



SAS Publishing



SAS/GRAPH[®] 9.1 Reference

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *SAS/GRAPH® 9.1 Reference, Volumes 1 and 2*. Cary, NC: SAS Institute Inc.

SAS/Graph® 9.1 Reference, Volumes 1 and 2

Copyright © 2004, SAS Institute Inc., Cary, NC, USA

ISBN 159047-408-2

All rights reserved. Produced in the United States of America. Your use of this e-book shall be governed by the terms established by the vendor at the time you acquire this e-book.

U.S. Government Restricted Rights Notice: Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19, Commercial Computer Software-Restricted Rights (June 1987).

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

1st printing, January 2004

SAS Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at support.sas.com/pubs or call 1-800-727-3228.

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

Other brand and product names are trademarks of their respective companies

Contents

What's New **xiii**

Overview **xiii**

Details **xiv**

PART 1 SAS/GRAPH Concepts 1

Chapter 1 △ Introduction to SAS/GRAPH Software 3

Overview **4**

Generating Graphs **4**

About this Book **16**

Conventions Used in This Book **16**

Information You Should Know **20**

Chapter 2 △ SAS/GRAPH Programs 25

Overview **25**

Language Elements **26**

SAS Data Sets **29**

Using Engines with SAS/GRAPH Software **31**

Running SAS/GRAPH Programs **31**

Procedure Output and the Graphics Output Area **34**

Chapter 3 △ Device Drivers 41

Overview **41**

About Device Drivers **42**

Selecting a Device Driver **43**

Controlling Output with Device Drivers **45**

Chapter 4 △ SAS/GRAPH Output 47

About SAS/GRAPH Output **48**

Displaying Graphics Output on Monitors or Terminals **49**

Printing Graphics Output **51**

Storing Graphics Output in SAS Catalogs **53**

Modifying SAS/GRAPH Output **55**

Transporting and Converting Graphics Output **56**

About Exporting SAS/GRAPH Output **59**

Exporting SAS/GRAPH Output Interactively **62**

Exporting SAS/GRAPH Output with Program Statements **62**

Exporting SAS/GRAPH Output Using Modified Device Entries **72**

Chapter 5 △ SAS/GRAPH Fonts 75

Overview **75**

Specifying Fonts in SAS/GRAPH Programs **75**

Using Hardware Fonts **78**

Specifying Special Characters	81
Using SAS/GRAPH Software Fonts	82
Chapter 6 \triangle SAS/GRAPH Colors and Images	91
Using SAS/GRAPH Colors and Images	92
Specifying Colors in SAS/GRAPH Programs	92
Specifying Images in SAS/GRAPH Programs	106
Chapter 7 \triangle SAS/GRAPH Statements	121
Overview	123
AXIS Statement	124
BY Statement	141
FOOTNOTE Statement	146
GOPTIONS Statement	146
LEGEND Statement	151
NOTE Statement	164
ODS HTML Statement	164
PATTERN Statement	169
SYMBOL Statement	183
TITLE, FOOTNOTE, and NOTE Statements	210
Example 1. Ordering Axis Tick Marks with SAS Datetime Values	226
Example 2. Specifying Logarithmic Axes	229
Example 3. Rotating Plot Symbols through the Colors List	231
Example 4. Creating and Modifying Box Plots	233
Example 5. Filling the Area between Plot Lines	236
Example 6. Enhancing Titles	238
Example 7. Using BY-group Processing to Generate a Series of Charts	240
Example 8. Creating a Simple Web Page with the ODS HTML Statement	245
Example 9. Combining Graphs and Reports in a Web Page	248
Example 10. Creating a Bar Chart with Drill-down for the Web	255
See Also	260
Chapter 8 \triangle Graphics Options and Device Parameters Dictionary	261
Introduction	261
Specifying Graphics Options and Device Parameters	261
Dictionary of Graphics Options and Device Parameters	262
PART 2 Bringing SAS/GRAPH Output to the Web	367
Chapter 9 \triangle Introducing SAS/GRAPH Output for the Web	369
Which Device Driver or Macro Do I Use?	369
Types of Web Presentations Available	370
Selecting a Type of Web Presentation	378
Generating Web Presentations	382
Changing the Location of Online Help for Java and ActiveX	385

Chapter 10 △ **Creating Interactive Output for ActiveX** 387

- Overview 387
- When to Use the ACTIVEX Device Driver 388
- Installing the ActiveX Control 389
- Generating Output for ActiveX 391
- Configuring Drill-Down Links with ACTIVEX 392
- ActiveX Examples 393
- Creating Graphs Interactively 395

Chapter 11 △ **Creating Interactive Output for Java** 397

- Overview 397
- When to Use the JAVA Device Driver 398
- Generating Output for Java 398
- Configuring Drill-Down Links for Java and ActiveX 400
- Examples of Interactive Java Output 415

Chapter 12 △ **Attributes and Parameters for Java and ActiveX** 421

- Specifying Parameters and Attributes for Java and ActiveX 421
- Parameter Reference for Java and ActiveX 424

Chapter 13 △ **Generating Static Graphics** 439

- What is a Static Graphic? 439
- Creating a Static Graphic with ODS 440
- ACTXIMG and JAVAIMG Device Drivers Compared to GIF, JPEG, and PNG Device Drivers 440
- Developing Web Presentations with the JAVAIMG and ACTXIMG Device Drivers 442
- Developing Web Presentations with the GIF, JPEG, and PNG Device Drivers 443
- Naming Conventions Used for Image Output Files 445
- Enhancing Web Presentations Generated with the GIF, JPEG, or PNG Device Driver 446
- Sample Programs for Static Images 447

Chapter 14 △ **Generating Web Animation with GIFANIM** 457

- Developing Web Presentations with the GIFANIM Device Driver 457
- When to Use the GIFANIM Device Driver 457
- Creating an Animated Sequence 458
- GOPTIONs for Configuring GIFANIM Presentations 459
- Sample Programs: GIFANIM 459

Chapter 15 △ **Generating Interactive Metagraphics Output** 469

- Developing Web Presentations for the Metaview Applet 469
- Using ODS with JAVAMETA 470
- Using the META2HTM Macro 471
- Adding Run-Time Controls to a Presentation 471
- Enhancing Web Presentations for the Metaview Applet 474
- Specifying Non-English Resource Files and Fonts 474
- Metaview Applet Parameters 475

