/* Compute the RGB percentages */
data _null_;     r = 231;
```
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.

---

**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";
```

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.

```plaintext
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;
```
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 500. 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, REFERENCESLNe, 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
- BANDPLOT
- BARCHART
- BARCHARTPARM
- BOXPLOT
- BOXPLOTParm
- BUBBLEPLOT
- BUBBLEPLOTParm
- ELLIPSEParm
- FRINGEPLOT
- HEATMAPParm
- HIGHLOWPLOT
- LINECHART
- LINEPARM
- LOESSPLOT
- MOSAICPLOTParm
- MODELBAND
- POLYGONPLOT
- POLYGONPLOT
- PIECHART
- POLYGONPLOT
- SCATTERPLOT
- SCATTERPLOTMATRIX
- SERIESPLOT
- STEPPLOT
- VECTORPLOT
- WATERFALLCHART

Starting with the second maintenance release of SAS 9.4, 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 23.1 Template for a Grouped Series Plot

```sas
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;
```

Executing this template with the default ODS style HTMLBLUE generates the following output.

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

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. The attributes are rotated for each group values as described in “Attribute Rotation Patterns” on page 500. 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 Appendix 2, “Graph Style Elements Used by ODS Graphics,” on page 597 for information about the GTL style elements and attributes. See “Attribute Rotation Patterns” on page 500 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 475. 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 477.

**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 23.2 Custom Style STOCKS**

```plaintext
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
```

![Bubble Plot](image1)

![Scatter Plot](image2)

![Box Plot](image3)

![Pie Chart](image4)
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 612 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 610 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 23.1 on page 473.

```sas
/* 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 494.

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.

<table>
<thead>
<tr>
<th>BEGINGRAPH Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRPRIORITY=</td>
<td>Specifies a priority for the cycling of the attributes from the GraphData1–GraphDataN style elements. See “Attribute Rotation Patterns” on page 500.</td>
</tr>
<tr>
<td>DATACOLORS=</td>
<td>Specifies the list of fill colors that replace the graph data colors from the GraphData1–GraphDataN style elements.</td>
</tr>
<tr>
<td>DATACONTRASTCOLORS=</td>
<td>Specifies the list of contrast colors that replace the graph data contrast colors from the GraphData1–GraphDataN style elements.</td>
</tr>
<tr>
<td>DATASYMBOLS=</td>
<td>Specifies the list of marker symbols that replace the graph data marker symbols from the GraphData1–GraphDataN style elements.</td>
</tr>
<tr>
<td>DATALINEPATTERNS=</td>
<td>Specifies the list of line patterns that replace the graph data line patterns from the GraphData1–GraphDataN style elements.</td>
</tr>
</tbody>
</table>

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.

```sas
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.

![Stock Index Performance - 2002](image)

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 tenth 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 612.)
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.

```sas
proc template;
  define statgraph groupedseries2;
    dynamic year;
    begingroup
      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;
    endgroup;
  end;
end;
run;

proc sgrender data=sashelp.stocks template=groupedseries2;
  where date between '1jan02'd and '31dec02'd;
  format date monnam3.;
  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 ATTRPRIORITRY=NONE option to ensure that color does not take priority in the attribute rotation.

Here is the SAS code for this example.

```sas
proc template;
  define statgraph groupedscatter;
  dynamic year;
  begingraph / attrpriority=none
    datasymbols=(circlefilled squarefilled trianglefilled)
    datacolors=(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 23.8 GROUPDISPLAY Option Values and the Applicable Plots for Each**

<table>
<thead>
<tr>
<th>Value</th>
<th>Applicable Plot Statements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STACK</td>
<td>BARCHART, BARCHARPARM</td>
<td>Stacks each group value on a single bar at the category value on the axis.</td>
</tr>
</tbody>
</table>
| 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. |
<table>
<thead>
<tr>
<th>Value</th>
<th>Applicable Plot Statements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVERLAY</td>
<td>BOXPLOT</td>
<td>Overlays the group values at the category value on the axis.</td>
</tr>
<tr>
<td></td>
<td>BOXPLOTPARM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NEEDLEPLOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCATTERPLOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SERIESPLOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STEPPLOT</td>
<td></td>
</tr>
</tbody>
</table>

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 23.2 on page 475.

![Index Volume By Month for First Six Months of 2002](image)

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.

```sas
/* 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;
  enddefine;
endproc;
```

Controlling the Appearance of Grouped Data 483
endgraph;
end;
run;

/* Generate the bar chart using the bar template. */
ods graphics on / reset outputfmt=static imasename="stocks_&year";;
ods _all_ close;
ods html file="stocks_&year..html" path=typep 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 23.2 on page 475.) 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 exists 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 23.8 on page 481), 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 23.2 on page 475.
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.

<table>
<thead>
<tr>
<th>purchase_</th>
<th>location</th>
<th>method</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Atlanta</td>
<td>In Person</td>
<td>8.4</td>
</tr>
<tr>
<td>2</td>
<td>Atlanta</td>
<td>In Person</td>
<td>7.4</td>
</tr>
<tr>
<td>3</td>
<td>Seattle</td>
<td>In Person</td>
<td>6.4</td>
</tr>
<tr>
<td>4</td>
<td>Vermont</td>
<td>Web</td>
<td>9.0</td>
</tr>
<tr>
<td>5</td>
<td>Vermont</td>
<td></td>
<td>5.0</td>
</tr>
<tr>
<td>6</td>
<td>Vermont</td>
<td>In Person</td>
<td>9.8</td>
</tr>
<tr>
<td>7</td>
<td>Seattle</td>
<td>In Person</td>
<td>8.8</td>
</tr>
<tr>
<td>8</td>
<td>Seattle</td>
<td>Web</td>
<td>9.5</td>
</tr>
<tr>
<td>9</td>
<td>Atlanta</td>
<td>Web</td>
<td>8.5</td>
</tr>
</tbody>
</table>

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.

```sas
proc template;
  define statgraph survey;
$$
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.

```sas
/* 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 SORDERTYPEFMT is then applied to the Purchase_Method column in the PROC SGRENDER statement.

```sas
/* 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 HTMLBlue 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 23.1 on page 473 and the custom style STOCKS that was created in Example Code 23.2 on page 475.

`/* 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
eexample.

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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>Displays the group values in the order in which they appear in the plot data (default)</td>
</tr>
<tr>
<td>ASCENDING</td>
<td>Displays the group values in ascending order</td>
</tr>
</tbody>
</table>
Value | Description
---|---
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-GraphDataN 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 the input data source is modified, sorted, or filtered, the order of the group values and their associations with GraphData1–GraphDataN might change. If you do not care which line pattern, marker symbols, or colors are associated with particular group values, this 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 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, we want the plot to always use the attributes shown in the following table for each plot.

<table>
<thead>
<tr>
<th>Stock</th>
<th>Line Pattern</th>
<th>Marker</th>
<th>Line and Marker Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>solid</td>
<td>filled circle</td>
<td>blue</td>
</tr>
<tr>
<td>Intel</td>
<td>solid</td>
<td>filled square</td>
<td>orange</td>
</tr>
<tr>
<td>Microsoft</td>
<td>solid</td>
<td>filled triangle</td>
<td>dark red</td>
</tr>
</tbody>
</table>

To enforce this consistency, we 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 23.2 on page 475.
Here is the output.

**Figure 23.5  Graph Using a Discrete Attribute Map**

![Graph Using a Discrete Attribute Map](image)

To verify that the plot attributes are consistent regardless of the data order, we can 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.

```sas
/* Sort the SASHELP.STOCKS data by date and stock name. */
proc sort data=sashelp.stocks out=work.stocks;
  by date descending stock;
run;
```

Next, we can generate the graph again using the sorted data set.

```sas
/* Render the graph. */
proc sgrender data= stocks template=groupindex;
  where date between '1jan02'd and '31dec04'D;
run;
```

Here is the result.

**Figure 23.6  Graph with the Data in Descending Order**

![Graph with the Data in Descending Order](image)
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 494.

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 489 modified to use the INDEX= option instead of a discrete attribute map to achieve the same result.

To enforce this consistency, we 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.

```plaintext
/* 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;
```

Chapter 23 • Managing Your Graph’s Appearance
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.

The 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. Here is an example that creates and applies a discrete attribute map to the values in column Stock of the plot data set.

```sas
/* 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
daisplay=(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 HTMLBlue 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 23.7 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 Statement” in *SAS Graph Template Language: Reference* for more information.

**Defining Your Discrete Attribute Map in a SAS Data Set**

Starting with the first maintenance release of SAS 9.4, you can define your discrete attribute map in a SAS data set rather than in your template. 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 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 495, you can create the following data set to replace the DISCRETEATTRMAP block in the template code:

```sas
/* 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.

```sas
/* 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
      attrmaps="stockname";
    layout overlay;
      seriesplot x=date y=close /
        group=stockmarkers
```
/* Plot the stock trends */
proc sgrender data=stocks dattrmap=attrmap template=stockchart;
   dattrvar stock="stockname";
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 23.7 on page 496. 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.
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 your template.
- Associate the attribute map with a response variable in the plot data.
- 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.

```sas
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 343.

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 501 and “The Color-Priority Attribute Rotation Pattern” on page 504.

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 ATTRPRIORIY= option can override the current AttrPriority style attribute setting with NONE or COLOR. (See “ODS GRAPHICS Statement” in SAS ODS Graphics: Procedures Guide.) The BEGINGRAPH statement ATTRPRIORIY= option overrides the ODS GRAPHICS statement ATTRPRIORIY= option for all plots in a BEGINGRAPH block. (See ATTRPRIORIY=.)

With both the default rotation pattern and the color-priority pattern, the plot appearance is dependent on the data order. Any change in the data order can affect the appearance of grouped plots.

**T I P**  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 489.

### 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:* Fill patterns are supported only by selected ODS styles such as JOURNAL2 and MONOCHROMEPRINTER. For more information, see SAS Output Delivery System: User's Guide.

For the contrast colors, marker symbols, fill patterns, and line patterns, when all of the values in the list have been used, the values iterate through the list again. For the area fill colors, when all of the values 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 is being used. The DEFAULT style, for example, defines the attributes that are shown in the following table.

**Table 23.9  Colors, Marker Symbols, and Line Patterns Defined in the DEFAULT Style**

<table>
<thead>
<tr>
<th>Style Element</th>
<th>Color</th>
<th>Contrast Color</th>
<th>Marker Symbol</th>
<th>Line Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphData1</td>
<td>cx7C95CA</td>
<td>cx2A25D9</td>
<td>CIRCLE</td>
<td>1</td>
</tr>
<tr>
<td>GraphData2</td>
<td>cxDE7E6F</td>
<td>cxB2182B</td>
<td>PLUS</td>
<td>4</td>
</tr>
<tr>
<td>GraphData3</td>
<td>cx66A5A0</td>
<td>cx01665E</td>
<td>X</td>
<td>8</td>
</tr>
<tr>
<td>GraphData4</td>
<td>cxA9865B</td>
<td>cx543005</td>
<td>TRIANGLE</td>
<td>5</td>
</tr>
<tr>
<td>GraphData5</td>
<td>cxB689CD</td>
<td>cx9D3CDB</td>
<td>SQUARE</td>
<td>14</td>
</tr>
<tr>
<td>GraphData6</td>
<td>cxBABC5C</td>
<td>cx7F8E1F</td>
<td>ASTERISK</td>
<td>26</td>
</tr>
<tr>
<td>GraphData7</td>
<td>cx94BDE1</td>
<td>cx2597FA</td>
<td>DIAMOND</td>
<td>15</td>
</tr>
<tr>
<td>GraphData8</td>
<td>cxCD7BA1</td>
<td>cxB26084</td>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>
The colors are defined by GDATA1 through GDATA12, and the contrast colors are defined by GCDATA1 through GCDATA12 in the style template.

** See the SYMBOL= option in “Marker Options” on page 610.

*** See “Available Line Patterns” on page 612.

These attribute definitions yield the following results:

- 36 unique area fill colors (12 base colors + 12 lighter colors + 12 darker colors)
- 84 unique marker symbol and color combinations (7 symbols x 12 contrast colors or fill colors)
- 132 unique line color and line pattern combinations (11 patterns x 12 contrast colors)

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.

<table>
<thead>
<tr>
<th>Complete Default Marker Rotation for DEFAULT Style</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Marker Rotation Table" /></td>
</tr>
</tbody>
</table>

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 612). 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.

<table>
<thead>
<tr>
<th>Complete Default Line Rotation for DEFAULT Style</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Line Rotation Table" /></td>
</tr>
</tbody>
</table>

Some styles such as JOURNAL2 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 12 gray-scale fill colors and 15 unique fill patterns, which yield 180 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.

<table>
<thead>
<tr>
<th>Attribute Rotation Patterns</th>
<th>503</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Fill Rotation Table" /></td>
<td></td>
</tr>
</tbody>
</table>
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 23.9 on page 501.

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.

<table>
<thead>
<tr>
<th>Complete Color-Priority Marker Rotation for DEFAULT Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>1   2   3   4   5   6   7   8   9  10  11  12</td>
</tr>
<tr>
<td>+   +   +   +   +   +   +   +   +   +   +   +</td>
</tr>
<tr>
<td>x   x   x   x   x   x   x   x   x   x   x   x</td>
</tr>
<tr>
<td>△   △   △   △   △   △   △   △   △   △   △   △</td>
</tr>
<tr>
<td>□   □   □   □   □   □   □   □   □   □   □   □</td>
</tr>
<tr>
<td>x   x   x   x   x   x   x   x   x   x   x   x</td>
</tr>
<tr>
<td>61  62  63  64  65  66  67  68  69  70  71  72</td>
</tr>
<tr>
<td>73  74  75  76  77  78  79  80  81  82  83  84</td>
</tr>
</tbody>
</table>
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 612). 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.

<table>
<thead>
<tr>
<th>Complete Color-Priority Line Rotation for DEFAULT Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>37</td>
</tr>
<tr>
<td>43</td>
</tr>
<tr>
<td>49</td>
</tr>
<tr>
<td>55</td>
</tr>
<tr>
<td>61</td>
</tr>
<tr>
<td>67</td>
</tr>
<tr>
<td>73</td>
</tr>
<tr>
<td>79</td>
</tr>
<tr>
<td>85</td>
</tr>
<tr>
<td>91</td>
</tr>
<tr>
<td>97</td>
</tr>
<tr>
<td>103</td>
</tr>
<tr>
<td>109</td>
</tr>
<tr>
<td>115</td>
</tr>
<tr>
<td>121</td>
</tr>
<tr>
<td>127</td>
</tr>
</tbody>
</table>

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. The following figure shows the color-priority fill color and fill pattern rotation for the first 36 iterations using the JOURNAL2 style.

<table>
<thead>
<tr>
<th>Partial Color-Priority Fill Color and Fill Pattern Rotation for JOURNAL2 Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-12</td>
</tr>
<tr>
<td>13-24</td>
</tr>
<tr>
<td>25-36</td>
</tr>
</tbody>
</table>
Using Transparency

The 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.

```sas
/* 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 the third maintenance release of SAS 9.4, 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;
```
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.
The next figure shows each of the data skins applied to large HOMEOWNFILLED markers.

**Figure 23.9  Data Skins Applied to HOMEOWNFILLED 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 23.10  Data Skins Applied to Plot Lines

Figure 23.11  Data Skins Applied to Unfilled Bubbles

Figure 23.12  Data Skins Applied to HOMEDOWN Markers
You can specify data skins in the following GTL plot statements:

- BARCHART
- BARCHARTPARM
- BOXPLOT
- BOXPLOTPARM
- BUBBLEPLOT
- DROPLINE
- HIGHLOWPLOT
- NEEDLEPLOT
- POLYGONPLOT
- REFERENCELINE
- PIECHART
- POLYCLINE
- REFERENCELINE
- SERIESPLOT
- SCATTERPLOT
- SCATTERPLOTPARM
- STEPPLOT
- STEPPLOTPARM
- WATERFALLCHART
- VECTORPLOT
- WATERFALLCHART

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;
begingroup;
    entrytitle "Spline Fit";
    layout overlay;
        scatterplot x=height y=weight / primary=true datalabel=name;
        pbsplineplot x=height y=weight;
    endlayout;
endgroup;
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 the third maintenance release of SAS 9.4, 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 the third maintenance release of SAS 9.4, 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 ANTI_ALIAS_MAX= 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 ANTI_ALIAS_MAX= 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 ANTI_ALIAS_MAX= option, see “ODS GRAPHICS Statement” in SAS ODS Graphics: Procedures Guide.

Using Subpixel Rendering

You can specify subpixel rendering in order to generate smooth curves and more precise bar spacing in many plots. In the second maintenance release of SAS 9.4 and in earlier releases, you can use subpixel rendering for plots that are generated with the following statements:

- BANDPLOT
- HIGHLOWPLOT
- REGRESSIONPLOT
- BARCHART
- LINECHART
- SERIESPLOT
- BARCHART_PARM
- LOESSPLOT
- DENSITYPLOT
- PBSPLINEPLOT

Starting with the third maintenance release of SAS 9.4, you can also use subpixel rendering for plots that are generated with the following statements:

- BOXPLOT
- HEATMAP
- SCATTERPLOT
- BOX_PLOT_PARM
- HEATMAP_PARM
- SCATTERPLOT MATRIX
- BUBBLEPLOT
- HIGHLOWPLOT
- WATERFALLCHART
CONTOURPLOTPARM  HISTOGRAMPARM
DENSITYPLOTPLOT  POLYGONPLOTPLOT

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 the second maintenance release of SAS 9.4 and in earlier releases. Starting with the third maintenance release of SAS 9.4, subpixel rendering is always enabled for vector-graphics output. Prior to the third maintenance release of SAS 9.4, 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 the third maintenance release of SAS 9.4, 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.

![Image of a series plot with smoothed lines](image)

Notice that the line curves appear slightly jagged.

To enable subpixel rendering, include the SUBPIXEL=ON option in your BEGINGRAPH statement. Starting with the third maintenance release of SAS 9.4, 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 Statement” in SAS ODS Graphics: Procedures Guide.

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.

```sas
/* 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.
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 448 for more information.
Chapter 24
Managing Your Graphics Output

Introduction to ODS Graphics Output .................................................. 519
SAS Registry Settings for ODS Graphics ............................................. 520
ODS Destination Statement Options That Affect ODS Graphics ............ 521
ODS GRAPHICS Statement Options .................................................. 523
Controlling the Image Name and Image Format ................................... 526
  Specifying and Resetting the Image Name ...................................... 526
  Specifying the Image Format .......................................................... 526
  Using a Universal Printer with ODS PRINTER to Control the Image Format . . . 528
Controlling the Location of the Image Output ...................................... 529
Controlling Graph Size ....................................................................... 531
  Overview of Graph Size Control ...................................................... 531
  BEGINGRAPH Statement ................................................................. 531
Scaling Graphs .................................................................................. 532
Controlling Image Resolution ............................................................ 536
Creating a Graph That Can Be Edited ................................................ 538
Creating a Graph That You Can Import into Microsoft Office Applications . . 540

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:

<table>
<thead>
<tr>
<th>SAS Registry</th>
<th>Provides a repository of defaults for many options that affect ODS Graphics</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODS Destination statement</td>
<td>Provides options specific to destinations, such as HTML, PDF, and RTF</td>
</tr>
<tr>
<td>ODS GRAPHICS statement</td>
<td>Provides many global options that affect ODS graphics</td>
</tr>
</tbody>
</table>

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.

```sas
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.

<table>
<thead>
<tr>
<th>Registry Key</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default State</td>
<td>Determines whether the ODS Graphics environment is active by default</td>
</tr>
<tr>
<td>Design Height</td>
<td>Determines the default height of a graph that is generated with GTL</td>
</tr>
</tbody>
</table>
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 only produce template-based graphics. They internally activate the ODS Graphics environment if it is not active and are unaffected by the Default State key value.

The Design Height and Design Width keys control the default graph size for all graph templates. The 640px by 480px size represents a 4/3 aspect ratio. If you change these values, any new or existing graph templates are affected unless you explicitly set a DESIGNWIDTH= or DESIGNHEIGHT= option in the BEGINGRAPH statement in the graph template definition. For details, see “Controlling Graph Size” on page 531.

---

**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 24.1  ODS Destination Options That Affect ODS Graphics

<table>
<thead>
<tr>
<th>ODS Destination</th>
<th>Options for ODS Graphics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LISTING</td>
<td></td>
<td>Creates a stand-alone image. The default image format is PNG. (See also “Using a Universal Printer with ODS PRINTER to Control the Image Format” on page 528.)</td>
</tr>
<tr>
<td>GPATH=directory-spec</td>
<td></td>
<td>Indicates the directory where images are created. The default is the current working directory.</td>
</tr>
</tbody>
</table>
| 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 the second maintenance release of SAS 9.4 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 starting with the third maintenance release of SAS 9.4. |
| 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. |
## 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 Statement” in SAS ODS Graphics: Procedures Guide.

<table>
<thead>
<tr>
<th>ODS Destination</th>
<th>Options for ODS Graphics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td><strong>FILE=“filename.html”</strong></td>
<td>Specifies the name of the HTML output file.</td>
</tr>
<tr>
<td></td>
<td><strong>PATH=“directory-spec”</strong></td>
<td>Specifies the directory where the HTML output and the images are created. Use the GPATH= option to specify a different directory for the images.</td>
</tr>
<tr>
<td></td>
<td><strong>GPATH=“directory-spec”</strong></td>
<td>Specifies the directory where the images are created. If not specified, the PATH= &quot;directory-spec&quot; is used.</td>
</tr>
<tr>
<td></td>
<td><strong>IMAGE_DPI=number</strong></td>
<td>Specifies the image resolution in dots per inch for output images. IMAGE_DPI=96 is the default.</td>
</tr>
<tr>
<td></td>
<td><strong>STYLE= style-definition</strong></td>
<td>Specifies the style to use. STYLE=HTMLBLUE is the default.</td>
</tr>
<tr>
<td>Task</td>
<td>Option</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td></td>
</tr>
</tbody>
</table>
| 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 the following warning appears in the SAS log:  
WARNING: plot-name statement has too many discrete values. The plot will not be drawn.  
In that case, use the DISCRETEMAX= option to increase the maximum number of discrete values that are allowed. |
| 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. |
| 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. |
<table>
<thead>
<tr>
<th>Task</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control whether legends are drawn.</td>
<td>LEGENDAREAMAX=\textit{n}</td>
</tr>
<tr>
<td></td>
<td>MAXLEGENDAREA=\textit{n}</td>
</tr>
<tr>
<td>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 the third maintenance release of SAS 9.4. Use the LEGENDAREAMAX= option starting with the third maintenance release of SAS 9.4. <strong>Note:</strong> You can continue using the MAXLEGENDAREA= option in the third maintenance release of SAS 9.4 and in later releases. However, the LEGENDAREAMAX= option is preferred.</td>
<td></td>
</tr>
<tr>
<td>Reset one or more ODS GRAPHICS options to its default. RESET by itself is the same as RESET=ALL.</td>
<td>RESET</td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Specify whether the content of any graph is scaled proportionally.</td>
<td>SCALE = ON</td>
</tr>
<tr>
<td>The default is ON.</td>
<td></td>
</tr>
<tr>
<td>Specify whether the plot markers are scaled with the graph size.</td>
<td>SCALEMARKERS=YES</td>
</tr>
<tr>
<td>The default is ON. The scale factor is based on the height of the graph cells and the height of the graph.</td>
<td></td>
</tr>
<tr>
<td>Specify the maximum number of distinct mouse-over areas allowed before data tips are disabled.</td>
<td>TIPMAX=\textit{n}</td>
</tr>
<tr>
<td>The default number is 500.</td>
<td></td>
</tr>
<tr>
<td>Specify the width of any graph.</td>
<td>WIDTH= \textit{dimension}</td>
</tr>
<tr>
<td>Supported dimension units include SPX (special pixels), PX (pixels), IN (inches), CM (centimeters), and MM (millimeters). This option overrides the design width specified by the template definition. The default unit is SPX. You should always provide a unit such as PX, IN, CM, or MM with the dimension value.</td>
<td></td>
</tr>
</tbody>
</table>
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 filename numbering sequence. For example, if you are developing a template 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.

Note: The ability to specify the value for the index counter applies to the third maintenance release in SAS 9.4 and later releases.

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 24.3 Generating Vector Graphics Output with ODS**

<table>
<thead>
<tr>
<th>ODS Destination</th>
<th>OUTPUTFMT=value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODS EPUB</td>
<td>OUTPUTFMT=SVG</td>
</tr>
<tr>
<td>ODS destination for Excel</td>
<td>OUTPUTFMT=EMF</td>
</tr>
<tr>
<td>ODS HTML</td>
<td>OUTPUTFMT=SVG</td>
</tr>
<tr>
<td>ODS LISTING</td>
<td>OUTPUTFMT=EMF</td>
</tr>
<tr>
<td></td>
<td>OUTPUTFMT=PDF</td>
</tr>
<tr>
<td></td>
<td>OUTPUTFMT=PS</td>
</tr>
<tr>
<td></td>
<td>OUTPUTFMT=SVG</td>
</tr>
<tr>
<td></td>
<td>OUTPUTFMT=PCL</td>
</tr>
<tr>
<td>ODS PDF</td>
<td>Vector graphics images are generated by default</td>
</tr>
<tr>
<td>ODS PCL</td>
<td>OUTPUTFMT=PCL (for PCL output)</td>
</tr>
<tr>
<td>ODS PS</td>
<td>OUTPUTFMT=PS (for PostScript output)</td>
</tr>
<tr>
<td>ODS destination for PowerPoint</td>
<td>OUTPUTFMT=EMF</td>
</tr>
<tr>
<td>ODS PRINTER</td>
<td>OUTPUTFMT=PCL (for PCL output)</td>
</tr>
<tr>
<td></td>
<td>OUTPUTFMT=PDF (for PDF output)</td>
</tr>
<tr>
<td></td>
<td>OUTPUTFMT=PS</td>
</tr>
<tr>
<td></td>
<td>OUTPUTFMT=SVG</td>
</tr>
<tr>
<td>ODS RTF</td>
<td>OUTPUTFMT=EMF</td>
</tr>
<tr>
<td>ODS Measured RTF</td>
<td>OUTPUTFMT=EMF</td>
</tr>
</tbody>
</table>

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 the second maintenance release of SAS 9.4, 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 the third maintenance release of SAS 9.4, vector graphics output can be generated in the EMF, PDF, and SVG output formats for the following cases:
• graphs that use data skins
  
  \textit{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
  
  \textit{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.

\textbf{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 \textit{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.
   
   \texttt{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:

```sas
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:

```html
<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:
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).

   proc template;
define statgraph squareplot;
dynamic title xvar yvar;
begingraph / designwidth=defaultDesignHeight;
entrytitle title;
layout overlayequated / equatetype=square;
scatterplot x=xvar y=yvar;
regressionplot x=xvar y=yvar;
endlayout;
endgraph;
end;
run;

If this template were executed with the following SGRENDER specification, a 480px by 480px graph would be created:

```
proc sgrender data=sashelp.cars template="squareplot";
    dynamic title="Square Plot" xvar="mpg_highway" yvar="mpg_city";
run;
```

If a 550px width or height were set in an ODS GRAPHICS statement before the template is executed with the SGRENDER procedure, a 550px by 550px graph would be created, maintaining the 1:1 aspect ratio:

```
ods graphics / width=550px;
proc sgrender data=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.


---

### 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;
```
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;
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 448 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)
```

Here is the result.
The previous two graphs were rendered using the DOCIMAGE style. These next two graphs were rendered with the DOCIMAGE_SMALLFONT style.

```sas
/* 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 */
```

![Boxplot graph](image)

```sas
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.

```sas
/* 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 result.

/* 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 result.

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 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).

```sas
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;
```
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:

```sas
/* 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.

![Graphical User Interface for the ODS Graphics Editor](image-url)

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 dot per inch (DPI) setting, you can consider the default height to be 4.8 inches. If you render a graph at 480 pixels and 100 DPI, insert it into a document like an MS Office application, and then print the page, the graph height on paper will be 4.8 inches and all font sizes will look right in their point weights. You can render the graph at a higher DPI to get higher quality graphs. As long as the graph is then inserted in the document as a 4.8 inch graph, it will work as expected.

To alter the graph size or DPI for a graph that you want to include in an MS Office application, one technique that produces good results is to create a stand-alone image that is sized appropriately and has high resolution, say 200 DPI or 300 DPI.

```ods graphics / reset width=5in imagename="fitplot" 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 511 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 532.

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 8

Templates

Chapter 25
   Executing Graph Templates ........................................ 545

Chapter 26
   Using Dynamics and Macro Variables in Your Templates ...... 555

Chapter 27
   Using Conditional Logic and Expressions in Your Templates . 565

Chapter 28
   Using Functions in Your Templates ................................ 573

Chapter 29
   Sharing Your Custom Templates ................................... 585

Chapter 30
   Modifying Predefined Templates ................................. 587
Chapter 25
Executing Graph Templates

Techniques for Executing Templates ........................................... 545
Minimal Required Syntax .......................................................... 546
Managing the Input Data ........................................................... 547
Filtering the Input Data .............................................................. 547
Performing Data Transformations .................................................. 547
Initializing Template Dynamic Variables and Macro Variables ......... 548
Managing the Output Data Object ................................................ 550
Setting Labels and Formats for the Output Columns ....................... 550
Setting a Name and Label for the Output Data Object ..................... 550
Viewing the Data Object Name and Label in the SAS Windowing Environment .................................................. 551
Setting a Name for the Output Image File ...................................... 552
Converting the Output Data Object to a SAS Data Set ..................... 552

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 25.1 Simple GTL Template

```plaintext
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:

```plaintext
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:

```plaintext
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 25.1 on page 546. 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.

```sas
/* 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 template=mygraphs.scatter;
  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:

```sas
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:
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 25.2 REGFIT Template**

```sas
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`:

```sas
%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.

```sas
data _null_
  if _n_ ^=1 then call symput("study","CLASS data set");
set sashelp.class;
```
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 25.1 on page 546.) It also assigns a format to the WEIGHT variable.

```sas
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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Name with PROC SGRENDER</th>
<th>Default Name with DATA Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT= name</td>
<td>SGRENDER</td>
<td>FilePrint&lt;sub&gt;n&lt;/sub&gt; (each execution of the DATA step increments the object name: FilePrint1, FilePrint2, and so on)</td>
</tr>
<tr>
<td>OBJECTLABEL=&quot;string&quot;</td>
<td>The SGRENDER Procedure</td>
<td>same as object name</td>
</tr>
</tbody>
</table>
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:

```sas
/* 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;
```

The following example sets the object name and label on a DATA step:

```sas
/* 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 550. 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 25.2 on page 548).

```sas
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:

```sas
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:

```sas
ods output dataobj=regfit2;
```
data _null_;  
if _n_=1 then call symput("study","CLASS dataset");
set sashelp.class;
file print ods=(
    template="mygraphs.regfit"
    dynamic=(
        xvar="height" yvar="weight"
    )
    object=dataobj
);
put _ods_;  
run;
Chapter 26
Using Dynamics 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;

Dynamic variables and run-time macro variable references cannot be used in place of
statement or option keywords, or in place of punctuation that is part of the syntax
(parentheses, semicolons, and so on).

Note: If you precede a macro variable reference with an ampersand (&), the reference
will be resolved when the template is compiled, not when it is executed.

For example, it is permissible to define TRANS as an MVAR for use in the following
context:

proc template;
define statgraph timeseries;
  dynamic YVAR YLABEL;
  mvar LOCATE TRANS;
begingraph;
  layout overlay;
    seriesplot x=date y=YVAR  / curvelabel=YLABEL
      curvelabellocation= LOCATE datatransparency= TRANS;
  endlayout;
endgraph;
end;
run;

This context is valid because an automatic, internal conversion using the BEST. format will be performed (with no warning messages).

Initializing 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.

```sas
proc sgrender data=financial template=timeseries;
  dynamic yvar="inflation" ylabel="Inflation Rate";
run;
```

Values for dynamic variables that resolve to column names or strings should be quoted. Numeric values should not be quoted.

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.

```sas
%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 27, “Using Conditional Logic and Expressions in Your Templates,” on page 565.

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.

```sas
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) columnarange=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.

```sas
proc sgrender data=sashelp.heart template=distribution;
  dynamic var="Cholesterol"
    footnote="From Framingham Heart Study (SASHELP.HEART)";
run;
```
In this case, the template option `bins=BINS` drops out because the BINS dynamic 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.

```sas
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.

```sas
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 17, “Adding Insets to Your Graph,” on page 351.

We will now define a macro that can pass values to this template. For a given numeric variable, the macro computes the number of observations, the mean, and the standard deviation, storing these statistics in macro variables N, MEAN, and STD. The macro variables are available to the SGRENDER step when the macro executes. Here is the definition for the macro, which we will name HIST.

```sas
%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 on based on the amount of space that is available and the setting for the AUTOALIGN= option.

```sas
%hist(sashelp.heart, cholesterol, From SASHELP.HEART)
```
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 27, “Using Conditional Logic and Expressions in Your Templates,” on page 565.

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 27, “Using Conditional Logic and Expressions in Your Templates,” on page 565.

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 585.

### Special Dynamic Variables

Several special predefined dynamic variables are available that you can use with your templates. These special variables are listed in Table 26.1 on page 562.

**Table 26.1** Special Dynamic Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LIBNAME</strong></td>
<td>This variable represents the name of the library that contains the data set.</td>
</tr>
<tr>
<td><strong>MEMNAME</strong></td>
<td>This variable represents the name of the library member that contains the data set.</td>
</tr>
<tr>
<td><strong>BYFOOTNOTE</strong></td>
<td>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 Statement” in SAS ODS Graphics: Procedures Guide.</td>
</tr>
<tr>
<td><strong>BYLINE</strong></td>
<td>This variable represents the complete BY line when you specify a BY statement.</td>
</tr>
<tr>
<td><strong>BYTITLE</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>BYVAR</strong></td>
<td>This variable represents the name of the first BY variable when you specify a BY statement.</td>
</tr>
<tr>
<td><strong>BYVARn</strong></td>
<td>This variable represents the name of the nth BY variable when you specify a BY statement with multiple variables.</td>
</tr>
<tr>
<td><strong>BYVAL</strong></td>
<td>This variable represents the first BY value when you specify a BY statement.</td>
</tr>
<tr>
<td><strong>BYVALn</strong></td>
<td>This variable represents the value of the nth BY variable when you specify a BY statement with multiple variables.</td>
</tr>
</tbody>
</table>

* This variable is set automatically only for analytical procedures that support ODS GRAPHICS. For a list of these procedures, see “Automatic Graphics from SAS Analytical Procedures” on page 5. 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, the default BY line that is normally displayed when the BY statement is used is disabled.

Here is the code for this example.

```sas
/* 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";
  endgraph;
endproc;
```

*Special Dynamic Variables* 563
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.
Chapter 27
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 27, “Using Conditional Logic and Expressions in Your Templates,” on page 565.

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 (TRUE).

Here are some examples:

```gxl
/* 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;
```
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.

```gson
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:

```gson
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:

```gson
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:

```gson
/* 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:

```gson
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:

```plaintext
proc template;
  define statgraph conditional;
  dynamic NUMVAR "required" SCALE CURVE;
  begingraph;
    entrytitle "Distribution of " eval(colname(NUMVAR));
    if (colname(NUMVAR) ne collabel(NUMVAR))
      entrytitle "(" eval(collabel(NUMVAR)) ");
    endif;
    layout overlay / xaxisopts=(display=(ticks tickvalues line));
    histogram NUMVAR / scale=SCALE;
    if (upcase(CURVE) in (*ALL* "NORMAL") )
      densityplot NUMVAR / normal() name="N"
      lineattrs=GraphData1 legendlabel="Normal Distribution";
    endif;
    if (upcase(CURVE) in (*ALL* "KDE" "KERNEL") )
      densityplot NUMVAR / kernel() name="K"
      lineattrs=GraphData2 legendlabel="Kernel Density Estimate";
    endif;
    discretelegend "N" "K";
  endlayout;
  endgraph;
end;
run;
```

- The DYNAMIC statement identifies the dynamic variables.
- The first IF block specifies an ENTRYTITLE statement that is conditionally executed if the column name differs from the column label.
- The next two IF blocks evaluate the value of the dynamic variable CURVE. If CURVE is not used, the code in the conditional blocks is not executed. If CURVE is initialized to one of the strings "all" or "normal" in any letter case, then the first DENSITYPLOT statement is executed. If CURVE is initialized to one of the strings "all", "kde", or "kernel" in any letter case, then the second DENSITYPLOT statement is executed. Thus, the results of the conditional logic determine whether zero, one, or two density plots are generated in the graph.
• Constructing the legend does not require conditional logic because any referenced plot names that do not exist are not used.

After submitting the template code, we can execute the template with various combinations of dynamic values.

In this first execution, the NUMVAR dynamic 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.

![Distribution of MRW
(Metropolitan Relative Weight)](image)

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;
```
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.
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 SAS 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.

<table>
<thead>
<tr>
<th>SAS Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFC(logical-expression, &quot;true-value&quot;, &quot;false-value&quot; &lt;,&quot;missing-value&quot;&gt;)</td>
<td>Returns the character 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. The TRUE, FALSE, and MISSING values must be enclosed in quotation marks.</td>
</tr>
<tr>
<td>IFN(logical-expression, true-value, false-value &lt;,missing-value&gt;)</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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.

![Sales-Based Commission Graph]

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, we can use the IFC function for this purpose. This saves us from having to add a new column to the data in a DATA step or having to create and apply a custom format to the SalesUnit column.

Here is the SAS code that defines the template and generates the graph.

```sas
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.
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLC(“string-1”, “string-1&quot;&lt;, “string-n&quot; ...)</td>
<td>Converts a list of comma-separated string values into a temporary character column. Starting with the first maintenance release of SAS 9.4, you can use this function to specify values in options that accept a character column.</td>
</tr>
<tr>
<td>COLN(n-1, n-1&lt;n-N...)</td>
<td>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.</td>
</tr>
<tr>
<td>COLNAME(column)</td>
<td>Returns the case-sensitive name of the column.</td>
</tr>
<tr>
<td>COLLABEL(column)</td>
<td>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.</td>
</tr>
<tr>
<td>EXISTS(item)</td>
<td>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.</td>
</tr>
<tr>
<td>EXPAND(numeric-column, freq-column)</td>
<td>Creates a new column as (numeric-column * frequency-column) .</td>
</tr>
</tbody>
</table>
| 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). |

**Functions Defined Only in GTL**

577
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.

![Graph of Type and MPG (City)](image)

Here is the SAS code.

```sas
/* 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;
    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 576 for information about the COLLABEL function.
GTL Summary Statistic Functions

Commonly Used Summary Statistic Functions

The following functions 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. These functions take precedence over similar multi-argument DATA step functions.

\[
\text{number} = \text{EVAL(function-name(numeric-column))}
\]

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS</td>
<td>Corrected sum of squares</td>
</tr>
<tr>
<td>CV</td>
<td>Coefficient of variation</td>
</tr>
<tr>
<td>KURTOSIS</td>
<td>Kurtosis</td>
</tr>
<tr>
<td>LCLM</td>
<td>One-sided confidence limit below the mean</td>
</tr>
<tr>
<td>MAX</td>
<td>Largest (maximum) value</td>
</tr>
<tr>
<td>MEAN</td>
<td>Mean</td>
</tr>
<tr>
<td>MEDIAN</td>
<td>Median (50th percentile)</td>
</tr>
<tr>
<td>MIN</td>
<td>Smallest (minimum) value</td>
</tr>
<tr>
<td>N</td>
<td>Number of nonmissing values</td>
</tr>
<tr>
<td>NMISS</td>
<td>Number of missing values</td>
</tr>
<tr>
<td>P1</td>
<td>1st percentile</td>
</tr>
<tr>
<td>P5</td>
<td>5th percentile</td>
</tr>
<tr>
<td>P25</td>
<td>25th percentile</td>
</tr>
<tr>
<td>P50</td>
<td>50th percentile</td>
</tr>
<tr>
<td>P75</td>
<td>75th percentile</td>
</tr>
<tr>
<td>P90</td>
<td>90th percentile</td>
</tr>
<tr>
<td>P95</td>
<td>95th percentile</td>
</tr>
<tr>
<td>P99</td>
<td>99th percentile</td>
</tr>
</tbody>
</table>
**number = EVAL(function-name(numeric-column))**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROBT</td>
<td>p-value for Student’s t statistic</td>
</tr>
<tr>
<td>Q1</td>
<td>First quartile</td>
</tr>
<tr>
<td>Q3</td>
<td>Third quartile</td>
</tr>
<tr>
<td>QRANGE</td>
<td>Interquartile range</td>
</tr>
<tr>
<td>RANGE</td>
<td>Range</td>
</tr>
<tr>
<td>SKEWNESS</td>
<td>Skewness</td>
</tr>
<tr>
<td>STDDEV</td>
<td>Standard deviation</td>
</tr>
<tr>
<td>STDERR</td>
<td>Standard error of the mean</td>
</tr>
<tr>
<td>SUM</td>
<td>Sum</td>
</tr>
<tr>
<td>SUMWGT</td>
<td>Sum of weights</td>
</tr>
<tr>
<td>T</td>
<td>Student’s t statistic</td>
</tr>
<tr>
<td>UCLM</td>
<td>One-sided confidence limit above the mean</td>
</tr>
<tr>
<td>USS</td>
<td>Uncorrected sum of squares</td>
</tr>
<tr>
<td>VAR</td>
<td>Variance</td>
</tr>
</tbody>
</table>

**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.

![Distribution of MRW](image)

Here is the SAS code.

```sas
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;
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.

```sas
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.

```sas
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;

<table>
<thead>
<tr>
<th>Current ODS PATH list is:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. P.TEMPLAT(READ)</td>
</tr>
<tr>
<td>2. SASUSER.TEMPLAT(UPDATE)</td>
</tr>
<tr>
<td>3. SASHHELP.TMPLMST(READ)</td>
</tr>
</tbody>
</table>
Chapter 30
Modifying Predefined Templates

Predefined Templates for SAS Analytical Procedures .......................... 587
Modify a Predefined Template ................................................................. 589

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. You can find a list of the SAS analytical procedures that support ODS Graphics in “Automatic Graphics from SAS Analytical Procedures” on page 5. 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.

```sas
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;
odbs trace off;
```

In this case, the FREQ procedure outputs 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.
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.

```
odspath show;
```

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;
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 `ODSTEMPLES` in the command line, and then press Enter.

For more information about the ODS Graphics predefined templates, see *SAS/STAT User's Guide*. 

---

<table>
<thead>
<tr>
<th>Output Added:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
</tr>
<tr>
<td>Label:</td>
</tr>
<tr>
<td>Template:</td>
</tr>
<tr>
<td>Path:</td>
</tr>
<tr>
<td>---------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Added:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
</tr>
<tr>
<td>Label:</td>
</tr>
<tr>
<td>Template:</td>
</tr>
<tr>
<td>Path:</td>
</tr>
</tbody>
</table>

---
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 587.

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 9

Appendixes

Appendix 1
  Reserved Keywords and Unicode Values .......................... 593

Appendix 2
  Graph Style Elements Used by ODS Graphics ...................... 597

Appendix 3
  Display Attributes .................................................. 607

Appendix 4
  Predefined Colors .................................................. 613

Appendix 5
  Tick Value Fit Policy Applicability ............................... 627

Appendix 6
  SAS Formats Not Supported ........................................ 631

Appendix 7
  Memory Management for ODS Graphics ............................ 635

Appendix 8
  Understanding Hexadecimal Values ................................ 639

Appendix 9
  ODS Graphics and SAS/GRAPH ...................................... 643
Appendix 1
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 297.

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

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Glyph</th>
<th>Unicode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>α</td>
<td>03B1</td>
<td>lowercase alpha</td>
</tr>
<tr>
<td>beta</td>
<td>β</td>
<td>03B2</td>
<td>lowercase beta</td>
</tr>
<tr>
<td>gamma</td>
<td>γ</td>
<td>03B3</td>
<td>lowercase gamma</td>
</tr>
<tr>
<td>delta</td>
<td>δ</td>
<td>03B4</td>
<td>lowercase delta</td>
</tr>
<tr>
<td>epsilon</td>
<td>ε</td>
<td>03B5</td>
<td>lowercase epsilon</td>
</tr>
<tr>
<td>zeta</td>
<td>ζ</td>
<td>03B6</td>
<td>lowercase zeta</td>
</tr>
<tr>
<td>eta</td>
<td>η</td>
<td>03B7</td>
<td>lowercase eta</td>
</tr>
<tr>
<td>theta</td>
<td>θ</td>
<td>03B8</td>
<td>lowercase theta</td>
</tr>
<tr>
<td>iota</td>
<td>ι</td>
<td>03B9</td>
<td>lowercase iota</td>
</tr>
<tr>
<td>kappa</td>
<td>κ</td>
<td>03BA</td>
<td>lowercase kappa</td>
</tr>
<tr>
<td>lambda</td>
<td>λ</td>
<td>03BB</td>
<td>lowercase lambda</td>
</tr>
<tr>
<td>mu</td>
<td>μ</td>
<td>03BC</td>
<td>lowercase mu</td>
</tr>
</tbody>
</table>
### Reserved Keywords and Unicode Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Glyph</th>
<th>Unicode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nu</td>
<td>ν</td>
<td>03BD</td>
<td>lowercase nu</td>
</tr>
<tr>
<td>xi</td>
<td>ζ</td>
<td>03BE</td>
<td>lowercase xi</td>
</tr>
<tr>
<td>omicron</td>
<td>ο</td>
<td>03BF</td>
<td>lowercase omicron</td>
</tr>
<tr>
<td>pi</td>
<td>π</td>
<td>03C0</td>
<td>lowercase pi</td>
</tr>
<tr>
<td>rho</td>
<td>ρ</td>
<td>03C1</td>
<td>lowercase rho</td>
</tr>
<tr>
<td>sigma</td>
<td>σ</td>
<td>03C3</td>
<td>lowercase sigma</td>
</tr>
<tr>
<td>tau</td>
<td>τ</td>
<td>03C4</td>
<td>lowercase tau</td>
</tr>
<tr>
<td>upsilon</td>
<td>υ</td>
<td>03C5</td>
<td>lowercase upsilon</td>
</tr>
<tr>
<td>phi</td>
<td>φ</td>
<td>03C6</td>
<td>lowercase phi</td>
</tr>
<tr>
<td>chi</td>
<td>χ</td>
<td>03C7</td>
<td>lowercase chi</td>
</tr>
<tr>
<td>psi</td>
<td>ψ</td>
<td>03C8</td>
<td>lowercase psi</td>
</tr>
<tr>
<td>omega</td>
<td>ω</td>
<td>03C9</td>
<td>lowercase omega</td>
</tr>
</tbody>
</table>

### Uppercase Greek Letters

#### Table A1.1 Uppercase Greek Letters

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Glyph</th>
<th>Unicode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha_u</td>
<td>A</td>
<td>0391</td>
<td>uppercase alpha</td>
</tr>
<tr>
<td>beta_u</td>
<td>B</td>
<td>0392</td>
<td>uppercase beta</td>
</tr>
<tr>
<td>gamma_u</td>
<td>Γ</td>
<td>0393</td>
<td>uppercase gamma</td>
</tr>
<tr>
<td>delta_u</td>
<td>Δ</td>
<td>0394</td>
<td>uppercase delta</td>
</tr>
<tr>
<td>epsilon_u</td>
<td>E</td>
<td>0395</td>
<td>uppercase epsilon</td>
</tr>
<tr>
<td>zeta_u</td>
<td>Z</td>
<td>0396</td>
<td>uppercase zeta</td>
</tr>
<tr>
<td>eta_u</td>
<td>Η</td>
<td>0397</td>
<td>uppercase eta</td>
</tr>
<tr>
<td>theta_u</td>
<td>Θ</td>
<td>0398</td>
<td>uppercase theta</td>
</tr>
<tr>
<td>iota_u</td>
<td>Ι</td>
<td>0399</td>
<td>uppercase iota</td>
</tr>
<tr>
<td>Keyword</td>
<td>Glyph</td>
<td>Unicode</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>---------</td>
<td>---------------------</td>
</tr>
<tr>
<td>kappa_u</td>
<td>K</td>
<td>039A</td>
<td>uppercase kappa</td>
</tr>
<tr>
<td>lambda_u</td>
<td>Λ</td>
<td>039B</td>
<td>uppercase lambda</td>
</tr>
<tr>
<td>mu_u</td>
<td>M</td>
<td>039C</td>
<td>uppercase mu</td>
</tr>
<tr>
<td>nu_u</td>
<td>N</td>
<td>039D</td>
<td>uppercase nu</td>
</tr>
<tr>
<td>xi_u</td>
<td>Ξ</td>
<td>039E</td>
<td>uppercase xi</td>
</tr>
<tr>
<td>omicron_u</td>
<td>O</td>
<td>039F</td>
<td>uppercase omicron</td>
</tr>
<tr>
<td>pi_u</td>
<td>Π</td>
<td>03A0</td>
<td>uppercase pi</td>
</tr>
<tr>
<td>rho_u</td>
<td>P</td>
<td>03A1</td>
<td>uppercase rho</td>
</tr>
<tr>
<td>sigma_u</td>
<td>Σ</td>
<td>03A3</td>
<td>uppercase sigma</td>
</tr>
<tr>
<td>tau_u</td>
<td>T</td>
<td>03A4</td>
<td>uppercase theta</td>
</tr>
<tr>
<td>upsilon_u</td>
<td>Y</td>
<td>03A5</td>
<td>uppercase upsilon</td>
</tr>
<tr>
<td>phi_u</td>
<td>Φ</td>
<td>03A6</td>
<td>uppercase phi</td>
</tr>
<tr>
<td>chi_u</td>
<td>Χ</td>
<td>03A7</td>
<td>uppercase chi</td>
</tr>
<tr>
<td>psi_u</td>
<td>Ψ</td>
<td>03A8</td>
<td>uppercase psi</td>
</tr>
<tr>
<td>omega_u</td>
<td>Ω</td>
<td>03A9</td>
<td>uppercase omega</td>
</tr>
</tbody>
</table>