META2HTM Macro Arguments	478
Sample Programs: Metaview Applet	478
Chapter 16 △ Managing Web Output with ODS	487
Overview of ODS Enhancements for Web Output	487
Using ODS Styles	488
Managing ODS Destinations	489
ODS and Procedures that Support RUN-Group Processing	490
Specifying Body Files for Displaying Graphs	491
Controlling Titles and Footnotes with ODS Output	492
Adding Non-Graphics Output to a Web Page	494
Linking to Output through a Table of Contents	495
Linking to Output through a Table of Pages	496
Using Frames to Display ODS Output	497
Chapter 17 △ Generating Web Output with the Annotate Facility	499
Overview of Generating Web Output with the Annotate Facility	499
Generating Web Output with the Annotate Facility	499
Examples	501
Chapter 18 △ Creating Interactive Treeview Diagrams	503
Creating Treeview Diagrams	503
Enhancing Presentations for the Treeview Applet	506
DS2TREE Macro Arguments	507
Sample Programs: Treeview Macro	507
Chapter 19 △ Creating Interactive Constellation Diagrams	513
Creating Constellation Diagrams	513
Enhancing Presentations for the Constellation Applet	517
DS2CONST Macro Arguments	518
Sample Programs: Constellation Macro	518
Chapter 20 △ Creating Critical Success Factor Diagrams	527
Using the DS2CSF Macro	527
Enhancing Presentations for the Rangeview Applet	529
DS2CSF Macro Arguments	530
Sample Programs: DS2CSF Macro	530
Chapter 21 △ Macro Arguments for the DS2CONST, DS2TREE, DS2CSF, and META2HTM Macros	535
Macro Arguments	535
Chapter 22 △ Enhancing Web Output	567
Enhancing Web Output	567
Adding Data Tips to Web Presentations	568
Adding Drill-Down Links to Web Presentations	571
Chapter 23 △ Troubleshooting Web Output	579

Troubleshooting Web Output	579
Checking Browser Permissions	582
Using HTML Character Entities	582
Connecting to Web Servers that Require Authentication	583
Removing CLASSPATH Environment Variables	583
Correcting Text Fonts	583
Resolving Colors in Netscape	583
Resolving Differences Between Client and Server Graphs	584

PART 3 The Annotate Facility 585

Chapter 24 △ Using Annotate Data Sets	587
Overview	587
About the Annotate Data Set	589
About Annotate Graphics	595
Creating an Annotate Data Set	599
Producing Graphics Output from Annotate Data Sets	601
Annotate Processing Details	602
Examples	604
Chapter 25 △ Annotate Dictionary	613
Annotate Dictionary Overview	614
Annotate Functions	615
Annotate Variables	642
Annotate Internal Coordinates	678
Annotate Macros	679
Using Annotate Macros	697
Annotate Error Messages	699

PART 4 SAS/GRAPH Procedures 705

Chapter 26 △ The GANNO Procedure	707
Overview	707
Procedure Syntax	708
Examples	710
Chapter 27 △ The GAREABAR Procedure	725
Overview	725
Concepts	726
Procedure Syntax	727
Examples	729
Chapter 28 △ The GBARLINE Procedure	739
Overview	739
Concepts	741
Procedure Syntax	749

Examples	768
Chapter 29 \triangle The GCHART Procedure	773
Overview	774
Concepts	778
Procedure Syntax	785
Examples	842
References	884
Chapter 30 \triangle The GCONTOUR Procedure	885
Overview	885
Concepts	885
Procedure Syntax	888
Examples	904
References	913
Chapter 31 \triangle The GDEVICE Procedure	915
Overview	916
Concepts	916
Procedure Syntax	920
Using the GDEVICE Procedure	928
Examples	936
Chapter 32 \triangle The GFONT Procedure	939
Overview	939
Concepts	940
Procedure Syntax	942
Creating a Font	951
Examples	962
Chapter 33 \triangle The GIMPORT Procedure	969
Overview	969
Concepts	970
Procedure Syntax	972
Examples	976
References	981
Chapter 34 \triangle The GKEYMAP Procedure	983
Overview	983
Concepts	983
Procedure Syntax	988
Examples	990
Chapter 35 \triangle The GMAP Procedure	995
Overview	996
Concepts	999
Procedure Syntax	1007

Using FIPS Codes and Province Codes	1033
Using Formats for Maps	1035
SAS/GRAPH Map Data Sets Reference Information	1038
Examples	1045
Chapter 36 △ The GOPTIONS Procedure	1075
Overview	1075
Procedure Syntax	1076
Examples	1078
Chapter 37 △ The GPLOT Procedure	1081
Overview	1081
Concepts	1085
Procedure Syntax	1088
Examples	1120
Chapter 38 △ The GPRINT Procedure	1147
Overview	1147
Concepts	1148
Procedure Syntax	1148
Examples	1153
Chapter 39 △ The GPROJECT Procedure	1161
Overview	1161
Concepts	1163
Procedure Syntax	1167
Using the GPROJECT Procedure	1172
Examples	1173
References	1182
Chapter 40 △ The GRADAR Procedure	1183
Overview	1183
Procedure Syntax	1184
Examples	1196
Chapter 41 △ The GREDUCE Procedure	1213
Overview	1213
Concepts	1215
Procedure Syntax	1215
Using the GREDUCE Procedure	1218
Examples	1220
References	1222
Chapter 42 △ The GREMOVE Procedure	1223
Overview	1223
Concepts	1224
Procedure Syntax	1226

Examples 1228

Chapter 43 △ **The GREPLAY Procedure** 1237

Overview 1238

Concepts 1239

Procedure Syntax 1242

Using the GREPLAY Procedure 1264

Examples 1270

Chapter 44 △ **The GSLIDE Procedure** 1277

Overview 1277

Procedure Syntax 1278

Examples 1282

Chapter 45 △ **The GTESTIT Procedure** 1285

Overview 1285

Procedure Syntax 1290

Examples 1291

Chapter 46 △ **The G3D Procedure** 1295

Overview 1295

Concepts 1297

Procedure Syntax 1300

Examples 1314

References 1325

Chapter 47 △ **The G3GRID Procedure** 1327

Overview 1327

Concepts 1329

Procedure Syntax 1331

Examples 1336

References 1346

Chapter 48 △ **The MAPIMPORT Procedure** 1347

Overview 1347

Procedure Syntax 1348

Examples 1349

PART 5 **The Data Step Graphics Interface** 1351

Chapter 49 △ **The DATA Step Graphics Interface** 1353

Overview 1354

Applications of the DATA Step Graphics Interface 1356

Using the DATA Step Graphics Interface 1357

DSGI Graphics Summary 1360

Chapter 50 △ **DATA Step Graphics Interface Dictionary** 1401

Overview	1401
GASK Routines	1404
GDRAW Functions	1446
GRAPH Functions	1457
GSET Functions	1462
Return Codes for DSGI Routines and Functions	1501
See Also	1502
References	1503

PART 6 **Appendixes** **1505**

Appendix 1 **△ Summary of ActiveX and Java Support** **1507**

Introduction	1508
Global Statements	1508
PROC GAREABAR	1518
PROC GBARLINE	1519
PROC GCHART	1521
PROC GCONTOUR	1526
PROC GMAP	1527
PROC GPLOT	1530
PROC GRADAR	1535
PROC G3D	1537
Annotate Functions	1539