**Special Characters**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Glyph</th>
<th>Unicode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prime</td>
<td>′</td>
<td>00B4</td>
<td>single prime sign</td>
</tr>
<tr>
<td>bar</td>
<td>−</td>
<td>0305</td>
<td>combining overline</td>
</tr>
<tr>
<td>bar2</td>
<td>−</td>
<td>033F</td>
<td>combining double overline</td>
</tr>
<tr>
<td>tilde</td>
<td>~</td>
<td>0303</td>
<td>combining tilde</td>
</tr>
<tr>
<td>hat</td>
<td>*</td>
<td>0302</td>
<td>combining circumflex accent</td>
</tr>
</tbody>
</table>

* 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 ........................................ 597
General Graph Appearance Style Elements .............................. 597
Graphical Data Representation Style Elements (Non-Grouped Data) .... 600
Graphical Data Representation Style Elements (Grouped Data) ......... 604
Display Style Elements .......................................................... 605

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

<table>
<thead>
<tr>
<th>Style Element</th>
<th>Portion of Graph Affected</th>
<th>Recognized Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>Graph size and outer border appearance</td>
<td>OutputWidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OutputHeight</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BorderColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BorderWidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CellPadding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CellSpacing</td>
</tr>
<tr>
<td>GraphAnnoLine</td>
<td>Annotation lines</td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td>GraphAnnoShape</td>
<td>Annotation closed shapes such as circles, and squares</td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transparency</td>
</tr>
<tr>
<td>GraphAnnoText</td>
<td>Annotation text</td>
<td>Font</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or font-attributes*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MarkerSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MarkerSymbol</td>
</tr>
<tr>
<td>GraphAxisLines</td>
<td>X, Y and Z axis lines</td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TickDisplay</td>
</tr>
<tr>
<td>GraphBackground</td>
<td>Background of the graph</td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transparency</td>
</tr>
<tr>
<td>GraphBorderLines</td>
<td>Border around graph wall, legend border, borders to complete axis frame</td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td>GraphDataText</td>
<td>Text font and color for point and line labels</td>
<td>Font</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or font-attributes*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color</td>
</tr>
<tr>
<td>GraphFootnoteText</td>
<td>Text font and color for footnote(s)</td>
<td>Font</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or font-attributes*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color</td>
</tr>
<tr>
<td>Style Element</td>
<td>Portion of Graph Affected</td>
<td>Recognized Attributes</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>GraphGridLines</td>
<td>Horizontal and vertical grid lines drawn at major tick marks</td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DisplayOpts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td>GraphHeaderBackground</td>
<td>Background color of the legend title</td>
<td>Color</td>
</tr>
<tr>
<td>GraphLabelText</td>
<td>Text font and color for axis labels and legend titles</td>
<td>Font</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>font-attributes</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color</td>
</tr>
<tr>
<td>GraphLegendBackground</td>
<td>Background color of the legend</td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FrameBorder</td>
</tr>
<tr>
<td>GraphMinorGridLines</td>
<td>Appearance of the grid lines.</td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DisplayOpts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td>GraphOutlines</td>
<td>Outline properties for fill areas such as bars, pie slices, box plots, ellipses, and</td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td>histograms</td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td>GraphReference</td>
<td>Horizontal and vertical reference lines and drop lines</td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td>GraphTitleText</td>
<td>Text font and color for title(s)</td>
<td>Font</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>font-attributes</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color</td>
</tr>
<tr>
<td>GraphUnicodeText</td>
<td>Text font for Unicode values</td>
<td>Font</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>font-attributes</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color</td>
</tr>
<tr>
<td>GraphValueText</td>
<td>Text font and color for axis tick values and legend values</td>
<td>Font</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>font-attributes</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color</td>
</tr>
</tbody>
</table>
Graphical Data Representation Style Elements
(Non-Grouped Data)

The following table lists the graphical data representation style elements that affect non-grouped data.

*Font-attributes can be one of the following: `FONTFAMILY=`, `FONTSIZE=`, `FONTSTYLE=`, `FONTWEIGHT=`.

<table>
<thead>
<tr>
<th>Style Element</th>
<th>Portion of Graph Affected</th>
<th>Recognized Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphWalls</td>
<td>Vertical wall(s) bounded by axes</td>
<td>Color, FrameBorder, LineThickness, LineStyle, ContrastColor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Style Element</th>
<th>Portion of Graph Affected</th>
<th>Recognized Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphBoxMean</td>
<td>Marker for mean</td>
<td>ContrastColor, MarkerSize, MarkerSymbol</td>
</tr>
<tr>
<td>GraphBoxMedian</td>
<td>Line for median</td>
<td>ContrastColor, LineStyle, LineThickness</td>
</tr>
<tr>
<td>GraphBoxWhisker</td>
<td>Box whiskers and serifs</td>
<td>ContrastColor, LineStyle, LineThickness</td>
</tr>
<tr>
<td>GraphConfidence</td>
<td>Primary confidence lines and bands, colors for bands and lines</td>
<td>ContrastColor, Color, MarkerSize, MarkerSymbol, LineStyle, LineThickness</td>
</tr>
<tr>
<td>Style Element</td>
<td>Portion of Graph Affected</td>
<td>Recognized Attributes</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GraphConfidence2</td>
<td>Secondary confidence lines and bands, color for bands, and contrast color for lines</td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MarkerSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MarkerSymbol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td>GraphConnectLine</td>
<td>Line for connecting boxes or bars</td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td>GraphDataDefault</td>
<td>Primitives related to non-grouped data items, colors for filled areas, markers, and lines</td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MarkerSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MarkerSymbol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>StartColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NeutralColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EndColor</td>
</tr>
<tr>
<td>GraphCutLine</td>
<td>Cutline attributes for a dendogram</td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td>GraphDataDefault</td>
<td>Primitives related to non-grouped data items, colors for filled areas, markers, and lines</td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MarkerSymbol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MarkerSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineThickness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>StartColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NeutralColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EndColor</td>
</tr>
<tr>
<td>GraphError</td>
<td>Error line or error bar fill, ContrastColor for lines, Color for bar fill</td>
<td>CapStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ContrastColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LineStyle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transparency</td>
</tr>
<tr>
<td>Style Element</td>
<td>Portion of Graph Affected</td>
<td>Recognized Attributes</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>GraphFit</td>
<td>Primary fit lines such as a normal density curve</td>
<td>ContrastColor, Color, MarkerSize, MarkerSymbol, LineStyle, LineThickness</td>
</tr>
<tr>
<td>GraphFit2</td>
<td>Secondary fit lines such as a kernel density curve</td>
<td>ContrastColor, Color, MarkerSize, MarkerSymbol, LineStyle, LineThickness</td>
</tr>
<tr>
<td>GraphFinal</td>
<td>Final data for the waterfall chart. Color applies to filled areas.</td>
<td>Color, ContrastColor, LineStyle, LineThickness, MarkerSize, MarkerSymbol</td>
</tr>
<tr>
<td>GraphInitial</td>
<td>Initial data for the waterfall chart. Color applies to filled areas.</td>
<td>Color, ContrastColor, LineStyle, LineThickness, MarkerSize, MarkerSymbol</td>
</tr>
<tr>
<td>GraphMissing</td>
<td>Properties for graph items representing missing values</td>
<td>ContrastColor, Color, MarkerSymbol, MarkerSize, LineStyle, LineThickness, Transparency</td>
</tr>
<tr>
<td>GraphOther</td>
<td>Other data for the graph. Color applies to filled areas.</td>
<td>Color, ContrastColor, LineStyle, LineThickness, MarkerSize, MarkerSymbol</td>
</tr>
<tr>
<td>Style Element</td>
<td>Portion of Graph Affected</td>
<td>Recognized Attributes</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>GraphOverflow</td>
<td>Overflow data for the graph. Color applies to filled areas. ContrastColor applies to markers and lines.</td>
<td>Color, ContrastColor, LineStyle, LineThickness, MarkerSize, MarkerSymbol</td>
</tr>
<tr>
<td>GraphOutlier</td>
<td>Outlier data for the graph</td>
<td>ContrastColor, Color, MarkerSize, MarkerSymbol, LineStyle, LineThickness</td>
</tr>
<tr>
<td>GraphPrediction</td>
<td>Prediction lines</td>
<td>ContrastColor, Color, LineStyle, LineThickness, MarkerSize, MarkerSymbol</td>
</tr>
<tr>
<td>GraphPredictionLimits</td>
<td>Fills for prediction limits</td>
<td>ContrastColor, Color, MarkerSize, MarkerSymbol</td>
</tr>
<tr>
<td>GraphUnderflow</td>
<td>Underflow data for the graph. Color applies to filled areas. ContrastColor applies to markers and lines.</td>
<td>Color, ContrastColor, LineStyle, LineThickness, MarkerSize, MarkerSymbol, TextColor</td>
</tr>
<tr>
<td>GraphSelection</td>
<td>For interactive graphs, visual properties of selected item. Color for selected fill area, ContrastColor for selected marker or line.</td>
<td>ContrastColor, Color, MarkerSymbol, MarkerSize, LineStyle, LineThickness</td>
</tr>
</tbody>
</table>
## Graphical Data Representation Style Elements (Grouped Data)

The following table lists and describes the data-related style elements that effect grouped data.

**Table A2.3  Graphical Style Elements: Data Related (Grouped)**

<table>
<thead>
<tr>
<th>Style Elements</th>
<th>Portion of Graph Affected</th>
<th>Recognized Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphData1</td>
<td>Primitives related to the first 7 grouped data items. Color applies to filled areas. ContrastColor applies to markers and lines.</td>
<td>Color, ContrastColor, FillPattern*</td>
</tr>
<tr>
<td>GraphData2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphData3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphData4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphData5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphData6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphData7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphData8</td>
<td>Primitives related to the 8th through 11th grouped data items.</td>
<td>Color, ContrastColor, FillPattern*</td>
</tr>
<tr>
<td>GraphData9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphData10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphData11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphData12</td>
<td>Primitives related to the 12th grouped data item.</td>
<td>Color, ContrastColor, FillPattern*</td>
</tr>
</tbody>
</table>
Display Style Elements

The following table lists the display style elements.

<table>
<thead>
<tr>
<th>Style Element</th>
<th>Portion of Graph Affected</th>
<th>Recognized Attributes</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphAltBlock</td>
<td>Alternate fill color for block plots</td>
<td>Color</td>
<td>GraphColors(&quot;gablock&quot;)</td>
</tr>
<tr>
<td>GraphBand</td>
<td>Display options for confidence bands</td>
<td>DisplayOpts</td>
<td>&quot;Fill&quot;</td>
</tr>
<tr>
<td>GraphBar</td>
<td>Display options for bar charts</td>
<td>DisplayOpts</td>
<td>&quot;Fill outline&quot;</td>
</tr>
<tr>
<td>GraphBox</td>
<td>Display options for box plots</td>
<td>DisplayOpts, CapStyle, Connect</td>
<td>&quot;Fill caps mean&quot;, &quot;Median outliers&quot;, &quot;Serif&quot;, &quot;Mean&quot;</td>
</tr>
<tr>
<td>GraphBlock</td>
<td>Fill color for block plots</td>
<td>Color</td>
<td>GraphColors(&quot;gblock&quot;)</td>
</tr>
<tr>
<td>GraphEllipse</td>
<td>Display options for confidence ellipses</td>
<td>DisplayOpts</td>
<td>&quot;Outline&quot;</td>
</tr>
<tr>
<td>GraphHistogram</td>
<td>Display options for histograms</td>
<td>DisplayOpts</td>
<td>&quot;Fill outline&quot;</td>
</tr>
<tr>
<td>Style Element</td>
<td>Portion of Graph Affected</td>
<td>Recognized Attributes</td>
<td>Possible Values</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------</td>
<td>-----------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>GraphSkins</td>
<td>One or more display features</td>
<td>DataSkin</td>
<td>CRISP, GLOSS, MATTE, NONE, PRESSED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KpiSkin</td>
<td></td>
<td>BASIC</td>
<td>MODERN, NONE, ONYX, SATIN</td>
</tr>
</tbody>
</table>
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:

\[ \text{ATTRSOPTIONNAME} = \text{style-element | style-element (options) | (options)} \]

**style-element**
Name of a style element. Only style attributes relevant for rendering the fill, line, data marker, or text are used.

Example

\[ \text{ATTRSOPTIONNAME}=\text{GRAPHVALUETEXT} \]

**style-element (options)**
Name of a style element, plus individual options to be used as style overrides. Any options not specified are derived from the specified style element.

Example

\[ \text{ATTRSOPTIONNAME}=\text{GRAPHVALUETEXT (SIZE=10pt)} \]

**(options)**
Individual options. Any options not specified are derived from the default style element.
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 597 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 options, line options, marker options, or text options.

### Fill Options

When specifying the attributes for an area fill, the fill options can be one or more of the following settings. The option must be enclosed in parentheses and specified as a *name=value* pair. The value can be a style reference in the form *style-element:style-attribute*.

**COLOR=** *style-reference | color*

specifies the fill *color*. If you use a style reference, then the style attribute should be a valid attribute such as COLOR, CONTRASTCOLOR, STARTCOLOR, NEUTRAL, ENDCOLOR. The convention is to use the COLOR attribute for fill areas.

If you use a color, then SAS accepts color names, such as RED, or color codes, such as CXFF0000 or #FF0000. Color names must not exceed 64 characters. Color codes must not exceed 8 characters and must be in a valid SAS color-naming scheme, such as RGB, CMYK, HLS, or HSV (HSB).

**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.

<table>
<thead>
<tr>
<th>Default</th>
<th>The same as the setting of the statement’s DATATRANSPARENCY= option.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>0–1, where 0 is opaque and 1 is entirely transparent</td>
</tr>
<tr>
<td>Interaction</td>
<td>This setting overrides the statement’s DATATRANSPARENCY= setting for the fills but not for the outlines.</td>
</tr>
<tr>
<td>Example</td>
<td>fillattrs=(transparency=0.5)</td>
</tr>
</tbody>
</table>
**Line Options**

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

**COLOR=** style-reference | color

specifies the line color. If you use a style reference, then the style attribute should be a valid attribute such as COLOR, CONTRASTCOLOR, STARTCOLOR, NEUTRAL, ENDCOLOR. The convention is to use CONTRASTCOLOR for lines. If you specify a style element that does not have a CONTRASTCOLOR attribute, then the element’s COLOR attribute is used.

If you use a color, then SAS accepts color names, such as RED, or color codes, such as CXFF0000 or #FF0000. Color names must not exceed 64 characters. Color codes must not exceed 8 characters and must be in a valid SAS color-naming scheme, such as RGB, CMYK, HLS, or HSV (HSB).

**PATTERN=** style-reference | line-pattern-name | line-pattern-number

specifies the line pattern. If you use a style reference, then the style attribute should be LINESTYLE.

Line patterns can be specified as a pattern name or pattern number.

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Pattern Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solid</td>
<td>1</td>
</tr>
<tr>
<td>ShortDash</td>
<td>2</td>
</tr>
<tr>
<td>MediumDash</td>
<td>4</td>
</tr>
<tr>
<td>LongDash</td>
<td>5</td>
</tr>
<tr>
<td>MediumDashShortDash</td>
<td>8</td>
</tr>
<tr>
<td>DashDashDot</td>
<td>14</td>
</tr>
<tr>
<td>DashDotDot</td>
<td>15</td>
</tr>
<tr>
<td>Dash</td>
<td>20</td>
</tr>
<tr>
<td>LongDashShortDash</td>
<td>26</td>
</tr>
<tr>
<td>Dot</td>
<td>34</td>
</tr>
<tr>
<td>ThinDot</td>
<td>35</td>
</tr>
<tr>
<td>ShortDashDot</td>
<td>41</td>
</tr>
<tr>
<td>MediumDashDotDot</td>
<td>42</td>
</tr>
</tbody>
</table>

Valid pattern numbers range from 1 to 46. Not all pattern numbers have names. See “Available Line Patterns” on page 612 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 following command to disable anti-aliasing in order to show the line detail:

```ods graphics / antialias=off;```

**THICKNESS=** style-reference | dimension

specifies the line thickness. If you use a style reference, then the style attribute should be LINETHICKNESS.
Marker Options

When you specify the attributes for a data marker, the available marker options can be any one or more of the following settings. You must enclose the options in parentheses, and you specify each option as a `name=value` pair. In all cases, the value can be a style reference in the form `style-element:style-attribute`.

**COLOR=** style-reference | color
specifies the color of the marker. If you use a style reference, then the style attribute should be a valid attribute such as COLOR, CONTRASTCOLOR, STARTCOLOR, NEUTRAL, or ENDCOLOR. The convention is to use CONTRASTCOLOR for markers. For grouped data, this option keeps all markers the same color and the marker symbol alone distinguishes the group values.

If you use a color, then SAS accepts color names, such as RED, or color codes, such as CXFF0000 or #FF0000. Color names must not exceed 64 characters. Color codes must not exceed 8 characters and must be in a valid SAS color-naming scheme, such as RGB, CMYK, HLS, or HSV (HSB).

**Restriction**
This option has no effect on marker symbols that are created with the SYMBOLIMAGE statement.

See “SYMBOLIMAGE Statement” in *SAS Graph Template Language: Reference*

**SIZE=** style-reference | dimension
specifies the marker size (both width and height). If you use a style reference, then the style attribute should be MARKERSIZE.

See “dimension” in *SAS Graph Template Language: Reference*

**SYMBOL=** style-reference | marker-name
specifies the name of the marker. If you use a style reference, then the style attribute should be MARKERSYMBOL. The following SAS symbols are supported:

- ↓ ArrowDown
- ★ Asterisk
- ○ Circle
- ♦ Diamond
- > GreaterThan
- < LessThan
- # Hash
- ◆ HomeDown
- + Plus
- ☐ Square
- □ Tilde
- ❀ Star
- ☆ X
- ❁ Y
- ☊ Triangle
- ☉ Circle
- ☛ Diamond
- → TriangleLeft
- ▶ TriangleRight
- ◇ SquareFilled
- ★ StarFilled
- ▲ TriangleFilled
- ▼ TriangleDownFilled
- ◄ TriangleUpFilled
- ◄ TriangleLeftFilled
- ◄ TriangleRightFilled
- ◄ DiamondFilled

You can also specify the names of symbols that are created by the SYMBOLCHAR and SYMBOLIMAGE statements.

See “SYMBOLCHAR Statement” in *SAS Graph Template Language: Reference*

“SYMBOLIMAGE Statement” 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 Statement” in *SAS Graph Template Language: Reference*

“SYMBOLIMAGE Statement” in *SAS Graph Template Language: Reference*

---

**Text Options**

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

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

If you use a color, then SAS accepts color names, such as RED, or color codes, such as CXFF0000 or #FF0000. Color names must not exceed 64 characters. Color codes must not exceed 8 characters and must be in a valid SAS color-naming scheme, such as RGB, CMYK, HLS, or HSV (HSB).

**FAMILY=**style-reference | "string"
specifies the font family of the text. If you use a style reference, then the style attribute should be FONTFAMILY.

**SIZE=**style-reference | dimension
specifies the font size of the text. If you use a style reference, then the style attribute should be 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 instead.

**See**
“dimension” in *SAS Graph Template Language: Reference*

**STYLE=**style-reference | NORMAL | ITALIC
specifies the font style of the text. If you use a style reference, then the style attribute should be FONTSTYLE.
WEIGHT=style-reference | NORMAL | BOLD
specifies the font weight of the text. If you use a style reference, then the style
attribute should be 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.
Appendix 4
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 620.

Table A4.1  SAS Registry Colors by Name

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Color Sample</th>
<th>CX Color Code (RGB)</th>
<th>HLS Color Code*</th>
</tr>
</thead>
<tbody>
<tr>
<td>AliceBlue</td>
<td></td>
<td>CXF0F8FF</td>
<td>H148F8FF</td>
</tr>
<tr>
<td>AntiqueWhite</td>
<td></td>
<td>CXFAEBD7</td>
<td>H09AE9C6</td>
</tr>
<tr>
<td>Aqua</td>
<td></td>
<td>CX00FFFF</td>
<td>H12C80FF</td>
</tr>
<tr>
<td>Aquamarine</td>
<td></td>
<td>CX7FFDD4</td>
<td>H118BEF7</td>
</tr>
<tr>
<td>Azure</td>
<td></td>
<td>CXF0FFFF</td>
<td>H12CF8FF</td>
</tr>
<tr>
<td>Beige</td>
<td></td>
<td>CXF5F5DC</td>
<td>H0B4E98E</td>
</tr>
<tr>
<td>Bisque</td>
<td></td>
<td>CXFFE4C4</td>
<td>H098E2FF</td>
</tr>
<tr>
<td>Black</td>
<td></td>
<td>CX000000</td>
<td>H0000000</td>
</tr>
<tr>
<td>BlanchedAlmond</td>
<td></td>
<td>CXFFEBCD</td>
<td>H09CE6FF</td>
</tr>
<tr>
<td>Blue</td>
<td></td>
<td>CX0000FF</td>
<td>H00080FF</td>
</tr>
<tr>
<td>BlueViolet</td>
<td></td>
<td>CX8A2BE2</td>
<td>H01F87C2</td>
</tr>
<tr>
<td>Brown</td>
<td></td>
<td>CXA52A2A</td>
<td>H0786898</td>
</tr>
<tr>
<td>Burlywood</td>
<td></td>
<td>CXDEB887</td>
<td>H099B391</td>
</tr>
<tr>
<td>Color Name</td>
<td>Color Sample</td>
<td>CX Color Code (RGB)</td>
<td>HLS Color Code*</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td>---------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>CadetBlue</td>
<td></td>
<td>CX5F9EA0</td>
<td>H12D8041</td>
</tr>
<tr>
<td>Chartreuse</td>
<td></td>
<td>CX7FFF00</td>
<td>H0D280FF</td>
</tr>
<tr>
<td>Chocolate</td>
<td></td>
<td>CXD2691E</td>
<td>H09178BF</td>
</tr>
<tr>
<td>Coral</td>
<td></td>
<td>CXFF7F50</td>
<td>H088A8FF</td>
</tr>
<tr>
<td>CornflowerBlue</td>
<td></td>
<td>CX6495ED</td>
<td>H152A9CA</td>
</tr>
<tr>
<td>Cornsilk</td>
<td></td>
<td>CXFFF8DC</td>
<td>H0A7EEFF</td>
</tr>
<tr>
<td>Crimson</td>
<td></td>
<td>CXDC143C</td>
<td>H06C78D5</td>
</tr>
<tr>
<td>Cyan</td>
<td></td>
<td>CX00FFFF</td>
<td>H12C80FF</td>
</tr>
<tr>
<td>DarkBlue</td>
<td></td>
<td>CX00008B</td>
<td>H00046FF</td>
</tr>
<tr>
<td>DarkCyan</td>
<td></td>
<td>CX008B8B</td>
<td>H12C46FF</td>
</tr>
<tr>
<td>DarkGoldenrod</td>
<td></td>
<td>CXB8860B</td>
<td>H0A262E2</td>
</tr>
<tr>
<td>DarkGray</td>
<td></td>
<td>CXA9A9A9</td>
<td>H000A900</td>
</tr>
<tr>
<td>DarkGreen</td>
<td></td>
<td>CX006400</td>
<td>H0F032FF</td>
</tr>
<tr>
<td>DarkKhaki</td>
<td></td>
<td>CXBDB76B</td>
<td>H0AF9462</td>
</tr>
<tr>
<td>DarkMagenta</td>
<td></td>
<td>CX8B008B</td>
<td>H03C46FF</td>
</tr>
<tr>
<td>DarkOliveGreen</td>
<td></td>
<td>CX556B2F</td>
<td>H0CA4D63</td>
</tr>
<tr>
<td>DarkOrange</td>
<td></td>
<td>CXFF8C00</td>
<td>H09880FF</td>
</tr>
<tr>
<td>DarkOrchid</td>
<td></td>
<td>CX9932CC</td>
<td>H0287F9B</td>
</tr>
<tr>
<td>DarkRed</td>
<td></td>
<td>CX8B0000</td>
<td>H07846FF</td>
</tr>
<tr>
<td>DarkSalmon</td>
<td></td>
<td>CXE9967A</td>
<td>H087B2B7</td>
</tr>
<tr>
<td>DarkSeaGreen</td>
<td></td>
<td>CX8FBC8F</td>
<td>H0F0A640</td>
</tr>
<tr>
<td>Color Name</td>
<td>Color Sample</td>
<td>CX Color Code (RGB)</td>
<td>HLS Color Code</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>DarkSlateBlue</td>
<td></td>
<td>CX483D8B</td>
<td>H0086463</td>
</tr>
<tr>
<td>DarkSlateGray</td>
<td></td>
<td>CX2F4F4F</td>
<td>H12C3F41</td>
</tr>
<tr>
<td>DarkSlateGrey</td>
<td></td>
<td>CX2F2F2F</td>
<td>H0002F00</td>
</tr>
<tr>
<td>DarkTurquoise</td>
<td></td>
<td>CX00CED1</td>
<td>H12C69FF</td>
</tr>
<tr>
<td>DarkViolet</td>
<td></td>
<td>CX9400D3</td>
<td>H02A6AFF</td>
</tr>
<tr>
<td>DeepPink</td>
<td></td>
<td>CXFF1493</td>
<td>H0578AFF</td>
</tr>
<tr>
<td>DeepSkyBlue</td>
<td></td>
<td>CX00BFFF</td>
<td>H13B80FF</td>
</tr>
<tr>
<td>DimGray</td>
<td></td>
<td>CX696969</td>
<td>H0006900</td>
</tr>
<tr>
<td>DodgerBlue</td>
<td></td>
<td>CX1E90FF</td>
<td>H1498FFF</td>
</tr>
<tr>
<td>Firebrick</td>
<td></td>
<td>CXB22222</td>
<td>H0786AAD</td>
</tr>
<tr>
<td>FloralWhite</td>
<td></td>
<td>CXFFFFAF0</td>
<td>H0A0F8FF</td>
</tr>
<tr>
<td>ForestGreen</td>
<td></td>
<td>CX228B22</td>
<td>H0F0579B</td>
</tr>
<tr>
<td>Fuchsia</td>
<td></td>
<td>CXFF00FF</td>
<td>H03C80FF</td>
</tr>
<tr>
<td>Gainsboro</td>
<td></td>
<td>CXDADCDC</td>
<td>H000DC00</td>
</tr>
<tr>
<td>GhostWhite</td>
<td></td>
<td>CXF8F8FF</td>
<td>H000FCFF</td>
</tr>
<tr>
<td>Gold</td>
<td></td>
<td>CXFFD700</td>
<td>H0AA80FF</td>
</tr>
<tr>
<td>Goldenrod</td>
<td></td>
<td>CXDA520</td>
<td>H0A27DBE</td>
</tr>
<tr>
<td>Gray or Grey</td>
<td></td>
<td>CX808080</td>
<td>H0008000</td>
</tr>
<tr>
<td>Green</td>
<td></td>
<td>CX008000</td>
<td>H0F040FF</td>
</tr>
<tr>
<td>GreenYellow</td>
<td></td>
<td>CXADFF2F</td>
<td>H0CB97FF</td>
</tr>
<tr>
<td>Honeydew</td>
<td></td>
<td>CXF0FFF0</td>
<td>H0F0F8FF</td>
</tr>
<tr>
<td>Color Name</td>
<td>Color Sample</td>
<td>CX Color Code (RGB)</td>
<td>HLS Color Code*</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
<td>---------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>HotPink</td>
<td></td>
<td>CXFF69B4</td>
<td>H05AB4FF</td>
</tr>
<tr>
<td>IndianRed</td>
<td></td>
<td>CXCD5C5C</td>
<td>H0789587</td>
</tr>
<tr>
<td>Indigo</td>
<td></td>
<td>CX4B0082</td>
<td>H02241FF</td>
</tr>
<tr>
<td>Ivory</td>
<td></td>
<td>CXFFFFFF0</td>
<td>H0B4F8FF</td>
</tr>
<tr>
<td>Khaki</td>
<td></td>
<td>CXF0E68C</td>
<td>H0AE8EC4</td>
</tr>
<tr>
<td>Lavender</td>
<td></td>
<td>CXE6E6FA</td>
<td>H000F0AA</td>
</tr>
<tr>
<td>LavenderBlush</td>
<td></td>
<td>CXFFFF05</td>
<td>H064F8FF</td>
</tr>
<tr>
<td>LawnGreen</td>
<td></td>
<td>CX7FC00</td>
<td>H0D27EFF</td>
</tr>
<tr>
<td>LemonChiffon</td>
<td></td>
<td>CXFFACD</td>
<td>H0AEE6FF</td>
</tr>
<tr>
<td>LightBlue</td>
<td></td>
<td>CXADD8E6</td>
<td>H13ACA88</td>
</tr>
<tr>
<td>LightCoral</td>
<td></td>
<td>CXF08080</td>
<td>H078B8C9</td>
</tr>
<tr>
<td>LightCyan</td>
<td></td>
<td>CXE0FFFF</td>
<td>H12CF0FF</td>
</tr>
<tr>
<td>LightGoldenrodYellow</td>
<td></td>
<td>CXFAFAD2</td>
<td>H0B4E6CC</td>
</tr>
<tr>
<td>LightGray</td>
<td></td>
<td>CXD3D3D3</td>
<td>H000D300</td>
</tr>
<tr>
<td>LightGreen</td>
<td></td>
<td>CX90E990</td>
<td>H0F0BFBB</td>
</tr>
<tr>
<td>LightPink</td>
<td></td>
<td>CXFFB6C1</td>
<td>H06E8FF</td>
</tr>
<tr>
<td>LightSalmon</td>
<td></td>
<td>CXFFA07A</td>
<td>H089BDF9</td>
</tr>
<tr>
<td>LightSeaGreen</td>
<td></td>
<td>CX20B2AA</td>
<td>H12869B1</td>
</tr>
<tr>
<td>LightSkyBlue</td>
<td></td>
<td>CX87CFA</td>
<td>H142C1EB</td>
</tr>
<tr>
<td>LightSlateGray</td>
<td></td>
<td>CX778899</td>
<td>H14A824</td>
</tr>
<tr>
<td>LightSteelBlue</td>
<td></td>
<td>CXB0C4DE</td>
<td>H14DC769</td>
</tr>
<tr>
<td>Color Name</td>
<td>Color Sample</td>
<td>CX Color Code (RGB)</td>
<td>HLS Color Code*</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
<td>---------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>LightYellow</td>
<td></td>
<td>CXFFFFE0</td>
<td>H0BF40FF</td>
</tr>
<tr>
<td>Lime</td>
<td></td>
<td>CX00FF00</td>
<td>H0F80FF</td>
</tr>
<tr>
<td>LimeGreen</td>
<td></td>
<td>CX32CD32</td>
<td>H0F0809B</td>
</tr>
<tr>
<td>Linen</td>
<td></td>
<td>CXFAF0E6</td>
<td>H095F0AA</td>
</tr>
<tr>
<td>Magenta</td>
<td></td>
<td>CXFF00FF</td>
<td>H0C80FF</td>
</tr>
<tr>
<td>Maroon</td>
<td></td>
<td>CX800000</td>
<td>H07840FF</td>
</tr>
<tr>
<td>MediumAquamarine</td>
<td></td>
<td>CX66CDAA</td>
<td>H1179A81</td>
</tr>
<tr>
<td>MediumBlue</td>
<td></td>
<td>CX0000CD</td>
<td>H00067FF</td>
</tr>
<tr>
<td>MediumOrchid</td>
<td></td>
<td>CXBA55D3</td>
<td>H0309496</td>
</tr>
<tr>
<td>MediumPurple</td>
<td></td>
<td>CX9370DB</td>
<td>H013A698</td>
</tr>
<tr>
<td>MediumSeaGreen</td>
<td></td>
<td>CX3CB371</td>
<td>H10A787F</td>
</tr>
<tr>
<td>MediumSlateBlue</td>
<td></td>
<td>CX7B68EE</td>
<td>H008ABCB</td>
</tr>
<tr>
<td>MediumSpringGreen</td>
<td></td>
<td>CX00FA9A</td>
<td>H1147DFF</td>
</tr>
<tr>
<td>MediumTurquoise</td>
<td></td>
<td>CX48D1CC</td>
<td>H1298D99</td>
</tr>
<tr>
<td>MediumVioletRed</td>
<td></td>
<td>CXC71585</td>
<td>H0526ECE</td>
</tr>
<tr>
<td>MidnightBlue</td>
<td></td>
<td>CX191970</td>
<td>H00045A2</td>
</tr>
<tr>
<td>MintCream</td>
<td></td>
<td>CXF5FFFA</td>
<td>H10EFAFF</td>
</tr>
<tr>
<td>MistyRose</td>
<td></td>
<td>CXFFE4E1</td>
<td>H07E0FF</td>
</tr>
<tr>
<td>Moccasin</td>
<td></td>
<td>CXFFE4B5</td>
<td>H09E0AFF</td>
</tr>
<tr>
<td>NavajoWhite</td>
<td></td>
<td>CXFFDEAD</td>
<td>H09BD6FF</td>
</tr>
<tr>
<td>Navy</td>
<td></td>
<td>CX000080</td>
<td>H00040FF</td>
</tr>
<tr>
<td>Color Name</td>
<td>Color Sample</td>
<td>CX Color Code (RGB)</td>
<td>HLS Color Code*</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td>---------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>OldLace</td>
<td></td>
<td>CXFDF5E6</td>
<td>H09FF2D9</td>
</tr>
<tr>
<td>Olive</td>
<td></td>
<td>CX808000</td>
<td>H0B440FF</td>
</tr>
<tr>
<td>OliveDrab</td>
<td></td>
<td>CX6B8E23</td>
<td>H0C7599A</td>
</tr>
<tr>
<td>Orange</td>
<td></td>
<td>CXFFA500</td>
<td>H09E80FF</td>
</tr>
<tr>
<td>OrangeRed</td>
<td></td>
<td>CXFF4500</td>
<td>H08880FF</td>
</tr>
<tr>
<td>Orchid</td>
<td></td>
<td>CXDA70D6</td>
<td>H03EA596</td>
</tr>
<tr>
<td>PaleGoldenrod</td>
<td></td>
<td>CXEEE8AA</td>
<td>H0AECCAA</td>
</tr>
<tr>
<td>PaleGreen</td>
<td></td>
<td>CX98FB98</td>
<td>H0F0CAEC</td>
</tr>
<tr>
<td>PaleTurquoise</td>
<td></td>
<td>CXAFEEEE</td>
<td>H12CCFA6</td>
</tr>
<tr>
<td>PaleVioletRed</td>
<td></td>
<td>CXDB7093</td>
<td>H064A698</td>
</tr>
<tr>
<td>PapayaWhip</td>
<td></td>
<td>CXFFEFD5</td>
<td>H09DEAFF</td>
</tr>
<tr>
<td>Peachpuff</td>
<td></td>
<td>CXFFDAB9</td>
<td>H094DCCF</td>
</tr>
<tr>
<td>Peru</td>
<td></td>
<td>CXCD853F</td>
<td>H0958696</td>
</tr>
<tr>
<td>Pink</td>
<td></td>
<td>CXFFC0CB</td>
<td>H06DE0FF</td>
</tr>
<tr>
<td>Plum</td>
<td></td>
<td>CXDDA0DD</td>
<td>H03CBF79</td>
</tr>
<tr>
<td>PowderBlue</td>
<td></td>
<td>CXB0E0E6</td>
<td>H132CB84</td>
</tr>
<tr>
<td>Purple</td>
<td></td>
<td>CX800080</td>
<td>H03C40FF</td>
</tr>
<tr>
<td>Red</td>
<td></td>
<td>CXFF0000</td>
<td>H07880FF</td>
</tr>
<tr>
<td>RosyBrown</td>
<td></td>
<td>CXBC8F8F</td>
<td>H078A640</td>
</tr>
<tr>
<td>RoyalBlue</td>
<td></td>
<td>CX4169E1</td>
<td>H15991B9</td>
</tr>
<tr>
<td>SaddleBrown</td>
<td></td>
<td>CX8B4513</td>
<td>H0914FC2</td>
</tr>
<tr>
<td>Color Name</td>
<td>Color Sample</td>
<td>CX Color Code (RGB)</td>
<td>HLS Color Code*</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>---------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Salmon</td>
<td></td>
<td>CXFA8072</td>
<td>H07EB6EE</td>
</tr>
<tr>
<td>SandyBrown</td>
<td></td>
<td>CXF4A460</td>
<td>H093AADE</td>
</tr>
<tr>
<td>SeaGreen</td>
<td></td>
<td>CX2E8B57</td>
<td>H10A5D80</td>
</tr>
<tr>
<td>Seashell</td>
<td></td>
<td>CXFFF5EE</td>
<td>H090F7FF</td>
</tr>
<tr>
<td>Sienna</td>
<td></td>
<td>CXA0522D</td>
<td>H08B678F</td>
</tr>
<tr>
<td>Silver</td>
<td></td>
<td>CXC0C0C0</td>
<td>H000C000</td>
</tr>
<tr>
<td>SkyBlue</td>
<td></td>
<td>CX87CEEB</td>
<td>H13DB9B6</td>
</tr>
<tr>
<td>SlateBlue</td>
<td></td>
<td>CX6A5ACD</td>
<td>H0089488</td>
</tr>
<tr>
<td>SlateGray</td>
<td></td>
<td>CX708090</td>
<td>H14A8020</td>
</tr>
<tr>
<td>Snow</td>
<td></td>
<td>CXFFFAFA</td>
<td>H078FDF0</td>
</tr>
<tr>
<td>SpringGreen</td>
<td></td>
<td>CX00FF7F</td>
<td>H10D80FF</td>
</tr>
<tr>
<td>SteelBlue</td>
<td></td>
<td>CX468B4</td>
<td>H1477D70</td>
</tr>
<tr>
<td>Tan</td>
<td></td>
<td>CXD2B48C</td>
<td>H09AA70</td>
</tr>
<tr>
<td>Teal</td>
<td></td>
<td>CX008080</td>
<td>H12C40FF</td>
</tr>
<tr>
<td>Thistle</td>
<td></td>
<td>CXD8BFD8</td>
<td>H03CCC3E</td>
</tr>
<tr>
<td>Tomato</td>
<td></td>
<td>CXFF6347</td>
<td>H081A3FF</td>
</tr>
<tr>
<td>Turquoise</td>
<td></td>
<td>CX40E0D0</td>
<td>H12690B8</td>
</tr>
<tr>
<td>Violet</td>
<td></td>
<td>CXEE82EE</td>
<td>H03CB78C2</td>
</tr>
<tr>
<td>Wheat</td>
<td></td>
<td>CXF5DEB3</td>
<td>H09FD4C4</td>
</tr>
<tr>
<td>White</td>
<td></td>
<td>CXFFFFFF</td>
<td>H000FF00</td>
</tr>
<tr>
<td>WhiteSmoke</td>
<td></td>
<td>CXF5F5F5</td>
<td>H000F500</td>
</tr>
</tbody>
</table>
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 464.

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

<table>
<thead>
<tr>
<th>Color Sample</th>
<th>Color Name</th>
<th>CX Color Code (RGB)</th>
<th>HLS Color Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td></td>
<td>CX000000</td>
<td>H0000000</td>
</tr>
<tr>
<td>DarkSlateGrey</td>
<td></td>
<td>CX2F2F2F</td>
<td>H0002F00</td>
</tr>
<tr>
<td>DimGray</td>
<td></td>
<td>CX696969</td>
<td>H0006900</td>
</tr>
<tr>
<td>Gray or Grey</td>
<td></td>
<td>CX808080</td>
<td>H0008000</td>
</tr>
<tr>
<td>DarkGray</td>
<td></td>
<td>CXA9A9A9</td>
<td>H000A900</td>
</tr>
<tr>
<td>Silver</td>
<td></td>
<td>CXC0C0C0</td>
<td>H000C000</td>
</tr>
<tr>
<td>LightGray</td>
<td></td>
<td>CXD3D3D3</td>
<td>H000D300</td>
</tr>
<tr>
<td>Gainsboro</td>
<td></td>
<td>CXDCDCDC</td>
<td>H000DC00</td>
</tr>
<tr>
<td>WhiteSmoke</td>
<td></td>
<td>CXF5F5F5</td>
<td>H000F500</td>
</tr>
<tr>
<td>White</td>
<td></td>
<td>CXFFFFFF</td>
<td>H000FF00</td>
</tr>
<tr>
<td>MidnightBlue</td>
<td></td>
<td>CX191970</td>
<td>H00045A2</td>
</tr>
<tr>
<td>Lavender</td>
<td></td>
<td>CXE6E6FA</td>
<td>H000F0AA</td>
</tr>
<tr>
<td>Navy</td>
<td></td>
<td>CX000080</td>
<td>H00040FF</td>
</tr>
<tr>
<td>DarkBlue</td>
<td></td>
<td>CX00008B</td>
<td>H00046FF</td>
</tr>
<tr>
<td>MediumBlue</td>
<td></td>
<td>CX0000CD</td>
<td>H00067FF</td>
</tr>
<tr>
<td>Color Sample</td>
<td>Color Name</td>
<td>CX Code (RGB)</td>
<td>HLS Code</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td>🔵</td>
<td>Blue</td>
<td>CX0000FF</td>
<td>H00080FF</td>
</tr>
<tr>
<td>🔷</td>
<td>GhostWhite</td>
<td>CXF8F8FF</td>
<td>H000FCFF</td>
</tr>
<tr>
<td>🔽</td>
<td>SlateBlue</td>
<td>CX6A5ACD</td>
<td>H0089488</td>
</tr>
<tr>
<td>🔴</td>
<td>DarkSlateBlue</td>
<td>CX483D8B</td>
<td>H0086463</td>
</tr>
<tr>
<td>✈</td>
<td>MediumSlateBlue</td>
<td>CX7B68EE</td>
<td>H008ABC8</td>
</tr>
<tr>
<td>💜</td>
<td>MediumPurple</td>
<td>CX9370DB</td>
<td>H013A698</td>
</tr>
<tr>
<td>💜</td>
<td>BlueViolet</td>
<td>CX8A2BE2</td>
<td>H01F87C2</td>
</tr>
<tr>
<td>🔪</td>
<td>Indigo</td>
<td>CX4B0082</td>
<td>H02241FF</td>
</tr>
<tr>
<td>💜</td>
<td>DarkOrchid</td>
<td>CX9932CC</td>
<td>H0287F9B</td>
</tr>
<tr>
<td>💜</td>
<td>DarkViolet</td>
<td>CX9400D3</td>
<td>H02A6AFF</td>
</tr>
<tr>
<td>💜</td>
<td>MediumOrchid</td>
<td>CXBA55D3</td>
<td>H0309496</td>
</tr>
<tr>
<td>💜</td>
<td>Thistle</td>
<td>CXD8BFD8</td>
<td>H03CCC3E</td>
</tr>
<tr>
<td>🌿</td>
<td>Plum</td>
<td>CXDDA0DD</td>
<td>H03CBF79</td>
</tr>
<tr>
<td>🌿</td>
<td>Violet</td>
<td>CXEE82EE</td>
<td>H03CB8C2</td>
</tr>
<tr>
<td>💜</td>
<td>Purple</td>
<td>CX800080</td>
<td>H03C40FF</td>
</tr>
<tr>
<td>💜</td>
<td>DarkMagenta</td>
<td>CX8B008B</td>
<td>H03C46FF</td>
</tr>
<tr>
<td>🌴</td>
<td>Fuchsia</td>
<td>CXFF00FF</td>
<td>H03C80FF</td>
</tr>
<tr>
<td>🌴</td>
<td>Magenta</td>
<td>CXFF00FF</td>
<td>H03C80FF</td>
</tr>
<tr>
<td>🌴</td>
<td>Orchid</td>
<td>CXDA70D6</td>
<td>H03EA596</td>
</tr>
<tr>
<td>🌴</td>
<td>MediumVioletRed</td>
<td>CXC71585</td>
<td>H0526ECE</td>
</tr>
<tr>
<td>🌴</td>
<td>DeepPink</td>
<td>CXFF1493</td>
<td>H0578AFF</td>
</tr>
<tr>
<td>Color Sample</td>
<td>Color Name</td>
<td>CX Code (RGB)</td>
<td>HLS Code</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td>★</td>
<td>HotPink</td>
<td>CXFF69B4</td>
<td>H05AB4FF</td>
</tr>
<tr>
<td>★</td>
<td>LavenderBlush</td>
<td>CXFFF0F5</td>
<td>H064F8FF</td>
</tr>
<tr>
<td>★</td>
<td>PaleVioletRed</td>
<td>CXDB7093</td>
<td>H064A698</td>
</tr>
<tr>
<td>★</td>
<td>Crimson</td>
<td>CXDC143C</td>
<td>H06C78D5</td>
</tr>
<tr>
<td>★</td>
<td>Pink</td>
<td>CXFFC0CB</td>
<td>H06DE0FF</td>
</tr>
<tr>
<td>★</td>
<td>LightPink</td>
<td>CXFFB6C1</td>
<td>H06EDBFF</td>
</tr>
<tr>
<td>★</td>
<td>RosyBrown</td>
<td>CXBC8F8F</td>
<td>H078A640</td>
</tr>
<tr>
<td>★</td>
<td>IndianRed</td>
<td>CXCD5C5C</td>
<td>H0789587</td>
</tr>
<tr>
<td>★</td>
<td>Brown</td>
<td>CXA52A2A</td>
<td>H0786898</td>
</tr>
<tr>
<td>★</td>
<td>Firebrick</td>
<td>CXB22222</td>
<td>H0786AAD</td>
</tr>
<tr>
<td>★</td>
<td>LightCoral</td>
<td>CXF08080</td>
<td>H078B8C9</td>
</tr>
<tr>
<td>★</td>
<td>Maroon</td>
<td>CX800000</td>
<td>H07840FF</td>
</tr>
<tr>
<td>★</td>
<td>DarkRed</td>
<td>CX8B0000</td>
<td>H07846FF</td>
</tr>
<tr>
<td>★</td>
<td>Red</td>
<td>CXFF0000</td>
<td>H07880FF</td>
</tr>
<tr>
<td>★</td>
<td>Snow</td>
<td>CXFFFAFA</td>
<td>H078FDFF</td>
</tr>
<tr>
<td>★</td>
<td>MistyRose</td>
<td>CXFFE4E1</td>
<td>H07EF0FF</td>
</tr>
<tr>
<td>★</td>
<td>Salmon</td>
<td>CXFA8072</td>
<td>H07EB6EE</td>
</tr>
<tr>
<td>★</td>
<td>Tomato</td>
<td>CXFF6347</td>
<td>H081A3FF</td>
</tr>
<tr>
<td>★</td>
<td>DarkSalmon</td>
<td>CXE9967A</td>
<td>H087B2B7</td>
</tr>
<tr>
<td>★</td>
<td>Coral</td>
<td>CXFF7F50</td>
<td>H088A8FF</td>
</tr>
<tr>
<td>★</td>
<td>OrangeRed</td>
<td>CXFF4500</td>
<td>H08880FF</td>
</tr>
<tr>
<td>Color Sample</td>
<td>Color Name</td>
<td>CX Code (RGB)</td>
<td>HLS Code</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>LightSalmon</td>
<td>CXFFA07A</td>
<td>H089BDDF</td>
</tr>
<tr>
<td></td>
<td>Sienna</td>
<td>CXA0522D</td>
<td>H08B678F</td>
</tr>
<tr>
<td></td>
<td>Seashell</td>
<td>CXFFF5EE</td>
<td>H090F7FF</td>
</tr>
<tr>
<td></td>
<td>Chocolate</td>
<td>CXD2691E</td>
<td>H09178BF</td>
</tr>
<tr>
<td></td>
<td>SaddleBrown</td>
<td>CX8B4513</td>
<td>H0914FC2</td>
</tr>
<tr>
<td></td>
<td>SandyBrown</td>
<td>CXF4A460</td>
<td>H093AADE</td>
</tr>
<tr>
<td></td>
<td>Peachpuff</td>
<td>CXFFDAB9</td>
<td>H094DFF</td>
</tr>
<tr>
<td></td>
<td>Peru</td>
<td>CXCD853F</td>
<td>H0958696</td>
</tr>
<tr>
<td></td>
<td>Linen</td>
<td>CXFAF0E6</td>
<td>H095F0AA</td>
</tr>
<tr>
<td></td>
<td>Bisque</td>
<td>CXFFE4C4</td>
<td>H098E2FF</td>
</tr>
<tr>
<td></td>
<td>DarkOrange</td>
<td>CXFF8C00</td>
<td>H09880FF</td>
</tr>
<tr>
<td></td>
<td>Burlywood</td>
<td>CXDEB887</td>
<td>H099B391</td>
</tr>
<tr>
<td></td>
<td>Tan</td>
<td>CXD2B48C</td>
<td>H09AA7F0</td>
</tr>
<tr>
<td></td>
<td>AntiqueWhite</td>
<td>CXFAEBD7</td>
<td>H09AE9C6</td>
</tr>
<tr>
<td></td>
<td>NavajoWhite</td>
<td>CXFFDEAD</td>
<td>H09BD6FF</td>
</tr>
<tr>
<td></td>
<td>BlanchedAlmond</td>
<td>CXFFEBCD</td>
<td>H09CE6FF</td>
</tr>
<tr>
<td></td>
<td>Papaya Whip</td>
<td>CXFFEF5</td>
<td>H09DEAFF</td>
</tr>
<tr>
<td></td>
<td>Moccasin</td>
<td>CXFFE4B5</td>
<td>H09EDAFF</td>
</tr>
<tr>
<td></td>
<td>Orange</td>
<td>CXFFA500</td>
<td>H09E80FF</td>
</tr>
<tr>
<td></td>
<td>Wheat</td>
<td>CXF5DEB3</td>
<td>H09FD4C4</td>
</tr>
<tr>
<td></td>
<td>OldLace</td>
<td>CXFDF5E6</td>
<td>H09FF2D9</td>
</tr>
<tr>
<td>Color Name</td>
<td>CX Code (RGB)</td>
<td>HLS Code</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>FloralWhite</td>
<td>CXFFFAF0</td>
<td>H0A0F8FF</td>
<td></td>
</tr>
<tr>
<td>DarkGoldenrod</td>
<td>CXB8860B</td>
<td>H0A262E2</td>
<td></td>
</tr>
<tr>
<td>Goldenrod</td>
<td>CXDAA520</td>
<td>H0A27DBE</td>
<td></td>
</tr>
<tr>
<td>Cornsilk</td>
<td>CXFFF8DC</td>
<td>H0A7EEFF</td>
<td></td>
</tr>
<tr>
<td>Gold</td>
<td>CXFD700</td>
<td>H0AA80FF</td>
<td></td>
</tr>
<tr>
<td>Khaki</td>
<td>CXF0E68C</td>
<td>H0AE8EC4</td>
<td></td>
</tr>
<tr>
<td>LemonChiffon</td>
<td>CXFFACD</td>
<td>H0AEE6FF</td>
<td></td>
</tr>
<tr>
<td>PaleGoldenrod</td>
<td>CXEEE8AA</td>
<td>H0AECCAA</td>
<td></td>
</tr>
<tr>
<td>DarkKhaki</td>
<td>CXBDB76B</td>
<td>H0AF9462</td>
<td></td>
</tr>
<tr>
<td>Beige</td>
<td>CXF5F5DC</td>
<td>H0B4E98E</td>
<td></td>
</tr>
<tr>
<td>LightGoldenrodYellow</td>
<td>CXFAFAD2</td>
<td>H0B4E6CC</td>
<td></td>
</tr>
<tr>
<td>Olive</td>
<td>CX808000</td>
<td>H0B440FF</td>
<td></td>
</tr>
<tr>
<td>Yellow</td>
<td>CXFFFF00</td>
<td>H0B480FF</td>
<td></td>
</tr>
<tr>
<td>LightYellow</td>
<td>CXFFFFE0</td>
<td>H0B4F0FF</td>
<td></td>
</tr>
<tr>
<td>Ivory</td>
<td>CXFFFFF0</td>
<td>H0B4F8FF</td>
<td></td>
</tr>
<tr>
<td>OliveDrab</td>
<td>CX6B8E23</td>
<td>H0C7599A</td>
<td></td>
</tr>
<tr>
<td>YellowGreen</td>
<td>CX9ACD32</td>
<td>H0C7809B</td>
<td></td>
</tr>
<tr>
<td>DarkOliveGreen</td>
<td>CX556B2F</td>
<td>H0CA4D63</td>
<td></td>
</tr>
<tr>
<td>GreenYellow</td>
<td>CXADFF2F</td>
<td>H0CB97FF</td>
<td></td>
</tr>
<tr>
<td>Chartreuse</td>
<td>CX7FF00</td>
<td>H0D280FF</td>
<td></td>
</tr>
<tr>
<td>LawnGreen</td>
<td>CX7FC00</td>
<td>H0D27EFF</td>
<td></td>
</tr>
<tr>
<td>Color Sample</td>
<td>Color Name</td>
<td>CX Code (RGB)</td>
<td>HLS Code</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>DarkSeaGreen</td>
<td>CX8FBC8F</td>
<td>H0F0A640</td>
</tr>
<tr>
<td></td>
<td>ForestGreen</td>
<td>CX228B22</td>
<td>H0F0579B</td>
</tr>
<tr>
<td></td>
<td>LimeGreen</td>
<td>CX32CD32</td>
<td>H0F0809B</td>
</tr>
<tr>
<td></td>
<td>LightGreen</td>
<td>CX90EE90</td>
<td>H0F0BFBB</td>
</tr>
<tr>
<td></td>
<td>PaleGreen</td>
<td>CX98FB98</td>
<td>H0F0CAEC</td>
</tr>
<tr>
<td></td>
<td>DarkGreen</td>
<td>CX006400</td>
<td>H0F032FF</td>
</tr>
<tr>
<td></td>
<td>Green</td>
<td>CX008000</td>
<td>H0F040FF</td>
</tr>
<tr>
<td></td>
<td>Lime</td>
<td>CX00FF00</td>
<td>H0F080FF</td>
</tr>
<tr>
<td></td>
<td>Honeydew</td>
<td>CXF0FFF0</td>
<td>H0F0F8FF</td>
</tr>
<tr>
<td></td>
<td>SeaGreen</td>
<td>CX2E8B57</td>
<td>H10A5D80</td>
</tr>
<tr>
<td></td>
<td>MediumSeaGreen</td>
<td>CX3CB371</td>
<td>H10A787F</td>
</tr>
<tr>
<td></td>
<td>SpringGreen</td>
<td>CX00FF7F</td>
<td>H10D80FF</td>
</tr>
<tr>
<td></td>
<td>MintCream</td>
<td>CXF5FFFA</td>
<td>H10EFAFF</td>
</tr>
<tr>
<td></td>
<td>MediumSpringGreen</td>
<td>CX00FA9A</td>
<td>H1147DFF</td>
</tr>
<tr>
<td></td>
<td>MediumAquamarine</td>
<td>CX66CDAA</td>
<td>H1179A81</td>
</tr>
<tr>
<td></td>
<td>Aquamarine</td>
<td>CX7FFDD4</td>
<td>H118BEF7</td>
</tr>
<tr>
<td></td>
<td>Turquoise</td>
<td>CX40E0D0</td>
<td>H12690B8</td>
</tr>
<tr>
<td></td>
<td>LightSeaGreen</td>
<td>CX20B2AA</td>
<td>H12869B1</td>
</tr>
<tr>
<td></td>
<td>MediumTurquoise</td>
<td>CX48D1CC</td>
<td>H1298D99</td>
</tr>
<tr>
<td></td>
<td>DarkSlateGray</td>
<td>CX2F4F4F</td>
<td>H12C3F41</td>
</tr>
<tr>
<td></td>
<td>PaleTurquoise</td>
<td>CXAFEEEE</td>
<td>H12CCFA6</td>
</tr>
</tbody>
</table>
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 464.

<table>
<thead>
<tr>
<th>Color Sample</th>
<th>Color Name</th>
<th>CX Code (RGB)</th>
<th>HLS Code*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teal</td>
<td>CX008080</td>
<td>H12C40FF</td>
<td></td>
</tr>
<tr>
<td>DarkCyan</td>
<td>CX008B8B</td>
<td>H12C46FF</td>
<td></td>
</tr>
<tr>
<td>Aqua</td>
<td>CX00FFFF</td>
<td>H12C80FF</td>
<td></td>
</tr>
<tr>
<td>Cyan</td>
<td>CX00FFFF</td>
<td>H12C80FF</td>
<td></td>
</tr>
<tr>
<td>LightCyan</td>
<td>CXE0FFFF</td>
<td>H12CF0FF</td>
<td></td>
</tr>
<tr>
<td>Azure</td>
<td>CXF0FFFF</td>
<td>H12CF8FF</td>
<td></td>
</tr>
<tr>
<td>DarkTurquoise</td>
<td>CX00CED1</td>
<td>H12C69FF</td>
<td></td>
</tr>
<tr>
<td>CadetBlue</td>
<td>CX5F9EA0</td>
<td>H12D8041</td>
<td></td>
</tr>
<tr>
<td>PowderBlue</td>
<td>CXB0E0E6</td>
<td>H132CB84</td>
<td></td>
</tr>
<tr>
<td>LightBlue</td>
<td>CXADD8E6</td>
<td>H13ACA88</td>
<td></td>
</tr>
<tr>
<td>DeepSkyBlue</td>
<td>CX00BFFF</td>
<td>H13B80FF</td>
<td></td>
</tr>
<tr>
<td>SkyBlue</td>
<td>CX87CEE8</td>
<td>H13DB9B6</td>
<td></td>
</tr>
<tr>
<td>LightSkyBlue</td>
<td>CX87CEFA</td>
<td>H142C1EB</td>
<td></td>
</tr>
<tr>
<td>SteelBlue</td>
<td>CX4682B4</td>
<td>H1477D70</td>
<td></td>
</tr>
<tr>
<td>AliceBlue</td>
<td>CXF0F8FF</td>
<td>H148F8FF</td>
<td></td>
</tr>
<tr>
<td>DodgerBlue</td>
<td>CX1E90FF</td>
<td>H1498FFF</td>
<td></td>
</tr>
<tr>
<td>SlateGray</td>
<td>CX708090</td>
<td>H14A8020</td>
<td></td>
</tr>
<tr>
<td>LightSlateGray</td>
<td>CX778899</td>
<td>H14A8824</td>
<td></td>
</tr>
<tr>
<td>LightSteelBlue</td>
<td>CXB0C4DE</td>
<td>H14DC769</td>
<td></td>
</tr>
<tr>
<td>CornflowerBlue</td>
<td>CX6495ED</td>
<td>H152A9CA</td>
<td></td>
</tr>
<tr>
<td>RoyalBlue</td>
<td>CX4169E1</td>
<td>H15991B9</td>
<td></td>
</tr>
</tbody>
</table>
Table A5.1 on page 628 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>
<thead>
<tr>
<th>Fit Policy</th>
<th>LAYOUT OVERLAY</th>
<th>LAYOUT LATTICE</th>
<th>LAYOUT DATALATTICE</th>
<th>LAYOUT DATAPANEL</th>
<th>LAYOUT EQUATED</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>V</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td>No adjustment is attempted.</td>
</tr>
<tr>
<td>ROTATE</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>The tick values are rotated 45 degrees when a collision occurs.</td>
</tr>
<tr>
<td>ROTATEALWAYS</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>The tick values are always rotated 45 degrees regardless of whether a collision occurs.</td>
</tr>
<tr>
<td>ROTATEALWAYSDROP</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The tick values are always rotated 45 degrees regardless of whether a collision occurs. If unsuccessful, the values are dropped.</td>
</tr>
<tr>
<td>ROTATETHIN</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>The ROTATE policy is attempted first. If unsuccessful, the THIN policy is applied.</td>
</tr>
<tr>
<td>SPLIT</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The tick values on a discrete axis are split into multiple lines on blank spaces by default when a collision occurs.</td>
</tr>
<tr>
<td>SPLITALWAYS</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>SPLITALWAYSTHIN</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The SPLITALWAYS policy is attempted first. If unsuccessful, the THIN policy is applied.</td>
</tr>
<tr>
<td>SPLITROTATE</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The SPLIT policy is attempted first. If unsuccessful, the ROTATE policy is applied.</td>
</tr>
<tr>
<td>Fit Policy</td>
<td>LAYOUT OVERLAY</td>
<td>LAYOUT LATTICE</td>
<td>LAYOUT DATALATTICE</td>
<td>LAYOUT DATAPANEL</td>
<td>LAYOUT EQUATED</td>
<td>Notes</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>----------------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>SPLITTHIN</td>
<td>V</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td>The SPLIT policy is attempted first. If unsuccessful, the THIN policy is applied.</td>
</tr>
<tr>
<td>STAGGER</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>The tick values alternate between two rows.</td>
</tr>
<tr>
<td>STAGGERROTATE</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>The STAGGER policy is attempted first. If unsuccessful, the ROTATE policy is applied.</td>
</tr>
<tr>
<td>STAGGERTHIN</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>The STAGGER policy is attempted first. If unsuccessful, the THIN policy is applied.</td>
</tr>
<tr>
<td>STAGGERTRUNCATE</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td>The STAGGER policy is attempted first. If unsuccessful, the TRUNCATE policy is applied.</td>
</tr>
<tr>
<td>TRUNCATE</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td>The tick values are shortened.</td>
</tr>
<tr>
<td>TRUNCATEROTATE</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td>The TRUNCATE policy is attempted first. If unsuccessful, the ROTATE policy is applied.</td>
</tr>
<tr>
<td>TRUNCATESTAGGER</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td>The TRUNCATE policy is attempted first. If unsuccessful, the STAGGER policy is applied.</td>
</tr>
<tr>
<td>TRUNCATEETHIN</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td>The TRUNCATE policy is attempted first. If unsuccessful, the THIN policy is applied.</td>
</tr>
<tr>
<td>THIN</td>
<td>B</td>
<td>B</td>
<td>H</td>
<td>B</td>
<td>B</td>
<td>Some tick values are removed.</td>
</tr>
<tr>
<td>EXTRACT</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The tick values are extracted to an axis legend only if the values cannot be fit on the axis.</td>
</tr>
<tr>
<td>Fit Policy</td>
<td>LAYOUT OVERLAY</td>
<td>LAYOUT LATTICE LAYOUT DATALATTICE LAYOUT DATAPANEL</td>
<td>LAYOUT EQUATED</td>
<td>Notes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
<td>---------------------------------------------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXTRACTALWAYS</td>
<td>B</td>
<td>B</td>
<td></td>
<td>The tick values are always extracted to an axis legend even if the values can be fit on the axis.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix 6
SAS Formats Not Supported

Using SAS Formats
SAS formats can be assigned to input data columns with the FORMAT statement of the SGRENDER procedure. Also, several GTL statement options enable a SAS format as an option value. Examples include the TICKVALUEFORMAT= option for formatting axis tick values, and the TIPFORMAT= option for formatting data tips.

Not all SAS formats are supported in the GTL or with the SGPLOT, SGSCATTER, SGPANEL, and SGRENDER procedures. The tables in the following sections show the character and numeric SAS formats that are not supported.

When the GTL encounters an unsupported format, a note similar to the following is written to the SAS log:

```
TICKVALUEFORMAT=bestx. is invalid. The format is invalid or unsupported. The default will be used.
```

Unsupported Numeric Formats
The following numeric formats are not supported in the GTL:

<table>
<thead>
<tr>
<th>Format</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>BESTD</td>
<td>IB</td>
</tr>
<tr>
<td>BESTX</td>
<td>IBR</td>
</tr>
<tr>
<td>D</td>
<td>IEEE</td>
</tr>
<tr>
<td>FLOAT</td>
<td>IEEER</td>
</tr>
<tr>
<td>FRAC</td>
<td></td>
</tr>
</tbody>
</table>
Unsupported Date and Time Formats Related to ISO 8601

The following date and time formats are not supported in the GTL:

<table>
<thead>
<tr>
<th>Format</th>
<th>Format</th>
<th>Format</th>
<th>Format</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N8601B</td>
<td>$N8601BA</td>
<td>$N8601E</td>
<td>$N8601EA</td>
<td>$N8601EH</td>
</tr>
<tr>
<td>$N8601EX</td>
<td>$N8601H</td>
<td>$N8601X</td>
<td>B8601DA</td>
<td>B8601DN</td>
</tr>
<tr>
<td>B8601DT</td>
<td>B8601DZ</td>
<td>B8601LZ</td>
<td>B8601TM</td>
<td>B8601TZ</td>
</tr>
<tr>
<td>E8601DA</td>
<td>E8601DN</td>
<td>E8601DT</td>
<td>E8601DZ</td>
<td>E8601LZ</td>
</tr>
<tr>
<td>E8601TM</td>
<td>E8601TZ</td>
<td>IS8601DA</td>
<td>IS8601DN</td>
<td>IS8601DT</td>
</tr>
<tr>
<td>IS8601DZ</td>
<td>IS8601LZ</td>
<td>IS8601TM</td>
<td>IS8601TZ</td>
<td></td>
</tr>
</tbody>
</table>

Other Unsupported Date and Time Formats

The following date and time formats are not supported in the GTL:

<table>
<thead>
<tr>
<th>Format</th>
<th>Format</th>
<th>Format</th>
<th>Format</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDATE</td>
<td>HEBDATE</td>
<td>JDASTEMDW</td>
<td>JDATEMNW</td>
<td>JDATEWK</td>
</tr>
<tr>
<td>JDATEYDW</td>
<td>JDATEYM</td>
<td>JDATEYMD</td>
<td>JDATEYMW</td>
<td>JDATEYT</td>
</tr>
<tr>
<td>JDATEYTW</td>
<td>JNENGO</td>
<td>JNENGOT</td>
<td>JNENGOTW</td>
<td>JNENGOW</td>
</tr>
<tr>
<td>JTIMEH</td>
<td>JTIMEHM</td>
<td>JTIMEHMS</td>
<td>JTIMEHW</td>
<td>JTIMEMW</td>
</tr>
<tr>
<td>JTIMESW</td>
<td>MDYAMPM</td>
<td>MINGUO</td>
<td>NENGO</td>
<td>NLDATEYQ</td>
</tr>
<tr>
<td>NLDATEYR</td>
<td>NLDATEYW</td>
<td>NLDATMYQ</td>
<td>NLDATMYR</td>
<td>NLDATMYW</td>
</tr>
<tr>
<td>NLSTRMON</td>
<td>NLSTRQTR</td>
<td>NLSTRWK</td>
<td>PDJULG</td>
<td>PDJULI</td>
</tr>
<tr>
<td>TWMDY</td>
<td>XYYMMDD</td>
<td>YYQZ</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Unsupported Currency Formats

The following currency formats are not supported in the GTL:

<table>
<thead>
<tr>
<th>EURFRATS</th>
<th>EURFRBEF</th>
<th>EURFRCHF</th>
<th>EURFRCZK</th>
<th>EURFRDEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EURFRDKK</td>
<td>EURFRESP</td>
<td>EURFRFIM</td>
<td>EURFRFRF</td>
<td>EURFRGBP</td>
</tr>
<tr>
<td>EURFRGRD</td>
<td>EURFRHU</td>
<td>EURFRIEP</td>
<td>EURFRITL</td>
<td>EURFRLUF</td>
</tr>
<tr>
<td>EURFRLG</td>
<td>EURFRNO</td>
<td>EURFRPLZ</td>
<td>EURFRPTE</td>
<td>EURFRROL</td>
</tr>
<tr>
<td>EURFRRUR</td>
<td>EURFRSEK</td>
<td>EURFRSIT</td>
<td>EURFRTRL</td>
<td>EURFRYUD</td>
</tr>
<tr>
<td>EURTOATS</td>
<td>EURTOBEF</td>
<td>EURTOCHF</td>
<td>EURTOCZK</td>
<td>EURTODEM</td>
</tr>
<tr>
<td>EURTODKK</td>
<td>EURTOESP</td>
<td>EURTOFIM</td>
<td>EURTOFRF</td>
<td>EURTOGBP</td>
</tr>
<tr>
<td>EURTOGRD</td>
<td>EURTOHU</td>
<td>EURTOIEP</td>
<td>EURTOITL</td>
<td>EURTOLUF</td>
</tr>
<tr>
<td>EURTONLG</td>
<td>EURTONOK</td>
<td>EURTOPLZ</td>
<td>EURTOPTL</td>
<td>EURTOROL</td>
</tr>
<tr>
<td>EURTORUR</td>
<td>EURTOSEK</td>
<td>EURTOSIT</td>
<td>EURTOTRL</td>
<td>EURTOYUD</td>
</tr>
</tbody>
</table>

Unsupported User-Defined Formats

In the second maintenance release of SAS 9.4 and in earlier releases, ODS Graphics does not support Unicode values in user-defined formats. Starting with the third maintenance release of SAS 9.4, 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 122.
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:

```sas
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:

```sas
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:
    ```sas
    -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:
    ```sas
    -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:

```sas
-jreoptions (-Xmx256m -Xms256m)
```

Alternatively, you might need to specify the setting in quotation marks:

```sas
-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

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.

<table>
<thead>
<tr>
<th>Hexadecimal Value</th>
<th>Decimal Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–9</td>
<td>0–9</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
</tr>
<tr>
<td>E</td>
<td>14</td>
</tr>
<tr>
<td>F</td>
<td>15</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Divide by 16</th>
<th>Quotient</th>
<th>Remainder</th>
<th>16 x Remainder</th>
<th>Hexadecimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>51364 / 16</td>
<td>3210</td>
<td>0.25</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Divide by 16 | Quotient | Remainder | 16 x | Hexadecimal Value
---|---|---|---|---
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.

```sas
%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.

```sas
%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.

```sas
%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.

```sas
%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 your 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 8 • Understanding Hexadecimal Values
Appendix 9

ODS Graphics and SAS/GRAPH

SAS produces graphics using two very distinct systems: ODS Graphics and SAS/GRAPH. ODS Graphics and the 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:

- The 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. The 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.

- The 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. The 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.

- The 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 the GTL. The 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/GROUP 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/GROUP also uses ODS styles by default. However, with SAS/GROUP, 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 448.

• The 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/GROUP also supports all of the ODS destinations. However, for the LISTING destination, SAS/GROUP creates a GRSEG node in the Results tree, and the image appears in the graph window automatically.

• The 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/GROUP does not support scaling of fonts and markers.

• The GTL does not support the SAS/GROUP 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 19, “Adding Data-Driven Annotations to Your Graph,” on page 405. For information about using the GTL draw statements, see Chapter 18, “Adding Code-Driven Graphics Elements to Your Graph,” on page 393. You can also use the ODS Graphics Editor to add annotations to your graphs. See SAS ODS Graphics Editor: User’s Guide.