Appendix 2 **△ Recommended Reading** **1547**

Recommended Reading	1547
---------------------	------

Glossary **1549**

Index **1561**

What's New

Overview

It's easier than ever to produce enhanced and detailed, informative graphics for your Web presentations. New features in SAS/GRAPH include:

- three new procedures: GAREABAR, GBARLINE, and MAPIMPORT
- new options in the GCHART procedure for producing standard pie or donut charts with a detailed, inner pie overlay
- new options in the GRADAR procedure for adjusting how a chart looks and for specifying annotation
- support for the new DOCUMENT procedure for the Output Delivery System (ODS)
- the new SAS Maps Online application
- server-side rendering using the ACTXIMG and JAVAIMG devices
- client support for annotation
- client support for ODS styles
- ActiveX support for creating graphs interactively
- ActiveX support for radar charts produced with the GRADAR procedure
- the new Java Constellation Applet
- user interface enhancements for the client graphs, such as redesigned dialogs.

Note:

- This section describes the features of SAS/GRAPH that are new or enhanced since SAS 8.2.
- z/OS is the successor to the OS/390 operating system. SAS/GRAPH 9.1 is supported on both OS/390 and z/OS operating systems and, throughout this document, any reference to z/OS also applies to OS/390, unless otherwise stated.

△

Details

Procedures

The following procedures are new or enhanced:

- The GAREABAR procedure generates bar graphs in which the width of the bars are proportional to a characteristic of the data element that is represented by the bars. Area bar charts are supported only when you specify **device=activex** or **device=activeximg**. For more information, see Chapter 27, “The GAREABAR Procedure,” on page 725.
- The new GBARLINE procedure enables you to create vertical bar charts that have line plot overlays.

Note: The GBARLINE procedure is not supported for Java. Δ

For more information, see Chapter 28, “The GBARLINE Procedure,” on page 739.

- The new MAPIMPORT procedure enables you to import ESRI Shapefiles into SAS/GRAPH map data sets. For more information, see Chapter 48, “The MAPIMPORT Procedure,” on page 1347.
- The new DOCUMENT procedure for ODS enables you to re-arrange or to duplicate reports—including graphs—without having to rerun your analysis. You can display output to any ODS output format without executing your SAS programs again. For more information, see the DOCUMENT procedure in *SAS Output Delivery System: User's Guide*.
- New options in the PIE statement in the GCHART procedure enable you to create detail pie charts that have an inner pie overlay. The slices in the overlay align with the slices in the outer pie and show detailed information about the major components that comprise the outer slice. The new options are **DETAIL=**, **DETAIL_PERCENT=**, **DETAIL_RADIUS=**, **DETAIL_SLICE=**, **DETAIL_THRESHOLD=**, and **DETAIL_VALUE=**. For more information, see Chapter 29, “The GCHART Procedure,” on page 773.
- The GRADAR procedure has the following new options:
 - **ANNOTATE=** specifies an annotate data set.
 - **INBORDER** requests a border around plots.
 - **INHEIGHT=** specifies the height in percent screen units of text used inside the frame of the chart.
 - **LAST=** specifies that the spoke that corresponds to the category is displayed to the left of the start angle.
 - **MAXNVERT=** specifies the maximum number of vertices.
 - **MISSING** accepts a missing value as a valid midpoint for the chart variable.
 - **NOFRAME** suppresses the frame that is drawn around the chart by default.
 - **NOZEROREF** turns off the zero reference line when negative values are plotted.
 - **OTHER=** specifies a new category that merges all categories that are not selected because of the **MAXNVERT=** option.
 - **ORDERACROSS=** specifies the display order for the values of the **ACROSS=** variable.
 - **SPIDER** draws lines on a radar chart that connect the spokes instead of the default tick marks. The resulting lines look similar to a “spider web.”

- STARINRADIUS= and STAROUTRADIUS= determine the diameter of the stars.

For more information, see Chapter 40, “The GRADAR Procedure,” on page 1183.

- The LEVELS=ALL option in the GMAP procedure uses a color ramp to assign a continuous color scheme to each response value. For more information, see Chapter 35, “The GMAP Procedure,” on page 995.

SYMBOL Statement

The SINGULAR= option tunes the algorithm that is used to check for singularities.

Graphics Options

The following graphics options are new:

USERINPUT

determines whether user input is enabled for the device. For more information, see “USERINPUT” on page 359.

SWFONTRENDER

specifies the method used to render software fonts. For more information, see “SWFONTRENDER” on page 353.

The Annotate Facility

The following macros are new:

%CENTROID

retrieves the centroids of polygons. For more information, see “%CENTROID Macro” on page 680.

%MAPLABEL

creates an output data set that can be used with the ANNO= option in PROC GMAP. For more information, see “%MAPLABEL Macro” on page 686.

SAS Maps Online Application

The new SAS Maps Online application enables you to download: data updates, sample SAS/GRAPH programs that use the map data sets delivered with SAS/GRAPH, and GIF images of maps. SAS Maps Online is located at support.sas.com/rnd/datavisualization/mapsonline/html

Pop-up Data Tips for Web Graphics

Web graphics now support pop-up data tips . A pop-up data tip is text that is displayed when a user moves the cursor over a portion of a Web graphic. You can add custom data tips to the output of any SAS/GRAPH procedure that supports the HTML= option. For more information, see “Adding Data Tips to Web Presentations” on page 568.

Server-Side Rendering

The ACTXIMG and JAVAIMG devices generate images on the server that match the look of the client graphs. These devices are especially useful when you do not need the

interactivity that is provided by the client graphs. ACTXIMG is only available for Windows. For more information, see “ACTXIMG and JAVAIMG Device Drivers Compared to GIF, JPEG, and PNG Device Drivers” on page 440.

Client Support for Annotation

The Java and ActiveX clients now support annotation through the Output Delivery System (ODS) for the G3D, GBARLINE, GCHART, GCONTOUR, GMAP, GPLOT, and GRADAR procedures. You can specify the ANNOTATE= option in these procedures when you are using the JAVA, JAVAIMG, ACTIVEX, and ACTXIMG device drivers. All annotate functions are available with each device driver (except the FRAME and IMAGE functions, which are available only with ACTIVEX and ACTXIMG). For more information, see “Annotate Functions” on page 1539.

Client Support for ODS Styles

ODS styles now affect both table and client graph output. Sixteen new graph styles provide a consistent look for your entire ODS output, which enhances readability and usability. For more information, see “Using ODS Styles” on page 488.

ActiveX Control

The following are enhancements for the ActiveX Control:

- The ActiveX control now supports creating graphs interactively. You can import data from SAS data sets, Microsoft Excel files, or Microsoft Access files.