• The GTL does not support RUN-group processing. SAS/GROUP supports RUN-group processing for some procedures.
Recommended Reading

Here is the recommended reading list for this title:

• *Getting Started with the Graph Template Language in SAS: Examples, Tips, and Techniques for Creating Custom Graphs*
• *Output Delivery System: The Basics and Beyond*
• *PROC TEMPLATE Made Easy: A Guide for SAS(R) Users*
• *Statistical Graphics in SAS: An Introduction to the Graph Template Language and the Statistical Graphics Procedures*
• *Statistical Programming in SAS*
• *The Little SAS Book: A Primer*

For a complete list of SAS publications, go to [sas.com/store/books](http://sas.com/store/books). If you have questions about which titles you need, please contact a SAS Representative:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: [sas.com/store/books](http://sas.com/store/books)
anti-aliasing
a rendering technique for improving the appearance of text and curved lines in a graph by blurring the jagged edges normally present. The degree of improvement is relative to the nature of the graphical content (for example, vertical and horizontal lines do not benefit from anti-aliasing). Extra processing is required to perform anti-aliasing.

attribute bundle
a common collection of visual properties associated with a graphical primitive such as a line, marker, or text. For example, all lines have visual properties of pattern, thickness, and color. All markers have visual properties of symbol, size, weight, and color. Attribute bundles can be associated with style elements in order to indirectly assign visual properties.

axis
a line that represents the midpoints (for a discrete axis) or the scale (for a continuous or interval axis) for graphing variable or data values. An axis typically consists of an axis line with tick marks, tick values (or midpoint values), and a label.

axis offset
the gaps that normally appear at the ends of an axis line. The gaps enable markers, bars, and other graphic primitives that are drawn at extreme data values to be rendered without clipping. An offset can also be used to add extra space between an axis line and visual elements in the graph.

axis threshold
a numerical bias from 0 to 1 that determines whether an extra tick is added at either end of a non-discrete, interval axis. If the minimum and maximum thresholds are set to 0, then no ticks are added beyond the actual data range. If both minimum and maximum thresholds are set to 1, then the data range is completely bounded by the first and last ticks.

axis tick mark
a short line segment perpendicular to the axis line. A tick can cross the axis line, or be drawn from the axis inside or outside the wall.

axis tick value
a formatted data value represented by a tick mark.
axis type
a keyword that denotes axis functionality. For example, the axis type of interval axes can be LINEAR, TIME, or LOG. The axis type of a discrete axis is DISCRETE.

band plot
a plot that draws a horizontal band with two Y values for each X value, or that draws a vertical band with two X values for each Y value. A band plot is typically used to show confidence, error, prediction, or control limits. The points on the upper and lower band boundaries can be joined to create two outlines, or the area between the boundaries can be filled.

bin
one of multiple numeric intervals into which continuous numeric data can be categorized.

binned data
data that has been summarized or transformed in some way to facilitate its rendering by a parameterized plot. Continuous numeric data is typically binned by setting a bin width (interval size) and then computing the number of bins, or by setting the number of bins and computing the bin width.

block
See statement block.

block plot
a plot that displays one or more rectangles (blocks) along an axis, where each rectangle identifies a block of consecutive observations having the same value for a specified block variable.

category variable
a classification variable with a finite number of distinct (discrete) values. These variables are typically used to split data into subsets. For example, in a bar chart, each unique value is displayed as a bar on a DISCRETE axis. In another example, the variable payment mode can have two values, prepaid and postpaid. Customers can be classified based on this variable as prepaid customers and postpaid customers.

cell
See graph cell.

cell block
a block beginning with a CELL statement and ending with an ENDCELL statement that defines the graphical content of a cell. The cell block is available only within a LATTICE layout.

child block
a block that is contained within another block when two or more blocks are nested. For example, a CELLHEADER block is always a child of a CELL block.

class variable
See classification variable.

classification level
for a single classification variable, each unique value is regarded as a classification level. For two or more variables, a classification level is one of the unique combinations (crossings) of the unique values of each variable. For example, if three variables have four, two, and three distinct values, there are 24 classification levels.
classification panel
a multi-cell graph in which the cell data is driven by the values of one or more
classification variables. The number of the cells is determined by the unique values
of the classification variables. Each cell of the panel has the same types of plots.

classification variable (class variable)
a variable whose values are used to classify the observations in a data set into
different groups that are meaningful for analysis. A classification variable can have
either character or numeric values. Classification variables include group, subgroup,
category, and BY variables.

clip
to truncate a plot or graphical element (such as a line, marker, or band) when it
reaches a boundary such as a plot wall.

column axis
an external axis appearing above or below a column of cells and serving as a
common reference for the column of a multi-cell layout, such as a LATTICE,
DATAPANEL, or DATALATTICE layout.

column gutter
the space between columns of cells in a multi-cell layout.

column header
text that labels the column contents in a multi-cell layout. This text can be aligned
above or below the cells in a column. In a LATTICE layout, the column header is not
restricted to text (it can contain a plot or a legend, for example).

column major order
an order for populating cells of a layout or entries in a legend when the number of
rows is specified. By default, cells or entries are filled starting from the top left and
moving down. When the bottom row of the first column is filled, a new column
begins filling to the right of the previous column, and so on until all content items
have been placed in cells or entries. There might be empty cells or entries in the last
column.

column weight
in a LATTICE layout, the proportion of width allotted to a specific column of the
layout. The sum of all column weights is 1.

computed plot
a plot in which input data is internally summarized or otherwise transformed to
create new data that is actually rendered by the plot. Examples of computed plot
statements are BARCHART, BOXPLOT, HISTOGRAM, ELLIPSE, and
REGRESSIONPLOT.

conditional logic
syntax that enables one set of statements or an optional alternate set of statements to
execute at run time.

continuous legend
a legend that shows a mapping between a color ramp or color segments and
 corresponding numeric values. Plots that support a COLORMODEL= option can use
this type of legend.
crossing
a combination of the unique values of one or more classification variables.

cube
in three-dimensional graphics, the outlines formed by the intersection of three pairs of parallel planes; each pair is orthogonal to the primary X, Y, and Z axes. The display of the cube is optional.

data object
a transient version of a SAS data set created by ODS. When an input SAS data set is bound to a compiled graph template, an ODS data object is created, based on all the columns requested in the template definition and any new columns that have been directly or indirectly computed. A data object can persist when used with the ODS OUTPUT statement.

data tip
data or other detailed information that is displayed when a user positions a mouse pointer over an element in a graph. For example, a data tip typically displays the data value that is represented by a bar, a plot point, or some other element.

define block
in the TEMPLATE procedure, a define block (beginning with a DEFINE statement and ending with an END statement) creates various types of templates, including STATGRAPH, STYLE, and TABLE.

dependent plot
a plot that cannot be rendered by itself. Dependent plots must be overlaid with a stand-alone plot. Dependent plots do not provide data ranges to establish axes. REFERENCELINE, DROPLINE, and LINEPARM statements produce dependent plots.

design size
the intended size of a graph that is specified in the graph template definition. The DESIGNHEIGHT and DESIGNWIDTH options of the BEGINGRAPH statement set the intended height and width, which are used to determine the scale factors when the graph is resized. The intended height and width are used unless overridden by the ODS Graphics statement HEIGHT or WIDTH options when the template is executed.

device-based graphic
a graph created with SAS/GRAPH software for which a user-specified or default device (DEVICE= option) controls certain aspects of the graphical output.

discrete axis
an axis for categorical data values. The distance between ticks has no significance. A bar chart always has a discrete axis.

discrete legend
a legend that provides values or descriptive information about graphical elements in a grouped or overlaid plot.

dots per inch (DPI)
a measure of the graph resolution by its dot density.

DPI
See dots per inch.
drop line
a line drawn from a point in the plot area perpendicular to an axis.

dynamic variable
a variable defined in a template with the DYNAMIC statement that can be initialized at template run time.

equated axes
in two-dimensional plots, axes that use the same drawing scale (ratio of display distance to data interval) on both axes. For example, an interval of 2 on the X axis maps to the same display distance as an interval of 2 on the Y axis. The aspect ratio of the plot display equals the aspect ratio of the plot data. In other words, a 45-degree slope in data will be represented by a 45-degree slope in the display. Equated axes are always of TYPE=LINEAR. The number of intervals displayed on each axis does not have to be the same.

external axis
an axis that is outside all cells of a layout. An external axis represents a common scale for all plots in a row or column of a multi-cell layout.

fill
to apply a color within a bounded area. Many plots, such bar charts and band plots, have bounded areas that can be filled or unfilled. When filled, a color is applied. When unfilled, the areas are transparent.

fit policy
one of several algorithms for avoiding tick-value collision when space allotted to a predefined area does not permit all the text to fit. For example, an axis might have a THIN policy that eliminates the display of tick values for alternate ticks. A ROTATE policy would turn the tick values at a 45-degree angle. A TRUNCATE policy would truncate all long tick values to a fixed length and add an ellipsis ( . . .) at the end to imply truncation. A STAGGER policy would create two rows of tick values with consecutive tick values alternating between rows. A compound policy such as STAGGERROTATE could be used to automatically choose the best fit policy for the situation.

footnote area
the region below the graph area where text produced by ENTRYFOOTNOTE statements appears.

frequency variable
in an input data set, a non-negative and non-zero integer variable that represents the frequency of occurrence of the current observation, essentially treating the data set as if each observation appeared n times, where n is the value of the FREQ variable for the observation.

fringe plot
a plot consisting of short, equal-length line segments drawn from and perpendicular to an axis. Each observation of a numeric variable corresponds to the location for a line segment.

function
See SAS function.
**glyph**
the most basic element (a grapheme or combination of graphemes) of a typeface or font that carries meaning in the text of a writing system. For example, the Z character can be represented by a number of different glyphs—boldface, italic, or in varying font styles, all of which represent the letter "Z."

**graph cell (cell)**
a distinct rectangular subregion of a graph that can contain plots, text, or legends.

**graph panel**
a graph with multiple cells.

**graphics template**
See ODS template.

**grid**
a uniform arrangement of the rows and columns of a multi-cell layout.

**gridded data**
input that contains at least three numeric variables. Two of the variables are treated as X and Y variables and the third variable Z is treated as if it were a function of X and Y. The X and Y variable values occur at uniformly spaced intervals (although the size and number of intervals might be different for X and Y). All X,Y pairs are unique, and Z values are interpolated so that every X,Y pair has a Z value. Raw data that has at least three numeric variables can be converted to gridded data with the G3GRID procedure (in SAS/GRAPH). The procedure offers both bivariate and spline interpolation methods for computing Z values.

**group index**
a numeric variable with positive integer values that correspond to values of a group variable. The index values are used to associate GraphData1 GraphDataN style elements with group values.

**group variable**
a variable in the input data set that is used to categorize chart variable values into groups. A group variable enables the data for each distinct group value to be rendered in a visually different manner. For example, a grouped scatter plot displays a distinct marker and color for each group value.

**image format**
a file format that displays a graphical representation. PNG, GIF, TIFF, and JPEG are examples of image formats, each with different characteristics.

**inset**
a graphical element such as a legend, line of text, or a table of text that is embedded inside of a graph's plot area.

**interval axis**
an axis where the distance between tick marks represents monotonically increasing or decreasing numeric units of some scale (like a ruler). The standard interval axis is called a LINEAR axis. Specialized interval axes include a TIME axis and a LOG axis.

**layout**
a generic term for a rectangular container that lays out the positions and sizes of its child components.
layout block
- a block beginning with a LAYOUT statement and ending with an ENDLAYOUT statement.

layout grid
- a multi-cell layout arranged as a grid of cells in rows and columns.

layout row (row)
- a set of layout cells that are side-by-side and share the same alignment.

layout type
- a keyword indicating the functionality of the layout. For example OVERLAY, LATTICE, and DATAPANEL are layout types.

legend entry
- a combination of a graphical element such as a marker or line along with text describing the value or use of the graphical element. A discrete legend can have several legend entries.

legend title
- text that explains how to interpret the legend.

line property
- a value that defines the pattern, thickness, or color of a line. By default, the value for a line property is derived from a style element in the current style. See also attribute bundle.

linear axis
- an interval axis with ticks placed on a linear scale.

log axis
- an axis displaying a logarithmic scale. A log axis is useful when data values span orders of magnitude.

macro variable reference
- a string that contains the name of a macro variable that is referenced in order to substitute a value that is located or defined elsewhere.

marker
- a symbol such as a diamond, a circle, or a triangle that is used to indicate the location of, or annotate, a data point in a plot or graph.

marker property
- a value that defines the symbol used as a marker, or its size, weight, or color. By default, the value for a marker property is derived from a style element in the current style. See also attribute bundle.

multi-cell layout
- a layout that supports a rectangular grid of cells, each of which can contain a graphical element, such as a plot, a legend, a nested layout, and so on.

nested layout
- a layout block that appears within the scope of another layout block.

ODS
- See Output Delivery System.
ODS Graphics
an extension to ODS that is used to create analytical graphs using the Graph Template Language.

ODS Graphics Editor
an interactive application that can be used to edit and annotate ODS Graphics output.

ODS style (style)
a combination of colors, fonts, lines, marker symbols, and so on that provide a specific appearance for SAS output. A style is defined in ODS by a style template.

ODS template (graphics template)
a description of how output should appear when it is formatted. ODS templates are stored as compiled entries in a template store, also known as an item store. Common template types include STATGRAPH, STYLE, CROSSTABS, TAGSET, and TABLE.

opaque
a property of a background. Opaque backgrounds are filled with a color. Non-opaque backgrounds are transparent.

outlier
a data point that differs from the general trend of the data by more than is expected by chance alone. An outlier might be an erroneous data point or one that is not from the same sampling model as the rest of the data.

Output Delivery System (ODS)
a component of SAS software that can produce output in a variety of formats such as markup languages (HTML, XML), PDF, listing, RTF, PostScript, and SAS data sets.

overlay
a plot that can be superimposed on another plot when specified within an overlay-type layout. A common overlay combination is a fit line on a scatter plot.

overlay layout
a type of layout that supports the superimposition of graphical components, such as plots, legends, and nested layouts.

parameterized plot
a non-computed plot that requires parameterized data. The Graph Template Language offers several plots in both computed and parameterized versions, for example, BARCHART and BARCHARTPARG. Some computed plots such as REGRESSIONPLOT can be emulated with a SERIESPLOT if the input data represented points on a fit line.

parent block
when two or more blocks are nested, any layout block that contains one or more layout blocks is a parent of the contained blocks.

plot
a visual representation of data such as a scatter plot, needle plot, or contour plot.

plot area
the space, bounded by the axes, where a visual representation of data, such as a scatter plot, a series line, or a histogram, is drawn.
plot type
a plot family such as bar chart (which would include horizontal, vertical, and grouped bar charts), or a classification scheme for plots based on some useful criteria, such as whether the plots are computed or parameterized.

primary axis
the X or Y axis contrasted to the X2 or Y2 secondary axis.

primary plot
the plot in an overlay that determines axis features, such as axis type and axis label.

prototype layout
an overlay plot composite that appears in each cell of a classification panel. Each instance of the prototype represents a different subset (classification level) of the data.

regression plot
a straight or curved line showing a linear or higher order regression fit for a set of points.

required argument
a variable or constant that must be specified in order to evaluate an expression or render a plot, legend, text, or a layout. For example, a scatter plot has two required arguments: X=column and Y=column.

role
a description of the purpose that a variable serves in a plot. For example, a series plot has predefined roles named for X, Y, GROUP, and CURVELABEL.

row
See layout row.

row axis
an external axis appearing on the left or right of a row of cells in a multi-cell layout.

row gutter
space between rows of cells of a multi-cell layout.

row header
typically, the text that identifies the row contents in a multi-cell layout. This text can be aligned to the right or left of the cells in a row. The row header is not restricted to text (it can contain a plot or a legend, for example).

row major order
an order for populating cells of a layout or entries of a legend when the number of columns is specified. For example, in the default case: Start at the top left and fill cells or entries left-to-right. When the right-most column is filled, begin a new row below the previous row. Continue this until all content items have been placed in cells or entries. There might be empty cells/entries in the last row.

row weight
in a LATTICE layout, the proportion of height allotted to a specific row of the layout. The sum of all row weights is 1.
**SAS function (function)**
a type of SAS language element that is used to process one or more arguments and then to return a result that can be used in either an assignment statement or an expression.

**secondary axis**
an X2 or Y2 axis, as contrasted with the primary axes X or Y.

**SGE file**
a file created in the ODS Graphics environment that contains an editable graph. Such files have a .sge file extension and can be edited only with the ODS Graphics Editor. You can edit SGE files from the SAS Results window or by opening the SGE file from within the ODS Graphics Editor.

**sidebar**
an area of certain multi-cell layouts external to the grid of cells where text or other graphical elements can appear. The LATTICE, DATAPANEL, and DATALATTICE layout support four sidebar areas (TOP, BOTTOM, LEFT, and RIGHT).

**single-cell layout**
a layout type that supports only one cell. The OVERLAY, OVERLAY3D, and OVERLAYEQUATED layouts are examples of single-cell layouts.

**sparse data**
in classification panels with two or more classifiers, some crossings of the classification values might not be present in the input data. Such input data is called sparse data. By default, a DATAPANEL layout does not generate cells for sparse data, but if requested, it can produce empty cells as place holders for the non-existent crossings.

**stand-alone plot**
a plot that has its own data range and can therefore appear by itself in a layout.

**statement block (block)**
a group of statements that has both a logical beginning and ending statement. For example, a LAYOUT statement along with its ENDLAYOUT statement and all contained statements are a block. Some blocks can be nested within other blocks.

**style**
See ODS style.

**style attribute**
a visual property, such as color, font properties, and line characteristics, that is defined in ODS with a reserved name and value. Style attributes are collectively referenced by a style element within a style template.

**style element**
a named collection of style attributes that affects specific parts of ODS output. For example, a style element might specify the color and font properties of title text or other text in in a table or graph. See also style attribute.

**style reference**
a part of the Graph Template Language syntax that indicates the current value of a specific attribute of a specific style element. For example, SIZE=GraphTitleText:FontSize means to assign to SIZE the value of the FontSize attribute of the GraphTitleText style element from the current style.
**template compile time**
the phase when the source program of a template definition is submitted. The syntax of the definition is evaluated for correctness. If no errors are detected, the definition is converted to a binary format and stored for later access.

**template definition (template source)**
the TEMPLATE procedure source program that creates a template. A template definition can be generated from a compiled template. Also called the template source.

**template run time**
the actions performed when a compiled template is bound to a data object and then rendered to produce a graph. Run-time errors can occur that prevent a graph from being produced.

**template source**
See template definition.

**template store**
an item store that contains definitions that were created by the TEMPLATE procedure. Definitions that SAS provides are in the item store Sashelp.Tmplmst. You can store definitions that you create in any template store to which you have Write access.

**template-based graphic**
graphical output produced by a compiled ODS template of the type STATGRAPH. That is, a graph that is produced within the ODS graphics environment rather than in the traditional device-based environment.

**text property**
any of a common set of characteristics that can be specified for any text string: color, family, size, weight, and style. By default, values for these properties are derived from a style element in the current style. See also attribute bundle, style attribute.

**time axis**
an axis type that displays only SAS date, time, or datetime values. Axis tick value increments can be specified as time or date intervals, such as MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR.

**title area**
the region above the graph area where text produced by ENTRYTITLE statements appears.

**transparency**
the degree to which a graphic element (such as a marker or filled area) is opaque or transparent. Transparency is indicated with a number from 0 (completely opaque) to 1 (completely transparent).

**Unicode**
a 16-bit encoding that is the industry standard for supporting the interchange, processing, and display of characters and symbols from most of the world's writing systems.

**wall**
the area bounded by orthogonal axis pairs. In two-dimensional graphs, there is one wall bounded by the XY axes. In three-dimensional graphs, there are three walls,
bounded by the XY, YZ, and XZ axes. A wall has an optional outline and can be opaque or transparent.

**weight variable**

a numeric variable that represents a weight (for example, costs) to be applied to observations.
Index

**Numbers**
- 2-D plots  62
- 3-D plots  62

**A**
- ACROSS= option
  - DISCRETELEGEND statement  329
- advanced multi-cell layout
  - See LAYOUT LATTICE statement
- alternating wall color bands
  - setting for discrete intervals  127
- anchoring
  - elements drawn with draw statements  396
- anti-aliasing
  - for graphics output  511, 524
- ANTIALIAS= option
  - ODS GRAPHICS statement  524
- ANTIALIASMAX= option
  - ODS GRAPHICS statement  524
- appearance
  - of grouped data in graphs  473
  - of grouped data in various plot types  474
  - of non-grouped data in graphs  470
- appearance of grouped data
  - controlling with a discrete attribute map  494
  - controlling with a range attribute map  499
- arrows
  - drawing with DRAWARROW statements  398
- ASORT function  577
- attribute maps
  - controlling the appearance of grouped data with  477
  - discrete  494
  - range  499
- AUTOALIGN= option
  - ENTRY statement  305, 352
  - LAYOUT GRIDDED statement  356
- legend statements  317
- AUTOITEMSIZE= option
- DISCRETELEGEND statement  332
- axis features
  - avoiding tick value collision  117, 134
  - axis line and wall outline  91
  - converting tick values to integers  115
  - data range on LINEAR axes  108
  - DISCRETE axes  97
  - displaying a secondary axis  96
  - displaying grid lines  96
  - displaying select features  95
  - equivalent primary and secondary axes  108
- general principles  85
- how affected by plot statements  86
- how axis label is determined  99
- how axis range is determined  98
- how axis type is determined  97
- how constructed  86, 97
- how tick values are determined  106
- label splitting  103
- LINEAR axes  97, 113
- LOG axes  97, 137
- log axis with log data  141
- mapping data columns to axes  86
- offsets on axes  110
- overlay-type layouts  85
- OVERLAYEQUATED layouts  159
- primary plot and  99
- scaling the tick values  116
- setting axis type  138
- specifying alternate short label  102
- specifying an axis label  100
- specifying axis options  95
- style elements that control features  94
- suppressing the axis label  96
- suppressing tick marks  95
- terminology  85
- thresholds on axes  106
- TIME axes  97, 131
- turning off the wall outline  93
- axis labels
  - on computed plots  101
  - specifying an alternate short label  102
specifying on an axis 100
splitting into multiple lines 103
suppressing on an axis 96
why truncated 102
axis options
BASE= suboption 139
DISPLAY= option 95
DISPLAYSECONDARY= option 96
INTERVAL= option 132
OFFSETMAX= 110
OFFSETMIN= 110
THRESHOLDMAX= 106
THRESHOLDMIN= 106
TICKINTERVALSTYLE= option 139
TICKVALUEFITPOLICY= option 118
TYPE= 97, 138
VIEWMAX= 98
VIEWMIN= 98
X2AXISOPTS= 95
XAXISOPTS= 95
Y2AXISOPTS= 95
YAXISOPTS= 95

B
BACKGROUND= option
text statements 301
BACKGROUNDCOLOR= option
LAYOUT GRIDDED statement 358
layout statements 23
legend statements 319
BANDPLOT statement
and SERIESPLOT statement 151
categorized by plot type 67
used for a fit plot 151
BARCHART statement
categorized by plot type 65
COLORBYFREQ= option 343
COLORMODEL= option 343
COLORRESPONSE= option 343
COLORSTAT= option 343
BARCHARTPARM statement 145
categorized by plot type 67
COLORMODEL= option 344
COLORRESPONSE= option 344
with a BLOCKPLOT 382
BASE=
axis log base 139
BEGINGRAPH statement
See also templates
DEFAULTDESIGNHEIGHT keyword 531
DEFAULTDESIGNWIDTH keyword 531
DESIGNHEIGHT= option 198, 219, 243, 521, 531
BEGINPLOYGON statement 402
BEGINPLOYLINE statement 401
BIHISTOGRAM3D Parm statement
and gaps among axis tick values 174
categorized by plot type 69
displaying percentages on Z axis 171
eliminating bins that have no data 171
labeling axes with endpoints 173
setting bin width 172
with PROC KDE 169
BLOCKPLOT statement
categorized by plot type 67
CLASS= option 382
creating an inset 377
DATATRANSPARENCY= option 381
DISPLAY= option 380
FILLTYPE= option 381
LABELATTRS= option 380
LABELPOSITION= option 380
VALUEATTRS= option 380
VALUEHALIGN= option 380
VALUEVALIGN= option 380
blocks
See statements
BORDER= option
layout statements 24
legend statements 321
ODS GRAPHICS statement 524
text statements 301
BORDERATTRS= option
layout statements 24
legend statements 321
borders
for graphics output 524
BOXPLOT statement
categorized by plot type 65
BOXPLOTPARM statement
categorized by plot type 67
BUBBLEPLOT statement
categorized by plot type 67
COLORMODEL= option 343
COLORRESPONSE= option 343
REVERSECOLORMODEL= option 343