Note: To create graphs interactively, you must have Enterprise Guide 2.0 HotFix 11 or higher installed. △

- The ActiveX control now supports radar charts that are produced by using the GRADAR procedure.
- The ActiveX control menus now enable the following additional languages: Chinese, Japanese, Korean, and Russian. (The following languages were also available in SAS 8.2: French, German, Hebrew, Hungarian, Italian, Polish, and Spanish.

Java Constellation Applet and DS2CONST Macro

The new Java Constellation Applet, which you can generate by using the DS2CONST macro, enables you to see the relationships among node link data, such as Web click data, network flow data, and simple affinity data. You can interactively select a set of nodes to see the relationships among the nodes. You can see all of the links coming to the set of nodes or going out of a set of nodes. For more information, see “Creating Constellation Diagrams” on page 513.

Java Treeview Applet and DS2TREE Macro

The new Java Treeview applet, which you can generate by using the DS2TREE macro, shows the parent-child relationships of elements in a hierarchical structure. It provides an optional “fish-eye” distortion that highlights the central area of interest, and enables you to search for, hide, and display element subtrees. A Treeview diagram

is ideal for displaying data such as organizational charts or the hierarchical relationships of the pages of a Web site. For more information, see Chapter 18, “Creating Interactive Treeview Diagrams,” on page 503.

Java Contour Applet

The following are enhancements for the Java Contour Applet:

- a new plot style, Smooth, enables you to display flat (linearly interpolated) planes with no outlines.
- several new parameters. For a complete list, see “Parameter Reference for Java and ActiveX” on page 424.

Java Graph Applet

The following are enhancements in the Java Graph Applet:

- For bar charts, error bars and the CERROR= option (which sets their color) are now enabled. Bars can be labeled by statistics by using the OUTSIDE= and INSIDE= options. Patterns are enabled for 2-D bars, and improved support for the VALUE= option in the AXIS statement is provided.
- For pie charts with group variables, the OTHER=, HTML=, INVISIBLE, and EXPLODE options are available. Data tips can now be displayed for groups. The V=EMPTY option in a PATTERN statement creates hollow pie slices. The LABEL= option enables you to specify font height and color for donut charts.
- Scatter plots now enable the BOX, STD, and HILOC interpolations. For these interpolations, you can use the SYMBOL statement to specify colors, font height and width, line type, point labels, and box width.
- The Graph applet menus now enable the following additional languages: Chinese, Japanese, Korean, and Russian. (The following languages were also available in SAS 8.2: French, German, Hebrew, Hungarian, Italian, Polish, and Spanish.)
- The MENUREMOVE parameter can be used to disable menus and menu options in the applet’s user interface.

User-Defined Formats

The Java and ActiveX devices now support user-defined formats, except for nested user-defined formats. For information about defining formats, see the documentation for the FORMAT procedure in the *Base SAS Procedures Guide*.

Colors

Color names can now be a maximum of 64 characters in length.

Fonts

The following fonts have been added:

- Davidb (Hebraic)
- Fsong (Chinese)
- Hebrewb

- Hei (Chinese)
- Mincho (Japanese)

PAGEFIT Attribute for PostScript

The new PAGEFIT image attribute enables you to adjust how a PostScript image fits on the page. The PAGEFIT attribute replaces the NOFIT attribute. For more information, see “Image Formats for Writing” on page 110.

GTITLE and GFOOTNOTE Options for the ODS Statement

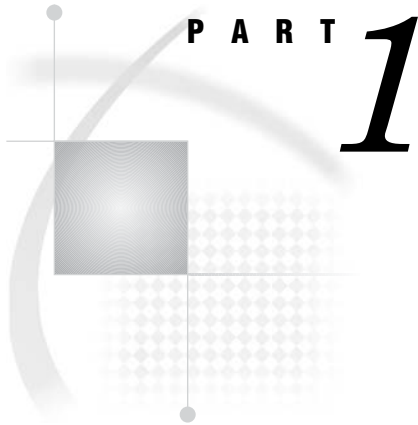
The behavior of the GTITLE and GFOOTNOTE options has been changed—when you specify NOGTITLE or NOGFOOTNOTE, the space in the graphic that would have been used for the title or footnote is allocated to the procedure output rather than being left empty. You need to be aware of this change if you are using annotation or mapping coordinates. For more information, see “Controlling Where Titles and Footnotes are Rendered” on page 492.

Enhancements in SAS/GRAPH Documentation

In addition to information about new features, the SAS/GRAPH documentation now includes information about the following:

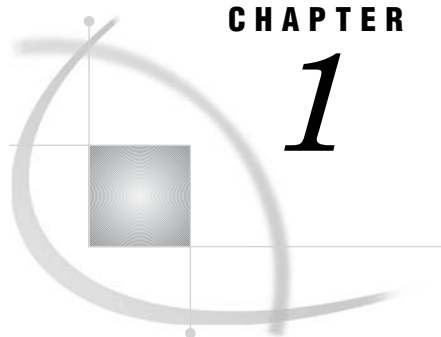
- COLORMAC and HLPCLR macros and expanded information about choosing color schemes
- DSGI routine GASK ('PATREP'), and the functions GSET('PATREP') and GRAPH('PLAY')
- map formats that are available with the GMAP procedure
- HTML generators, which are macros that generate HTML files that run one of the SAS/GRAPH applets: DS2CONST (Constellation Applet), DS2TREE (Treeview Applet), and DS2CSF (Rangeview Applet).
- attributes and parameters for Java and ActiveX, which were formerly documented in the *SAS Output Delivery System: User's Guide*.

Also, additional examples of Web-output programs have been added to the documentation and the SAS/GRAPH sample library. Documentation for the JAVA2 device driver has been removed, because JAVA and JAVA2 are now functionally equivalent.



SAS/GRAPH Concepts

<i>Chapter 1</i>	Introduction to SAS/GRAPH Software	<i>3</i>
<i>Chapter 2</i>	SAS/GRAPH Programs	<i>25</i>
<i>Chapter 3</i>	Device Drivers	<i>41</i>
<i>Chapter 4</i>	SAS/GRAPH Output	<i>47</i>
<i>Chapter 5</i>	SAS/GRAPH Fonts	<i>75</i>
<i>Chapter 6</i>	SAS/GRAPH Colors and Images	<i>91</i>
<i>Chapter 7</i>	SAS/GRAPH Statements	<i>121</i>
<i>Chapter 8</i>	Graphics Options and Device Parameters Dictionary	<i>261</i>



CHAPTER

1

Introduction to SAS/GRAPH Software

<i>Overview</i>	4
<i>Generating Graphs</i>	4
<i>Charts</i>	4
<i>Block charts</i>	4
<i>Horizontal bar charts</i>	5
<i>Vertical bar charts</i>	5
<i>Pie charts, 3-D Pie charts, and Donut charts</i>	6
<i>Star charts</i>	6
<i>Two-Dimensional Plots</i>	6
<i>Two-dimensional scatter plots</i>	7
<i>Simple line plots</i>	7
<i>Regression plots</i>	7
<i>High-low plots</i>	8
<i>Bubble plots</i>	8
<i>Three-Dimensional Plots</i>	9
<i>Surface plots</i>	9
<i>Scatter plots</i>	9
<i>Contour plots</i>	10
<i>Maps</i>	11
<i>Block maps</i>	11
<i>Choropleth maps</i>	11
<i>Prism maps</i>	11
<i>Surface maps</i>	12
<i>Creating Text Slide and Presentation Graphics</i>	12
<i>Text Slides</i>	13
<i>Combining Output into One Slide</i>	13
<i>Enhancing Graphics Output (graphs and text slides)</i>	14
<i>SAS/GRAPH Statements</i>	14
<i>The Annotate Facility</i>	14
<i>Creating Custom Graphics</i>	15
<i>The DATA Step Graphics Interface</i>	15
<i>Graph-N-Go</i>	15
<i>About this Book</i>	16
<i>Audience</i>	16
<i>Prerequisites</i>	16
<i>Conventions Used in This Book</i>	16
<i>Syntax Conventions</i>	17
<i>Conventions for Examples and Output</i>	19
<i>Information You Should Know</i>	20
<i>Support Personnel</i>	20
<i>Sample Programs</i>	21

Overview

SAS/GRAPH software is the data visualization and presentation (graphics) component of the SAS System. As such, SAS/GRAPH software:

- organizes the presentation of your data and visually represents the relationship between data values as two- and three-dimensional graphs, including charts, plots, and maps.
- enhances the appearance of your output by allowing you to select text fonts, colors, patterns, and line styles, and control the size and position of many graphics elements.
- creates presentation graphics. SAS/GRAPH software can create text slides, display several graphs at one time, combine graphs and text in one display, and create automated presentations.
- generates a variety of graphics output that you can display on your screen or in a Web browser, store in catalogs, review, or send to a hardcopy graphics output device such as a laser printer, plotter, or slide camera.
- provides utility procedures and statements to manage the output.

This chapter describes the graphs that are produced by SAS/GRAPH software and explains some of the parts and features of SAS/GRAPH programs.

Generating Graphs

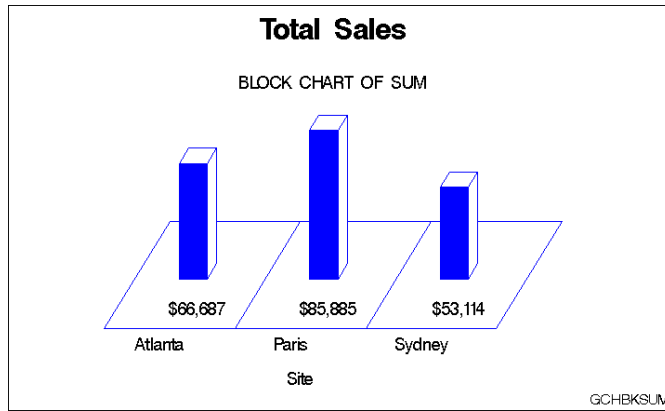
SAS/GRAPH software produces many kinds of charts, plots, and maps in both two- and three-dimensional versions. In addition to helping you understand the variety of graphs that are available to you, these descriptions will also help you choose the correct type of graph for your data and point you to the appropriate chapter.

Charts

SAS/GRAPH software uses the GCHART procedure to produce charts that graphically represent the value of a statistic for one or more variables in a SAS data set. See Chapter 29, “The GCHART Procedure,” on page 773 for a complete description.

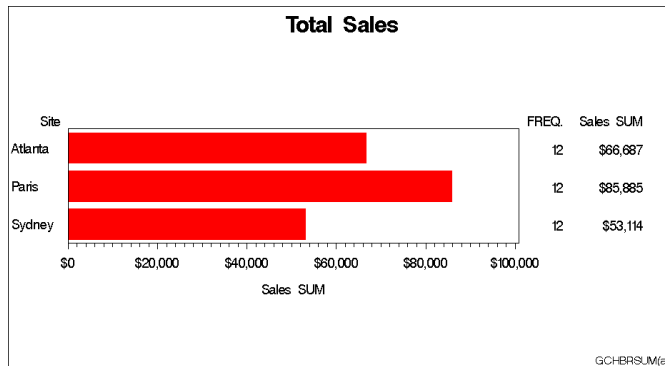
Block charts

Block charts use three-dimensional blocks to graphically represent values of statistics. Block charts are useful for emphasizing relative magnitudes and differences among data values.



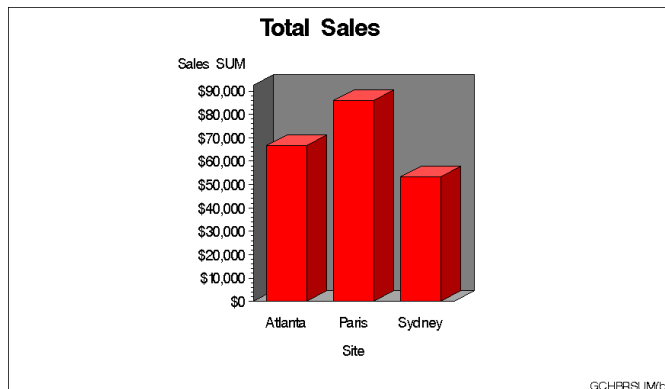
Horizontal bar charts

Horizontal bar charts use horizontal bars to represent statistics based on the values of one or more variables. Horizontal bar charts can generate a table of chart statistics and are useful for displaying exact magnitudes and emphasizing differences.



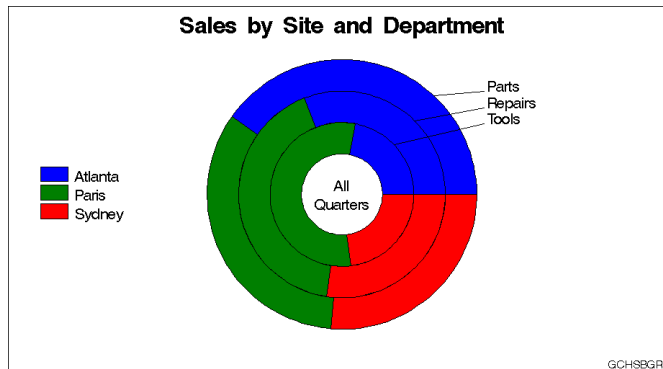
Vertical bar charts

Vertical bar charts use vertical bars to represent statistics based on the values of one or more variables. Vertical bar charts, which generate only one statistic, are useful for displaying exact magnitudes and emphasizing differences.