C
CALL SYMPUT routine
on a DATA step 549, 552
using to create insets 364
CELL block
LAYOUT LATTICE statement 208
CELL HEADER block
LAYOUT LATTICE statement 208
CELLHEIGHTMIN= option classification panels 245
CELLWIDTHMIN= option classification panels 245
circles drawing with DRAWOVAL statements 399
CLASS= option
BLOCKPLOT statement 382
classification panels
adding gutters between panels 242
adding insets 280, 369
adjusting the graph size 243
and panel axis features 247
and the size of panel cells 244
CELLHEIGHTMIN= option 245
CELLWIDTHMIN= option 245
CLASSV ARS= argument 275
COLUMNAXISOPTS= option 249
COLUMNNDATARANGE= option 247
COLUMNNGUTTER= option 242
COLUMNHEADERS= option
(DATALATTICE layout only) 251
COLUMNS= option 240
COLVAR= argument 274
controlling classification headers 251
controlling headers 256
difference between DATAPANEL and
DATALATTICE layouts 239
graph aspect ratio 243
HEADERBACKGROUND COLOR= option 252
HEADERLABELATTRS= option 252
HEADERLABELDISPLAY= option 251
HEADEROPAQUE= option 252
INSET= option 280, 369
INSETOPTS= option 280, 369
LAYOUT DATALATTICE statement 236
LAYOUT DATAPANEL statement 236
LAYOUT PROTOTYPE statement 275
ORDER= option (DATAPANEL layout only) 240
overview 236
PANELNUMBER= option 262
restrictions on the PROTOTYPE layout 238
ROWAXISOPTS= option 249
ROWDATARANGE= option 247
ROWGUTTER= option 242
ROWHEADERS= option
(DATALATTICE layout only) 251
ROWS= option 240
ROWVAR= argument 274
side bars 255
IGINEMPTYCELLS= option 257
SPARSE= option (DATAPANEL layout only) 264
START= option 241
using sidebars 277
with parameterized plots 251
classification values
including missing values 267
CLI= option
REGRESSIONPLOT statement 150
CLM= option
REGRESSIONPLOT statement 150
cluster width 484
CLUSTERWIDTH= option 484
CMYK color codes 463
CNS (SAS Color Naming Scheme) 466
COLC function 577
COLLABEL function 577
COLN function 577
COLNAME function 577
COLOR= attribute
for fills 608
for lines 609
for markers 610
for text 611
COLORBANDS= option
DISCRETEOPTS option 127
COLORBANDSATTRS= option
DISCRETEOPTS option 127
COLORBYFREQ= option
BARCHART statement 343
COLORMODEL= option
BUBBLEPLOT statement 343
CONTOURPLOTPARM statement 343
HEATMAPPARM statement 343
SCATTERPLOT statement 343
SURFACEPLOTPARM statement 179,
343
WATERFALLCHART statement 343
COLORRESPONSE= option
BARCHART statement 343
BARCHARTPARM statement 344
BUBBLEPLOT statement 343
HEATMAP statement 344
HEATMAPPARM statement 343
HIGHLOWPLOT statement 344
LINECHART statement 344
SERIESPLOT statement 344
VECTORPLOT statement 344
WATERFALLCHART statement 343
colors
CMYK codes 463
Color Naming System values 466
gray-scale codes 465
HLS codes 464
HSB codes 464
HSV codes 464
naming schemes 461
RGB codes 462
SAS color names and RGB values 467
COLORSTAT= option
   BARCHART statement 343
   BARCHARTPARM statement 344
   HEATMAP statement 344
   HIGHLOWPLOT statement 344
   LINECHART statement 344
   SERIESPLOT statement 344
   VECTORPLOT statement 344
   WATERFALLCHART statement 343
COLUMNAXIS statement
   LAYOUT LATTICE statement 212
   COLUMNAXISOPTS= option
classification panels 249
   COLUMNDATARANGE= option
classification panels 247
   COLUMNGUTTER= option
classification panels 242
   LAYOUT GRIDDED statement 190, 355
   LAYOUT LATTICE statement 206
   using to space inset text 355
   COLUMNHEADERS= option
classification panels (DATALATTICE layout only) 251
   COLUMNHEIGHTS= option
   PREFERRED keyword 230
   COLUMNS= option
classification panels 240
   LAYOUT GRIDDED statement 188
   LAYOUT LATTICE statement 204
   COLUMNGENERATE= option
   LAYOUT LATTICE statement 216
   COMMONAXISOPTS= option
   LAYOUT OVERLAYEQUATED statement 162
   conditional logic 567
   confidence limits
      generating with REGRESSIONPLOT statement 150
      on a fit plot 151
   CONTINUOUSLEGEND statement
      See legends
   CONTOURPLOTPARM statement
categorized by plot type 65
   COLORMODEL= option 343
   CONTOURTYPE= option 343
   NHINT= option 343
   NLEVELS= option 343
   REVERSECOLORMODEL= option 343
   CONTOURTYPE= option
   CONTOURPLOTPARM statement 343
   CORROPTS= option
   SCATTERPLOTPARM statement 367
   CSS function 580
   CUBE= option
      LAYOUT OVERLAY3D statement 182
   CURVELABEL= option
      labeling drop lines 292
      labeling plot lines 292
      labeling reference lines 292
      plot statements 20
      CURVELABELATTRS= option
      plot statements 20
      CURVELABELLOCATION= option
      plot statements 20
      CURVELABELLOWER= option
      plot statements 20
      CURVELABELPOSITION= option
      plot statements 20
      CURVELABELUPPER= option
      plot statements 20
   custom style
      controlling grouped data appearance
      with 475
      creating 475
      CV function 580
      CYCLEATTRS= option
      varying visual properties of overlaid plots 326

D
   DAT drawing space 394
data object
      building with SGRENDER procedure 31
data points
      labeling 292
text properties for labels 293
data range
      setting on LINEAR axes 108, 113
      setting on TIME axes 135
data skins 508
      appearance in bar and pie charts 508
      appearance in scatter plots 509
   DATA step
      _NULL_ keyword 546
      and the output data object 550
      CALL SYMPUT routine 549, 552
      executing GTL templates 545
      FILE statement 546
      filtering the input data 547
      FORMAT statement 550
      LABEL statement 550
      OBJECT= option 551
OBJECTLABEL= option 551
OBS= option 547
ODS= option 549
PUT statement 546
syntax for executing GTL templates 546
transforming the input data 547
WHERE statement 547
data tips
for graphics output 429, 430, 524
setting maximum mouse-over areas 525
data, input
filtering 547
generating with a procedure 151
transforming 547
data, output
See output data object
DATA= argument
SGRENDER procedure 31
DATA= option
SGRENDER procedure 546
DATALABEL= option
labeling data points 292
plot statements 19
DATALABELATTRS= option
plot statements 19
DATALATTICE layout
See classification panels
DATAPANEL layout
See classification panels
DATASKIN= option 511
DATATRANSPERCY= option
plot statements 19
default appearance
grouped data 473
Default State
SAS Registry 520
DEFAULTDESIGNHEIGHT keyword
BEGINGRAPH statement 531
DEFAULTDESIGNWIDTH keyword
BEGINGRAPH statement 531
DEFINE statement
naming a GTL template 28
specifying type of template 28
DENDROGRAM statement
categorized by plot type 67
DENSITYPLOT statement
categorized by plot type 65
dependent plot statements 69
descriptive text
See text
Design Height
SAS Registry 520
Design Width
SAS Registry 521
DESIGNHEIGHT= option
BEGINGRAPH statement 198, 219, 243, 531
DESIGNWIDTH= option
BEGINGRAPH statement 198, 219, 243, 531
discrete attribute maps 494
creating 494
DISCRETE axes 97
discrete legend
adding items to 332, 334
controlling the label and item size in 332
filtering legend items contributed by multiple plots 336
removing items from 335
sorting items in 324
DISCRETELEGEND statement
See legends
DISCRETEOFFSET= option 129
display attributes
available line patterns 612
specifying fill color 608
specifying line color 609
specifying line pattern 609
specifying line thickness 609
specifying marker color 610
specifying marker size 610
specifying marker symbol 610
specifying marker weight 611
specifying text color 611
specifying text font family 611
specifying text size 611
specifying text style 611
specifying text weight 612
specifying transparency 608
DISPLAY= option
BLOCKPLOT statement 380
managing axis display 95
plot statements 18
primary-axis display features 95
DISPLAYCLIPPED= option
DISCRETELEGEND statement 329, 341
DISPLAYSECONDARY= option
displaying a secondary axis 96
secondary-axis display features 96
DOWN= option
DISCRETELEGEND statement 329
DPI= option
ODS PDF destination 522
DRAW statement
polygon block 402
polyline block 401
draw statements
DRAWARROW statement 398
DRAWIMAGE statement 403
drawing space and units 394
DRAWOVAL statement 399
DRAWRECTANGLE statement 400
DRAWTEXT statement 397
elements that can be drawn with 393
list of the statements and the elements
they draw 396
polygon block statements 402
polyline block statements 401
specifying anchor point in 396
DRAWARROW statement 398
DRAWIMAGE statement 403
drawing space
DATA 394
GRAPH 395
LAYOUT 395
specifying in draw statements 395
WALL 394
drawing space and units
See also DRAWSPACE= option
specifying for all draw statements 395
drawing units
specifying in draw statements 395
DRAWOVAL statement 399
DRAWRECTANGLE statement 400
DRAWSPACE= option 395
DRAWTEXT statement 397
drop lines
labeling 292
text properties for labels 293
DROPLINE statement
categorized by plot type 69
DSORT function 577
DYNAMIC statement
See dynamic variables
dynamic variables
and quotation marks 557
declaring 548, 555
initializing 549, 557
initializing on SGRENDER procedure 549
location in a template 555
overview 555
referencing 548, 556
syntax in a template 555
using in text 298
using on templates 548
using to create insets 359

E
edit-ready graphs
for graphics output 522, 538
ELLIPSE statement
categorized by plot type 65
ELLIPSEPARAM statement
categorized by plot type 67
ENTRY statement
adding descriptive text to a graph 292, 296
adding text to a graph 14
AUTOALIGN= option 305, 352
controlling the text 297
creating an inset 352, 353
greek letters 593
HALIGN= option 305, 352, 353
horizontal alignment of text 298
LAYOUT LATTICE statement 208
PAD= option 355
reserved keywords 593
rich text for 297
ROTATE= option 229
subscripts and superscripts 299
text background, borders, and padding 301
Unicode codes 299
Unicode values 593
VALIGN= option 305, 352
ENTRYFOOTNOTE statement
adding a footnote to a graph 14, 295
and size of graphical area 296
controlling the text 297
greek letters 593
horizontal alignment of text 298
location in a template 295
reserved keywords 593
rich text for 297
SHORTTEXT= option 303
subscripts and superscripts 299
text background, borders, and padding 301
text properties for 292
TEXTATTRS= option 297
TEXTFITPOICY= option 303
Unicode codes 299
Unicode values 593
ENTRYTITLE statement
adding a title to a graph 14, 295
and size of graphical area 296
controlling the text 297
greek letters 593
horizontal alignment of text 298
location in a template 295
reserved keywords 593
rich text for 297
SHORTTEXT= option 303
subscripts and superscripts 299
text background, borders, and padding 301
text properties for 292
TEXTATTRS= option 297
TEXTFITPOLICY= option 303
Unicode codes 299
Unicode values 593
EQUATETYPE= option
LAYOUT OVERLAYEQUATED statement 159
EVAL construct
with functions 573
EVAL function
See expressions
examples on the web 44
EXCLUDE= option
DISCRETELEGEND statement 335
EXISTS function 577
EXPAND function 577
expressions
and type conversion 566
building a text string 566
compared to SAS WHERE expressions 566
computing a constant 565
creating a new column 566
meaning in statement syntax 566
overview 565
EXTRACTSCALE =
axis tick values 115

F
FAMILY= attribute
for text 611
FILE statement
on a DATA step 546
fill attributes
specifying fill color 608
specifying transparency 608
FILLATTRS= option
plot statements 18
FILLTYPE= option
BLOCKPLOT statement 381
fit plot
with BANDPLOT statement 151
with confidence limits 151
with procedure-output data 151
with REGRESSIONPLOT statement 149
Fonts style element 453
footnotes
See ENTRYFOOTNOTE statement
FORMAT statement
on a DATA step 550
with the SGRENDER procedure 550
formats
See SAS formats
Formats

user-defined, with Unicode values 122, 633
FRINGEPLOT statement
categorized by plot type 67
functions
ASORT 577
COLC 577
COLLABEL 577
COLN 577
COLNAME 577
CSS 580
CV 580
DSORT 577
EXISTS 577
EXPAND 577
GTL only functions 576
IFC 574
IFN 574
KURTOSIS 580
LCLM 580
MAX 580
MEAN 580
MEDIAN 580
MIN 580
N 580
NMISS 580
NUMERATE 577
P1 580
P25 580
P5 580
P50 580
P75 580
P90 580
P95 580
P99 580
PROBT 581
PUT 359
Q1 581
Q3 581
ORANGE 581
RANGE 581
SKEWNESS 581
STDDEV 581
STDERR 581
STRIP 359
SUM 581
summary statistics functions 580
SUMWGT 581
T 581
TYPEOF 578
types available 573
UCLM 581
using to create insets 359
USS 581
VAR 581
G
  global legend
    creating 338
  GLOBALLEGEND statement 338
  glyphs
    lowercase Greek letters 593
    special characters 595
    uppercase Greek letters 594
  GPATH= option
    ODS HTML destination 523
    ODS LISTING destination 522
    output location for images 529
  graph borders 524
  GRAPH drawing space 395
  graph scaling
    graphics output 532
    graph styles
      See ODS styles
        GraphAltBlock style element 320
        GraphAnnoFont style element 454
        GraphAnnoText element 294
        GraphAxisLines style element 91
        GraphBlock style element 320
        GraphData1 style element 326
        GraphDataFont style element 454
        GraphDataText element 294
        GraphDataText style element 294
        GraphDataText style element 454
        GraphFootnoteFont style element 453
        GraphFootnoteText element 294
        GraphFooterText element 294
        GraphHeaderBackgroundColor style element
          320
        graphics output
          and BEGINGRAPH statement 531
          and SAS Registry 520
          anti-aliasing 511, 524
          edit-ready graphs 522, 538
          font sizes 533
          for MS Office applications 540
          graph borders 524
          graph height 524
          graph width 525
          image DPI (dots per inch) 536
          image format 524
          image resolution 522, 540
          imagemap for data tips 429, 430, 524
          managing 519
          naming output files 524, 552
          ODS destination options 522
          ODS styles 522
          output destinations 519
          output directory for images 522
          REGEDIT command 520
          reset ODS GRAPHICS options 525
          scale graphs proportionally 525
setting an image name 526
setting image format 526
shared templates 585
text in a graph 291
  GraphLabelFont style element 454
  GraphLabelText element 294
  GraphTitleFont style element 454
  GraphTitleText element 294
  GraphUnicodeFont style element 454
  GraphUnicodeText element 294
  GraphWalls style element 91, 358
  gray-scale color codes 465
grid lines
  displaying on an axis 96
  GRIDDED layout
    See LAYOUT GRIDDED statement
group display 481
  CLUSTER 481
  OVERLAY 482
  STACK 481
  group values
    including missing values 485
  GROUP= option
    plot statements 22
  GROUPDISPLAY= option 481
grouped data
  appearance in various plot types 474
  appearance of in graphs 473
  changing the order of 487
  controlling appearance with custom styles 475
  default appearance in graphs 473
  plots that support grouped data 473
grouped data appearance
  making independent of data order 489
grouped data order
  changing 487
  GROUPORDER= option 487, 488
H
  HALIGN= option
    ENTRY statement 305, 352
    LAYOUT GRIDDED statement 356
  layout statements 24, 25
  legend statements 315
  on text statements 298
  using to align insets 353
  HEADERBACKGROUNDCOLOR= option
  classification panels 252
  HEADERTAILATTRS= option
  classification panels 252
  HEADERTAILDISPLAY= option
classification panels 251
HEADEROPAQUE= option
classification panels 252
HEATMAP statement
categorized by plot type 66
COLORMODEL= option 344
COLORRESPONSE= option 344
COLORSTAT= option 344
REVERSECOLORMODEL= option 344
HEATMAPPARM statement
categorized by plot type 67
COLORMODEL= option 343
COLORRESPONSE= option 343
height
for graphics output 524
HEIGHT= option
ODS GRAPHICS statement 199, 220, 244, 341, 524, 532
HIGHLowPLOT statement
categorized by plot type 68
COLORMODEL= option 344
COLORRESPONSE= option 344
HISTOGRAM statement
categorized by plot type 66
HISTOGRAMPARM statement
categorized by plot type 68
HLS color codes 464
HSB color codes 464
HSV color codes 464

I
if-else statement
how and why used 568
nesting to form ELSE IF logic 569
overview 567
requirements for the conditional code 568
IFC function 574
IFN function 574
IMAGE_DPI= option
ODS HTML destination 523
ODS LISTING destination 522, 540
ODS RTF destination 522
IMAGEMAP= option
ODS GRAPHICS statement 524
IMAGENAME= option
ODS GRAPHICS statement 524, 552
images
default height in SAS Registry 520
default width in SAS Registry 521
drawing with DRAWIMAGE statement 403
format for graphics output 524
height for graphics output 524
names for graphics output 524, 552
output location 529
resolution for graphics output 522
scale for graphics output 525
width for graphics output 525
INCLUDEMISSINGGROUP= option 485
INDEX= option
mapping grouped data values 326
plot statements 22
INSET= option
classification panels 280, 369
SCATTERPLOTMATRIX statement 366
INSETOPTS= option
classification panels 280, 369
SCATTERPLOTMATRIX statement 367
insets
adding to a classification panel 369
adding to a SCATTERPLOTMATRIX graph 366
aligning on an axis 377
changing background fill 358
creating as table of text 193, 353
creating with a BLOCKPLOT statement 377
creating with an ENTRY statement 352, 353
creating with computed values 359
displaying title text for 355
overview 351
passing values to 361
positioning in a LATTICE layout 357
positioning in an OVERLAY layout 352, 356
INTEGER= option
formatting axis tick values 115
INTERVAL= axis tick values 132
item stores
storing a template 30

J
java environment
See memory management
JREOPTIONS option
managing Java memory 635

K
KURTOSIS function 580
LABEL statement
  on a DATA step 550
  with the SGRENDER procedure 327, 550
LABELATTRS= option
  BLOCKPLOT statement 380
LABELPOSITION= option
  BLOCKPLOT statement 380
labels
  for data points 292
  for drop lines 292
  for legends 292
  for plot lines 292
  for reference lines 292
text properties for 293
LATTICE layout
  See LAYOUT LATTICE statement
layout blocks
  BACKGROUNDCOLOR= option 23
  BORDER= option 24
  BORDERATTRS= option 24
  compared 15
  CYCLEATTRS= option 326
  features supported 23
  HALIGN= option 24, 25
  OPAQUE= option 23
  OUTERPAD= option 24
  PAD= option 24
  VALIGN= option 24, 25
layout containers
  See layout blocks
LAYOUT DATALLATTICE statement
  See also classification panels
  compared with other layouts 16
LAYOUT DATAPANEL statement
  See also classification panels
  compared with other layouts 16
LAYOUT draw space 395
LAYOUT GRIDDED statement
  adjusting graph size 198
  and empty cells 189
  AUTOALIGN= option 356
  BACKGROUNDCOLOR= option 358
  COLUMNGUTTER= option 190, 355
  COLUMNS= option 188
  compared with other layouts 16
  creating an inset 193, 353
  defining cells 190
  displaying title text 355
  HALIGN= option 353, 356
  ORDER= option 188, 354
  overview 187
  row and column sizes 194
  ROWGUTTER= option 190
  ROWS= option 188
  setting grid dimensions 188
  setting gutters 190
  setting up a grid 188
  sizing issues 194
  VALIGN= option 356
LAYOUT LATTICE statement
  adding cell headers 208
  adding sidebars 225
  adjusting graph size 218
  and empty cells 205
  and insets 357
  background color in cell headers 209
  cell axes 224
  CELL block 208
  CELL HEADER block 208
cell headers 202
column and row headers 203
COLUMNAXIS statement 212
COLUMNGUTTER= option 206
COLUMNS= option 204
COLUMNWEIGHTS= option 216
compared with other layouts 16
defining a basic lattice 204
defining cells 206
defining external axes 203, 212, 227
defining external axes and empty cells 214
defining external axes, restrictions 212
defining external secondary axes 213
defining internal axes 210
layout features 202
nested GRIDDED layout 209
ORDER= option 205
overview 201
rotating header text 229
ROWAXIS statement 212, 228
ROWDATARANGE= option 227
ROWGUTTER= option 206
ROWHEADERS block 228
ROWS= option 204
ROWWEIGHTS= option 216
setting grid dimensions 204
setting gutters 206
SIDEBAR block 225
sidebars 202
sizing issues 218
SKIPEMPTYCELLS= option 215
transforming the input data 221
uniform axis ranges 211
uniform axis ranges, restrictions 212
union of axes 203
using column or row headers 228
LAYOUT OVERLAY statement
  and appearance options 154
  and graph axes 85
  and grouped data 152
  and insets 352
and multiple axes
and plot axes
and primary plots
avoiding plot conflicts
common overlay combinations
compared with other layouts
CYCLEATTRS= option
nested in a GRIDDED layout
overview
plots with incompatible data
restrictions
specifying axis options
stacking order
statements allowed in layout
VIEWMAX= option
VIEWMIN= option
X2AXISOPTS= 
XAXIS= option
XAXISOPTS= 
Y2AXISOPTS= 
YAXIS= option
YAXISOPTS= 
LAYOUT OVERLAY3D statement
and bivariate histograms
and surface plots
compared with other layouts
CUBE= option
data requirements
defining a viewpoint
defining axes
displaying cube lines
displaying filled walls
overview
ROTATE= option
TILT= option
WALLDISPLAY= option
ZOOM= option
LAYOUT OVERLAYEQUATED statement
axis features
COMMONAXISOPTS= option
compared to OVERLAY layout
compared with other layouts
display features
EQUATETYPE= option
overview
types of axes
when to use
XAXISOPTS= option
YAXISOPTS= option
LAYOUT PROTOTYPE statement
See classification panels
LAYOUT REGION
as a top-level container
nesting in another layout
LCLM function
LEGENDAREAMAX= option
ODS GRAPHICS statement
LEGENDITEM statement
LEGENDLABEL= option
labeling legends
plot statements
legends
ACROSS= option
(DISCRETELEGEND only)
adding to a graph
and a continuous response variable
and plot statements
and the ODS GRAPHICS statement
arranging entries into columns and rows
assigning legend entry labels
AUTOALIGN= option
automatically aligning an inside legend
BACKGROUNDCOLOR= option
BORDER= option
BORDERATTRS= option
borders for
derewriting entry font sizes
continuous legends
CONTINUOUSLEGEND statement
controlling whether legend is drawn
discrete legends
DISCRETELEGEND statement
DISPLAYCLIPPED= option
(DISCRETELEGEND only)
displaying inside of plot wall
displaying outside of plot wall
down= option (DISCRETELEGEND only)
dropped legends
HALIGN= option
identifying overlaid plots
label for
legend wrapping
linking to plots
LOCATION= option
mapping grouped data values
NAME= option in plot statements
OPAQUE= option
options to control wrapping
ORDER= option (DISCRETELEGEND only)
dropping entries for grouped plots
ordering entries for non-grouped plots 326
organizing entries in a fixed number of columns 329
organizing entries in a fixed number of rows 331
plots that can use continuous legends 343
positioning a continuous legend 347
positioning options 315
scaling legend values in a continuous legend 349
showing group values 311
size issues in discrete legends 340
syntax 310
text properties for 322
text properties for legend titles 293
title borders 320
title for 292
TITLE= option 320
TITLEATTRS= option 322
TITLEBORDER= option 320
titles for 320
types of in GTL 310
using color gradients in a continuous legend 347
VALIGN= option 315
VALUEATTRS= option 322, 342
LEGENDTEXTITEMS statement 334
line attributes
available line patterns 612
specifying line color 609
specifying line pattern 609
specifying line thickness 609
LINEAR axes 97
data range on 108
EXTRACTSCALE = option 115
formatting tick values 115
INTEGER = option 115
setting data range 113
setting tick values 113
LINEATTRS= option 457
map of attributes to GTL options 458
plot statements 18
LINECHART statement
COLORMODEL= option 344
COLORRESPONSE= option 344
LINEPARM statement
categorized by plot type 69
LINEPLOT statement
categorized by plot type 66
LOCATION= option
legend statement 315
LOESSPLOT statement
categorized by plot type 66
LOG axes 97
formatting tick values 143
overview 137
setting log base 139
setting tick intervals 139

M
macro variables
and ampersand (&) preface 556
and current symbol table 548, 557
automatic macro variables 548, 557
data type when resolved 556
declaring 548, 555
difference between MVAR and NMVAR 556
finding runtime values 557
initializing 549, 557
location in a template overview 555
passing values to insets 361
referencing 548, 556
syntax in a template 555
using in text 298
using on templates 548
marker attributes
specifying marker color 610
specifying marker size 610
specifying marker symbol 610
specifying marker weight 611
MARKERATTRS= option 459
map of attributes to GTL options 459
plot statements 18
MARKERCOLORGRADIENT= option
SCATTERPLOT statement 343
MAX function 580
MAXLEGENDAREA= option
ODS GRAPHICS statement 341, 525
MEAN function 580
MEDIAN function 580
memory management
in java environment 635
JREOPTIONS option 635
MERGEDLEGEND statement 336
merging legends 336
MIN function 580
missing classification values 267
missing group values 485
MODELBAND statement
and REGRESSIONPLOT statement 150
categorized by plot type 69
MOSAICPLOTPARM statement
categorized by plot type 68
MS Office applications
graphics output for 540
multi-cell layout, advanced
See LAYOUT LATTICE statement
multi-cell layout, simple
See LAYOUT GRIDDED statement
MVAR statement
See macro variables