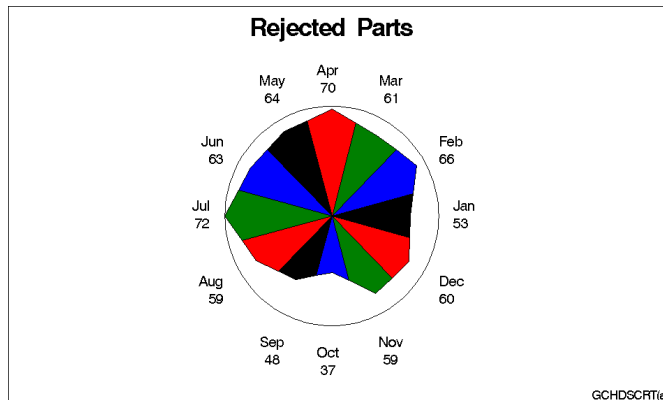
Pie charts, 3-D Pie charts, and Donut charts

Pie charts, 3-D Pie charts, and Donut charts use the angle of pie slices to graphically represent the value of a statistic for a data range. Pie charts are useful for examining how the values of a variable contribute to the whole and for comparing the values of several variables.



Star charts

Star charts use the length of spines to graphically represent the value of a statistic for a data range. Star charts are useful for analyzing where data are out of balance.



Two-Dimensional Plots

SAS/GRAPH software uses the GPLOT procedure to produce two-dimensional graphs that plot one or more dependent variables against an independent variable within a set of coordinate axes. GPLOT can display the data points as individual symbols (as in a scatter plot), or use interpolation methods specified by the SYMBOL statement to join the points, request spline interpolation or regression analysis, produce various high-low plots, or generate several other types of plots.

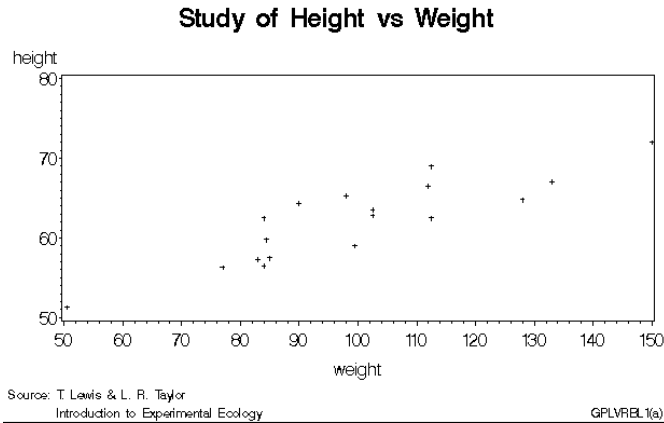
GPLOT can also display data as bubble plots in which circles of different sizes represent the values of a third variable.

Plots are useful for demonstrating the relationship between two or more variables and frequently compare trends or data values or depict movements of data values over time.

See Chapter 37, “The GPLOT Procedure,” on page 1081 for a complete description.

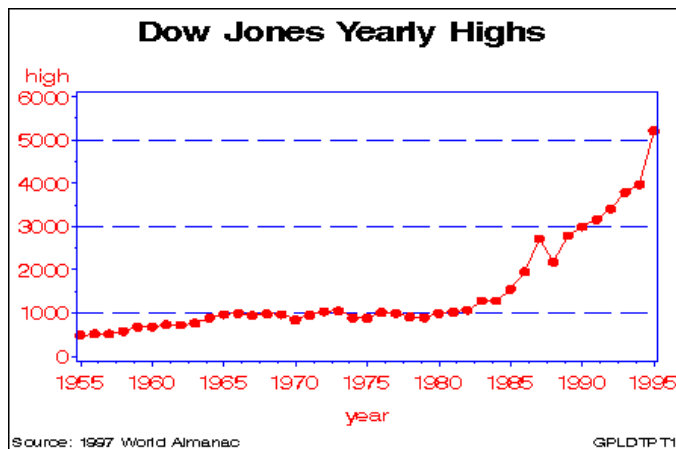
Two-dimensional scatter plots

Two-dimensional scatter plots show the relationship of one variable to another, often revealing concentrations or trends in the data. Typically, each variable value on the horizontal axis can have any number of corresponding values on the vertical axis.



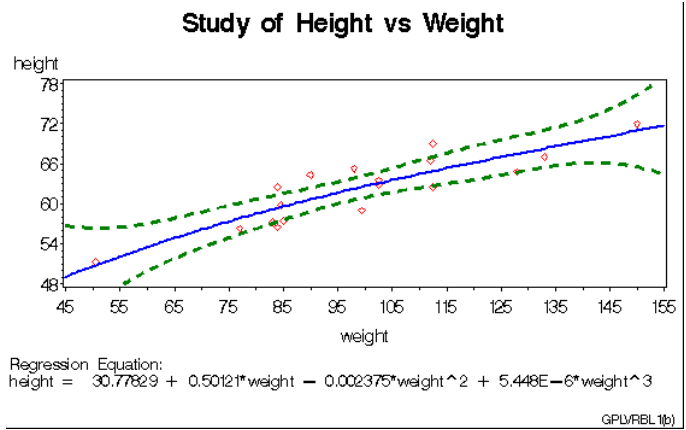
Simple line plots

Simple line plots show the relationship of one variable to another, often as movements or trends in the data over a period of time. Typically, each variable value on the horizontal axis has only one corresponding value on the vertical axis. The line connecting data points can be smoothed using a variety of interpolation methods, including the Lagrange and the cubic spline interpolation methods.



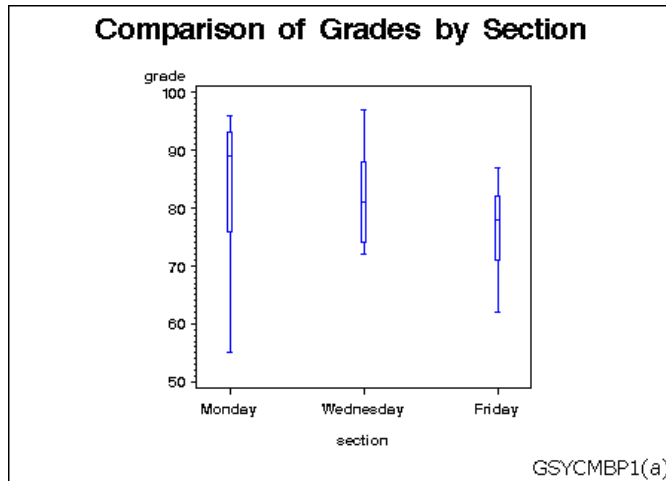
Regression plots

Regression plots specify that the plot is a regression analysis. You can specify one of three types of regression equation – linear, quadratic, or cubic – and optionally display confidence limits for mean predicted values or individual predicted values.



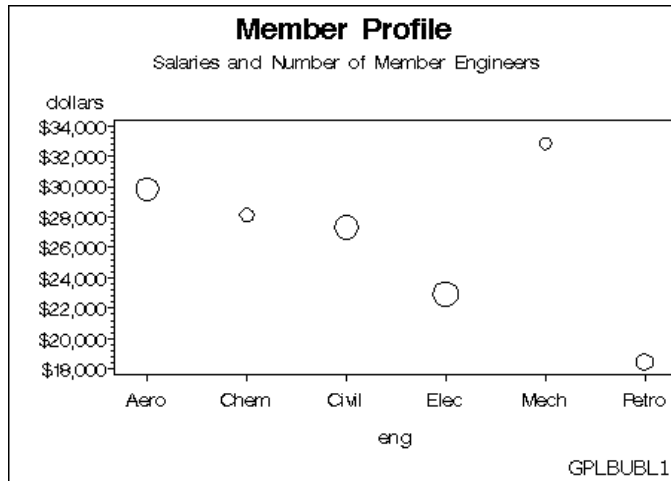
High-low plots

High-low plots show how several values of one variable relate to one value of another variable. Typically, each variable value on the horizontal axis has several corresponding values on the vertical axis. High-low plots include box, needle, and stock market plots.



Bubble plots

Bubble plots show the relative magnitude of one variable in relation to two other variables. The values of two variables determine the position of the bubble on the plot, and the value of a third variable determines the size of the bubble.



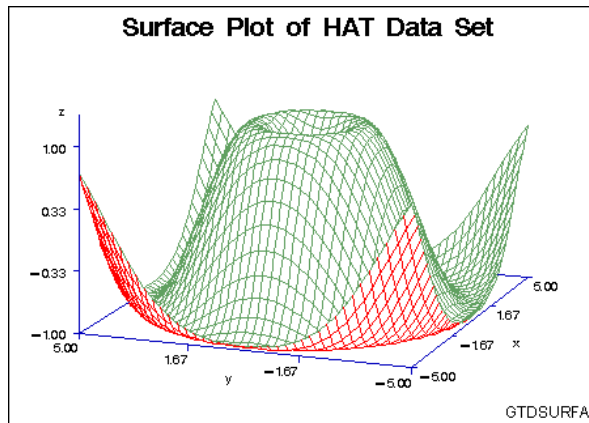
Three-Dimensional Plots

SAS/GRAPH software uses the G3D procedure to produce three-dimensional surface and scatter plots that examine the relationship among three variables. Variable values are plotted on a set of three coordinate axes.

See Chapter 46, “The G3D Procedure,” on page 1295 for a complete description.

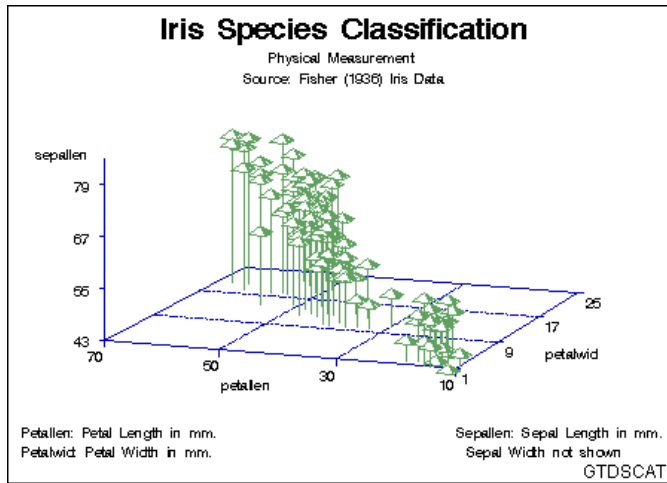
Surface plots

Surface plots are three-dimensional plots that display the relationship of three variables as a continuous surface. Surface plots examine the three-dimensional shape of data.



Scatter plots

Scatter plots enable you to examine three-dimensional data points instead of surfaces and to classify your data using size, color, shape, or a combination of these features.

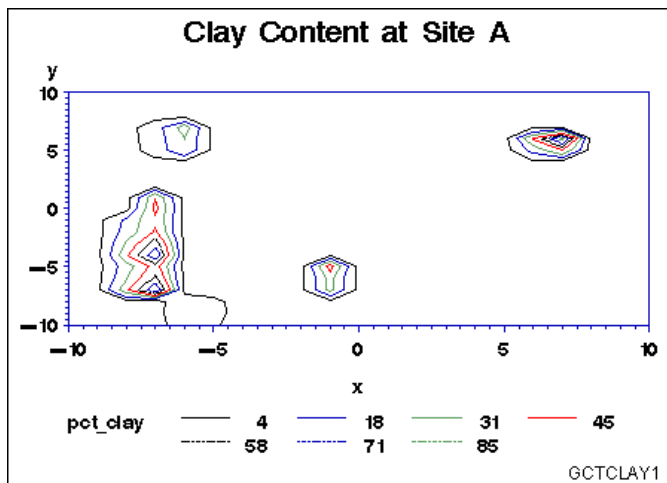


Contour plots

SAS/GRAPH software uses the GCONTOUR procedure to examine three-dimensional data in two dimensions. Lines or areas in a contour plot represent levels of magnitude (z) corresponding to a position on a plane (x,y).

See Chapter 30, “The GCONTOUR Procedure,” on page 885 for a complete description.

Contour plots are two-dimensional plots that show three-dimensional relationships. These plots use contour lines or patterns to represent levels of magnitude of a contour variable plotted on the horizontal and vertical axes.



When you need to interpolate or smooth data values that are used by the G3D and GCONTOUR procedures, use the G3GRID procedure. The G3GRID procedure does not produce graphics output but processes existing data sets to create data sets that the G3D or GCONTOUR procedure can use to produce three-dimensional surface or contour plots. See Chapter 47, “The G3GRID Procedure,” on page 1327 for a complete description.

Maps

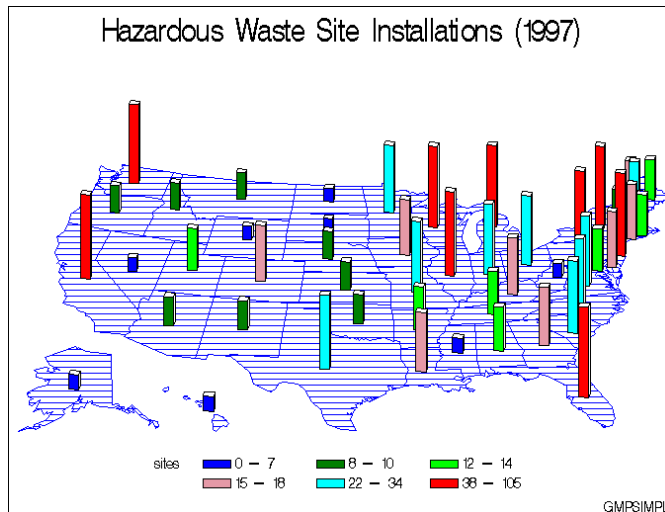
SAS/GRAPH software uses the GMAP procedure to produce two- and three-dimensional maps that can show an area or represent values of response variables for subareas.