N
N function 580
NAME= option
plot statements 19, 311
names
color-naming schemes 461
colors 466
naming graphs 34
NEEDLEPLOT statement 148
categorized by plot type 68
NHINT= option
CONTOURPLOTPARM statement 343
NLEVELS= option
CONTOURPLOTPARM statement 343
NMISS function 580
NVAR statement
See macro variables
non-grouped data
appearance of in graphs 470
NOTES statement
syntax 555
NUMERATE function 577

O
OBJECT= option
DATA step 551
SGRENDER procedure 551
OBJECTLABEL= option
DATA step 551
SGRENDER procedure 551
OBS= option
on a DATA step 547
SGRENDER procedure 547
ODS destinations 32
and graphics output 522
DPI= option 522
GPATH= option 522
HTML destination 32, 523
IMAGE_DPI= option 522, 540
LISTING destination 32, 522
PDF destination 522
RTF destination 522
SGE= option 539
STYLE= option 33, 522
ODS Graphics
default state in SAS Registry 520
managing java memory 635
modifying shipped templates 589
ODS Graphics Designer (GUI-based
designer) 7
ODS Graphics Editor (GUI-based
editor) 7
ODS GRAPHICS statement 5
SAS/STAT 5
SG procedures 4
ODS GRAPHICS statement 5
and graphics output 523
ANTIALIAS= option 524
ANTIALIASMAX= option 524
BORDER= option 524
HEIGHT= option 199, 220, 244, 341, 524, 532
IMAGEMAP= option 524
IMAGENAME= option 34, 524
LEGENDAREAMAX= option 525
MAXLEGENDAREA= option 341, 525
naming an output image file 552
OFF option 523
OUTPUTFMT= option 524
RESET= option 34, 199, 523, 525
SCALE= option 525, 532
TIPMAX= option 525
WIDTH= option 34, 199, 220, 244, 341, 525, 532
ODS LISTING statement
STYLE= option 33
ODS PATH statement
controlling ODS search paths 455
ODS styles
and legend background color 320
appearance in graphs 446
assigning a background color to a graph 33
COLOR= text property 293
defining a style for axis walls 91
defining a style for fonts 451
elements that control axis features 94
FAMILY= text property 294
Fonts style element 453
GraphAltBlock style element 320
GraphAnnoFont style element 454
GraphAnnoText element 294
GraphAxisLines style element 91
GraphBlock style element 320
GraphData1 style element 326
GraphDataFont style element 454
GraphDataText element 252, 294
GraphFonts style element 453
GraphFootnoteFont style element 454
GraphFootnoteText element 294
GraphHeaderBackground style element 320
GraphLabelFont style element 454
GraphLabelText element 294
GraphTitleFont style element 454
GraphTitleText element 294
GraphUnicodeFont style element 454
GraphUnicodeText element 294
GraphValueFont style element 454
GraphValueText element 294
GraphWalls style element 91, 358
ODS PATH statement 455
ODS search paths 455
parent styles 452
registry keys for fonts in MS Windows 453
SIZE= text property 294
style elements available 597
STYLE= text property 294
supplied with SAS 448
text properties in 293
WEIGHT= text property 294
ODS= option
  on a DATA step 549
ODSTEMPLATE command
  finding a compiled template 30
OFF option
  ODS GRAPHICS statement 523
Office (MS) applications
  graphics output for 540
OFFSETMAX=
    axis offsets 110
OFFSETMIN=
    axis offsets 110
offsets
  on axes 110
OPAQUE= option
  layout statements 23
  legend statements 319
ORDER= option
  classification panels (DATAPANEL layout only) 240
  DISCRETELEGEND statement 329
  LAYOUT GRIDDED statement 188, 354
  LAYOUT LATTICE statement 205
OUTERPAD= option
  layout statements 24
output
  See graphics output
output data object
  and template execution 546
converting to a SAS data set 552
default label 550
default name 550
labeling 550
naming 550
setting data-column formats 550
setting data-column labels 550
viewing in the Results window 551
OUTPUTFMT= option
  ODS GRAPHICS statement 524
ovals
drawing with DRAWOVAL statements 399
OVERLAY layout
  See LAYOUT OVERLAY statement
OVERLAY3D layout
  See LAYOUT OVERLAY3D statement
OVERLAYEQUATED Layout
  See LAYOUT OVERLAYEQUATED statement

P
P1 function 580
P25 function 580
P5 function 580
P50 function 580
P75 function 580
P90 function 580
P95 function 580
P99 function 580
PAD= option
  ENTRY statement 355
  layout statements 24
text statements 302
  using to space inset text 355
PANELNUMBER= option
classification panels 262
PATH= option
output location for graphics output 529
PATTERN= attribute
  for lines 609
PBSPLINEPLOT statement
categorized by plot type 66
PIECHART statement
categorized by plot type 66
plot lines
  labeling 292
text properties for labels 293
plot statement
  using procedure output data 151
plot statements
  and legends 13
  axis labels on computed plots 101
BANDPLOT 67
BARCHART 65
BARCHARTPARM 67, 145, 382
BIHISTOGRAM3DPARM 69
BIHISTOGRAM3DPARM statement 169
BLOCKPLOT 67, 377
BOXPLOT 65
BOXPLOTPARM 67
BUBBLEPLOT 67
computed plots 61
CONTOURPLOTPARM 65
controlling text in a graph 291
CURVELABEL= option 20
CURVELABELATTRS= option 20
CURVELABELLOCATION= option 20
CURVELABELLOWER= option 20
CURVELABELPOSITION= option 20
CURVELABELUPPER= option 20
DATALABEL= option 19
DATALABELATTRS= option 19
DATATRANSPARENCY= option 19
DENDROGRAM 67
DENSITYPLOT 65
dependent plots 62, 69
DISPLAY= option 18
DROPLINE 69
ELLIPSE 65
ELLIPSEPARM 67
features supported 18
FILLATTRS= option 18
FRINGE PLOT 67
GROUP= option 22
HEATMAP 66
HEATMAPPARM 67
HIGHLIGHTPLOT 68
HISTOGRAM 66
HISTOGRAM PARM 68
INDEX= option 22
LEGENDLABEL= option 19
LINEATTRS= option 18
LINECHART 66
LINEPARM 69
LOESSPLOT 66
MARKERATTRS= option 18
MODEL BAND 69
MOSAICPLOTPARM 68
NAME= option 19
NEEDLEPLOT 68, 148
parameterized plots 62
PBSPLINEPLOT 66
PIECHART 66
plots with incompatible data 144
POLYGONPLOT 68
REFERENCECEL LINE 69, 145
REGRESSIONPLOT 66
ROLENAME= option 23
SCATTERPLOT 68, 148
SCATTERPLOTMATRIX 66
SERIESPLOT 68, 377
stand-alone plots 62
stand-alone, 2-D, computed 65
stand-alone, 2-D, parameterized 67
stand-alone, 3-D, parameterized 69
STEPPLOT 68
SURFACEPLOTPARM 69
SURFACEPLOTPARM statement 176
TEXTATTRS= option 18
TEXT PLOT 68
TIP= option 23
TIPFORMAT= option 23
TIPLABEL= option 23
VECTORPLOT 68
WATERFALLCHART 66
XAXIS= option 23
YAXIS= option 23
plot types
2-D plots 62
3-D plots 62
computed plots 61
dependent plots 62
primary plots 63
stand-alone plots 62
POLYGONPLOT statement
categorized by plot type 68
primary plots 62, 63
avoiding plot conflicts 90
determining axis features 100
PRIMARY=
axis features 99
PRIMARY= option
avoiding plot conflicts 90
PROBT function 581
PROC SGRENDER statement
See SGRENDER procedure
programming features, runtime 573
PROTOTYPE layout
See classification panels
PUT function
using to create insets 359
PUT statement
on a DATA step 546

Q
Q1 function 581
Q3 function 581
QRANGE function 581

R
range attribute maps 499
controlling graph appearance with 499
RANGE function 581
rectangles
drawing with DRAWRECTANGLE statements 400
reference lines
labeling 292
Index

text properties for labels 293
REFERENCELINE statement 145
categorized by plot type 69
REGEDIT command
opening Registry Editor window 520
Registry Editor window
settings for ODS Graphics 520
REGRESSIONPLOT statement
and MODEL.BAND statement 150
and SCATTERPLOT statement 149
categorized by plot type 66
CLI= option 150
CLM= option 150
generating confidence limits 150
used for a fit plot 149
RESET= option
ODS GRAPHICS statement 199, 523, 525
Results window
viewing graphics output 32
REVERSECOLORMODEL= option
BUBBLEPLOT statement 343
CONTOURPLOT.PARM statement 343
SCATTERPLOT statement 343
SURFACEPLOT.PARM statement 179, 343
RGB color codes 462
SAS color names 467
ROLENAME= option
plot statements 23
ROTATE= option
ENTRY statement 229
LAYOUT OVERLAY3D statement 184
ROWAXIS statement
LAYOUT LATTICE statement 212, 228
ROWAXISOPTS= option
classification panels 249
ROWDATARANGE= option
classification panels 247
LAYOUT LATTICE statement 227
ROWGUTTER= option
classification panels 242
LAYOUT GRIDDED statement 190
LAYOUT LATTICE statement 206
ROWHEADERS block
LAYOUT LATTICE statement 228
ROWHEADERS= option
classification panels (DATALATTICE layout only) 251
ROWHEIGHTS= option
PREFERRED keyword 230
ROWS= option
classification panels 240
LAYOUT GRIDDED statement 188
LAYOUT LATTICE statement 204
ROWWEIGHTS= option
LAYOUT LATTICE statement 216
runtime programming constructs
GTL 565
runtime programming features 573
S
SAS Color Naming Scheme (CNS) 466
SAS formats
error message in log 631
unsupported currency formats 633
unsupported date and time formats 632
unsupported numeric formats 631
using 631
SAS Registry
and graphics output 520
Default State for ODS Graphics 520
Design Height for ODS Graphics 520
Design Width for ODS Graphics 521
opening with REGEDIT command 520
SAS/STAT 5
Sasuser.Templat item store
storing templates 30
SCALE= option
ODS GRAPHICS statement 525, 532
scaling graphs proportionally
for graphics output 525
SCATTERPLOT statement 148
and REGRESSIONPLOT statement 149
categorized by plot type 68
COLORMODEL= option 343
MARKERCOLORGRADIENT= option 343
REVERSECOLORMODEL= option 343
SCATTERPLOTMATRIX statement
adding an inset to the graph 366
categorized by plot type 66
CORROPTS= option 367
INSET= option 366
INSETOPTS= option 367
SERIESPLOT statement
and BANDPLOT statement 151
categorized by plot type 68
COLORMODEL= option 344
COLORRESPONSE= option 344
with a BLOCKPLOT 377
SGE= option
ODS LISTING destination 539
SGPANEL procedure 5
SGPLOT procedure 5
SGRENDER procedure 34
and the data object 31
and the output data object 550
DATA= argument 31
DATA= option 546
DYNAMIC statement 549
executing a compiled template 31
executing GTL templates 545
filtering the input data 547
OBJECT= option 551
OBJECTLABEL= option 551
OBS= option 547
syntax for executing GTL templates 546
TEMPLATE= argument 31
TEMPLATE= option 546
transforming the input data 547
WHERE statement 547
with a FORMAT statement 550
with a LABEL statement 327, 550
SGSCATTER procedure 5
SHORTTEXT= option 303
side bars
classification panels 255
SIDEBAR block
LAYOUT LATTICE statement 225
simple multi-cell layout
See LAYOUT GRIDDED statement
simple plot layouts
See LAYOUT OVERLAY statement
SIZE= attribute
for markers 610
for text 611
sizing graphs 34
SKEWNESS function 581
skins
See data skins
SKIPEMPTYCELLS= option
classification panels 257
LAYOUT LATTICE statement 215
sorting functions 577
SORTORDER= option
DISCRETELEGEND statement 324
SPARSE= option
classification panels (DATAPANEL layout only) 264
squares
drawing with DRAWRECTANGLE statements 400
stand-alone plot statements 65
START= option
classification panels 241
statements
blocks 12
categories of 13
general syntax for 11
layout blocks 15
nested blocks 12
overview 11
parent-child relationships 12
Statistical Graphics (SG) procedures 4
SGPANEL procedure 5
SGPLOT procedure 5
SGSCATTER procedure 5
STDDEV function 581
STDERR function 581
STEPLOT statement
categorized by plot type 68
STRIP function
using for insets 359
style elements
See ODS styles
style-reference
fill options 608
line options 609
marker options 610
text options 611
STYLE= attribute
for text 611
STYLE= option
ODS HTML destination 523
ODS LISTING destination 522
ODS LISTING statement 33
ODS PDF destination 522
ODS RTF destination 522
styles
See ODS styles
subscripts and superscripts 299
SUM function 581
SUMWGT function 581
SURFACECOLORGRADIENT= option
SURFACEPLOTPARM statement 343
SURFACECOLORGRADIENT= option
SURFACEPLOTPARM statement 179
SURFACEPLOTPARM statement
and missing Z values 177
categorized by plot type 69
COLORMODEL= option 179, 343
default features 176
REVERSECOLORMODEL= option 179, 343
SURFACECOLORGRADIENT= option 343
SURFACECOLORGRADIENT= option 179
SURFACETYPE= option 178
with PROC G3GRID 177
SURFACETYPE= option
SURFACEPLOTPARM statement 178
SYMBOL= attribute
for markers 610
SYMPUT
See CALL SYMPUT routine
Index

T

T function 581

TEMPLATE procedure
  defining a style template for fonts 452
  DYNAMIC statement 555
  MVAR statement 555
  NMVAR statement 555
  NOTES statement 555
template store
  for shared templates 585
  TEMPLATE= argument
  SGRENDER procedure 31
  TEMPLATE= option
  SGRENDER procedure 546
templates
  anti-aliasing 524
  BEGINGRAPH statement 28
  browsing the compiled source 31
  compiling 30
  creating 27
  DATA step 545
  DEFINE block 28
  directing output 32
dynamic variables on 555
  edit-ready graphs 522, 538
  executing 31, 545
  filtering the input data 547
  finding in an item store 30
  formatting output columns 550
graph block 28
graph borders 524
graph footnotes 295
graph height 520, 524
graph titles 295
graph width 521, 525
image format 524
image resolution 522, 540
imagemap for data tips 429, 430, 524
labeling output columns 550
layout block 29
macro variables on 555
managing graphics output 519
modifying graph appearance 33
modifying shipped templates 589
naming 28
naming output files 32, 34, 524
ODS destinations 32, 522
ODS styles 33, 522
ODTEMPLATE command 30
output directory for images 522
reset ODS GRAPHICS options 525
Results window 32
runtime programming constructs 565
Sasuser.Templat item store 30
scaling graphs proportionally 525
SGRENDER procedure 31, 34, 545
shared templates 585
sizing graphs 34
specifying graph contents 29
style elements available 597
transforming the input data 547
viewing graphics output 32
Templates Browser window
  browsing a compiled template 31
Templates window
  finding a compiled template 30
text
  adding and changing in a graph 291
  BACKGROUND= option 301
  BORDER= option 301
  COLOR= property 293
  CURVELABEL= option 292
  DATALABEL= option 292
drawing with DRAWTEXT statement 397
ENTRY statement 292, 296
ENTRYFOOTNOTE statement 295
ENTRYTITLE statement 295
FAMILY= property 294
GraphAnnoText element 294
GraphDataText element 294
GraphFootnoteText element 294
GraphLabelText element 294
GraphTitleText element 294
GraphUnicodeText element 294
GraphValueText element 294
horizontal alignment of 298
LABEL= option 292
LEGENDLABEL= option 292
PAD= option 302
positioning entry text 305
rich text for strings 297
SIZE= property 294
special characters 595
STYLE= property 294
subscripts and superscripts 299
text properties for 293
TITLE= option 292
Unicode codes 299
using dynamic variables and macro
  variables 298
WEIGHT= property 294
text attributes
  specifying text color 611
  specifying text font family 611
  specifying text size 611
  specifying text style 611
  specifying text weight 612
TEXTATTRE= option 460
  on text statements 297
plot statements 18
TEXTFITPOLICY= option
Index 677

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXTPLOT statement</td>
<td>303</td>
</tr>
<tr>
<td>categorized by plot type</td>
<td>68</td>
</tr>
<tr>
<td>THICKNESS= attribute for lines</td>
<td>609</td>
</tr>
<tr>
<td>THRESHOLDMAX= axis thresholds</td>
<td>106</td>
</tr>
<tr>
<td>THRESHOLDMIN= axis thresholds</td>
<td>106</td>
</tr>
<tr>
<td>thresholds on axes</td>
<td>106</td>
</tr>
<tr>
<td>tick marks</td>
<td>95</td>
</tr>
<tr>
<td>suppressing on an axis</td>
<td>95</td>
</tr>
<tr>
<td>tick value fit policy applicability</td>
<td>627</td>
</tr>
<tr>
<td>tick values</td>
<td>117</td>
</tr>
<tr>
<td>avoiding collision</td>
<td>124</td>
</tr>
<tr>
<td>converting to integers</td>
<td>115</td>
</tr>
<tr>
<td>fit policies</td>
<td>124</td>
</tr>
<tr>
<td>fit policy applicability</td>
<td>627</td>
</tr>
<tr>
<td>formatting on LINEAR axes</td>
<td>115</td>
</tr>
<tr>
<td>formatting on LOG axes</td>
<td>143</td>
</tr>
<tr>
<td>formatting on TIME axes</td>
<td>134</td>
</tr>
<tr>
<td>how determined on an axis</td>
<td>106</td>
</tr>
<tr>
<td>scaling</td>
<td>116</td>
</tr>
<tr>
<td>setting on LINEAR axes</td>
<td>113</td>
</tr>
<tr>
<td>setting on TIME axes</td>
<td>132</td>
</tr>
<tr>
<td>TICKINTERVALSTYLE= axis tick intervals</td>
<td>139</td>
</tr>
<tr>
<td>TICKVALUEFITPOLICY axis tick values</td>
<td>118</td>
</tr>
<tr>
<td>TICKVALUEFORMAT = axis tick values</td>
<td>115</td>
</tr>
<tr>
<td>log axis tick values</td>
<td>134</td>
</tr>
<tr>
<td>TICKVALUEFORMAT= XAXISPOPTS= option</td>
<td>227</td>
</tr>
<tr>
<td>TICKVALUELIST= option</td>
<td>113</td>
</tr>
<tr>
<td>DISCRETEOPTS option</td>
<td>135</td>
</tr>
<tr>
<td>TILT= option</td>
<td>121</td>
</tr>
<tr>
<td>LAYOUT OVERLAY3D statement</td>
<td>184</td>
</tr>
<tr>
<td>TIME axes</td>
<td>97</td>
</tr>
<tr>
<td>formatting tick values</td>
<td>134</td>
</tr>
<tr>
<td>overview</td>
<td>131</td>
</tr>
<tr>
<td>setting data range</td>
<td>135</td>
</tr>
<tr>
<td>setting tick values</td>
<td>132</td>
</tr>
<tr>
<td>TIP= option</td>
<td>23</td>
</tr>
<tr>
<td>plot statements</td>
<td>23</td>
</tr>
<tr>
<td>TIPFORMAT= option</td>
<td>23</td>
</tr>
<tr>
<td>plot statements</td>
<td>23</td>
</tr>
<tr>
<td>TIPLABEL= option</td>
<td>23</td>
</tr>
<tr>
<td>plot statements</td>
<td>23</td>
</tr>
<tr>
<td>TIPMAX= option</td>
<td>23</td>
</tr>
<tr>
<td>ODS GRAPHICS statement</td>
<td>525</td>
</tr>
<tr>
<td>TITLE= option</td>
<td>292</td>
</tr>
<tr>
<td>defining a title for a legend</td>
<td>320</td>
</tr>
<tr>
<td>legend statements</td>
<td>322</td>
</tr>
<tr>
<td>TITLEBORDER= option</td>
<td>320</td>
</tr>
<tr>
<td>legend statements</td>
<td>320</td>
</tr>
<tr>
<td>titles</td>
<td>320</td>
</tr>
<tr>
<td>See ENTRYTITLE statement</td>
<td></td>
</tr>
<tr>
<td>TRANSPARENCY= attribute for fills</td>
<td>608</td>
</tr>
<tr>
<td>type conversion expressions</td>
<td>566</td>
</tr>
<tr>
<td>TYPE= axis type</td>
<td>97</td>
</tr>
<tr>
<td>TYPE= option</td>
<td>138</td>
</tr>
<tr>
<td>DISCRETELEGEND statement</td>
<td>336</td>
</tr>
<tr>
<td>TYPOE function</td>
<td>578</td>
</tr>
<tr>
<td>Unicode codes</td>
<td>299</td>
</tr>
<tr>
<td>Unicode values</td>
<td>591</td>
</tr>
<tr>
<td>greek letters, lower case</td>
<td>593</td>
</tr>
<tr>
<td>greek letters, upper case</td>
<td>594</td>
</tr>
<tr>
<td>hexadecimal values</td>
<td>593</td>
</tr>
<tr>
<td>in user-defined formats</td>
<td>122</td>
</tr>
<tr>
<td>keywords</td>
<td>593</td>
</tr>
<tr>
<td>special characters</td>
<td>595</td>
</tr>
<tr>
<td>User-defined formats</td>
<td>122</td>
</tr>
<tr>
<td>with Unicode values</td>
<td>633</td>
</tr>
<tr>
<td>USS function</td>
<td>581</td>
</tr>
<tr>
<td>UCLM function</td>
<td>581</td>
</tr>
<tr>
<td>V ALIGN= option</td>
<td>305</td>
</tr>
<tr>
<td>ENTRY statement</td>
<td>352</td>
</tr>
<tr>
<td>LAYOUT GRIDDED statement</td>
<td>356</td>
</tr>
<tr>
<td>layout statements</td>
<td>24</td>
</tr>
<tr>
<td>legend statements</td>
<td>25</td>
</tr>
<tr>
<td>VALUEATRBS= option</td>
<td>315</td>
</tr>
<tr>
<td>BLOCKPLOT statement</td>
<td>380</td>
</tr>
<tr>
<td>legend statements</td>
<td>322</td>
</tr>
<tr>
<td>VALUEHALIGN= option</td>
<td>342</td>
</tr>
<tr>
<td>BLOCKPLOT statement</td>
<td>380</td>
</tr>
<tr>
<td>VALUEALIGN= option</td>
<td>380</td>
</tr>
<tr>
<td>BLOCKPLOT statement</td>
<td>380</td>
</tr>
<tr>
<td>VAR function</td>
<td>581</td>
</tr>
<tr>
<td>VECTORPLOT statement</td>
<td>68</td>
</tr>
<tr>
<td>categorized by plot type</td>
<td>344</td>
</tr>
<tr>
<td>COLORMODEL= option</td>
<td>344</td>
</tr>
<tr>
<td>COLORRESPONSE= option</td>
<td>344</td>
</tr>
<tr>
<td>VIEWMAX= axis data range</td>
<td>98</td>
</tr>
<tr>
<td>VIEWMIN= axis data range</td>
<td>108</td>
</tr>
</tbody>
</table>
W
WALL drawing space 394
WALLDISPLAY= option
  LAYOUT OVERLAY3D statement 183
WATERFALLCHART statement
categorized by plot type 66
COLORMODEL= option 343
COLORRESPONSE= option 343
COLORSTAT= option 343
WEIGHT= attribute
  for markers 611
  for text 612
WHERE statement
  on a DATA step 547
  SGRENDER procedure 547
width
  for graphics output 525
WIDTH= option
  ODS GRAPHICS statement 199, 220, 244, 341, 525, 532

X
X2AXISOPTS= option
  general syntax 95
XAXIS= option
  mapping data columns 87
  plot statements 23
XAXISOPTS= option
  general syntax 95
LABELSPLIT= 103
LAYOUT OVERLAYEQUATED statement 162
SHORTLABEL= 102
VIEWMAX= 108
VIEWMIN= 108
XAXISPOTS= option
  TICKVALUEFORMAT= 227

Y
Y2AXISOPTS= option
  general syntax 95
YAXIS= option
  mapping data columns 87
  plot statements 23
YAXISOPTS= option
  DISPLAY= 197
  general syntax 95
  LAYOUT OVERLAYEQUATED statement 162

Z
ZOOM= option
  LAYOUT OVERLAY3D statement 184
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore for additional books and resources.