SAS/GRAPH software includes data sets to produce geographic maps. In addition, you can create your own map data sets.

See Chapter 35, “The GMAP Procedure,” on page 995 for a complete description.

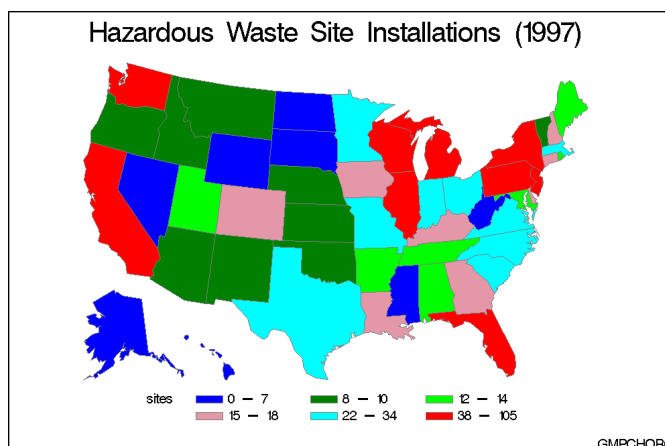
Block maps

Block maps are three-dimensional maps that represent data values as blocks of varying height rising from the middle of the map areas.



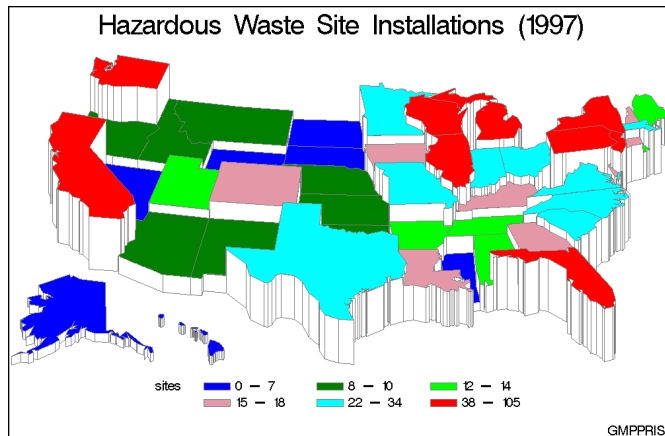
Choropleth maps

Choropleth maps are two-dimensional maps that display data values by filling map areas with combinations of patterns and color that represent the data values.



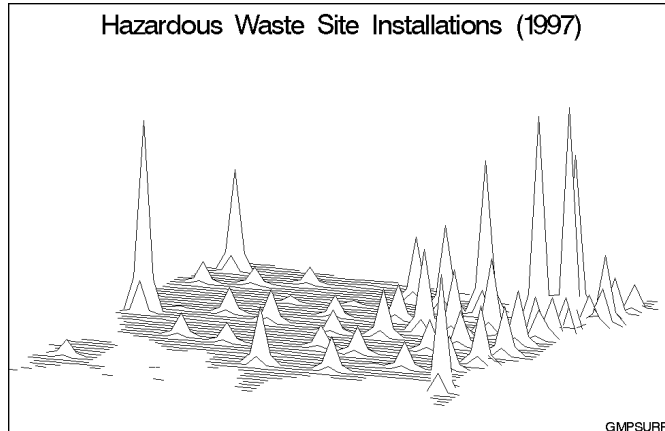
Prism maps

Prism maps are three-dimensional maps that display data by raising the map areas and filling them with combinations of patterns and colors.



Surface maps

Surface maps are three-dimensional maps that represent data values as spikes of varying heights.



SAS/GRAPH software also provides several utility procedures for handling map data.

The GPROJECT procedure lets you choose how geographic maps are projected. This is particularly important for large areas because producing a map of any large area on the Earth involves distorting some areas in the process of projecting the spherical surface of the Earth onto a flat plane. You can use the procedure to select the projection method that least distorts your map.

Map areas are constructed of joined data points. Each data point represents an observation in a SAS data set. For large maps, the amount of data can be prohibitively expensive (in terms of computing resources or time to process); the GREDUCE procedure enables you to reduce the number of points in the data set. The GREMOVE procedure enables you to remove boundary lines within a map.

Creating Text Slide and Presentation Graphics

You can use SAS/GRAPH software to create slide presentations of your graphs. With SAS/GRAPH you can

- create text slides with the GSLIDE and GPRINT procedures
- combine several graphs into one output with the GREPLAY procedure

- automatically or manually replay your graphs and text slides with the GREPLAY procedure.

Text Slides

Use the GSLIDE procedure to create text slides in which you can specify a variety of colors, fonts, sizes, angles, overlays, and other modifications as well as drawing lines and boxes on the output.

See Chapter 44, “The GSLIDE Procedure,” on page 1277 for a complete description.

Text slides display text as graphics output. Text slides can be used as title slides for presentations, or to produce certificates, signs, or other display text.



Use the GPRINT procedure to display as a graphic SAS procedure output that has been saved in a text file. With GPRINT, you bring the text file into SAS/GRAPH and then add titles, notes, and footnotes, and select colors for the output.

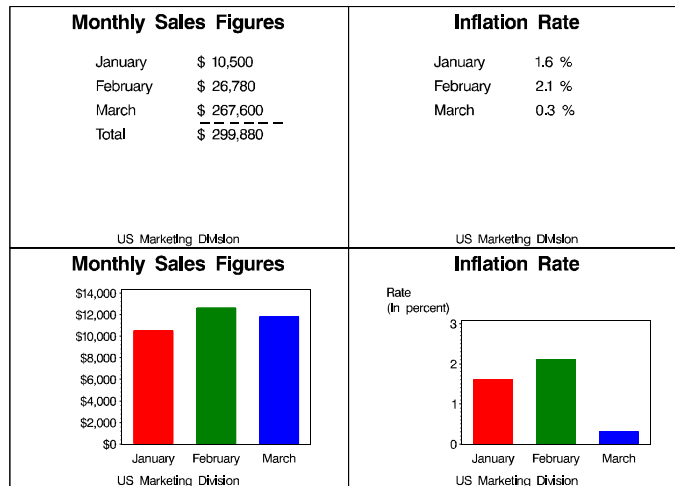
See Chapter 38, “The GPRINT Procedure,” on page 1147 for a complete description.

Combining Output into One Slide

Use the GREPLAY procedure to combine several graphs into a single output. You can create special effects by overlaying or rotating the graphs at any angle.

Templated graphs display two or more graphs or text slides as one output by replaying stored graphs into a template or framework. Like graphs and text slides, templated graphs can be ordered in groups and stored in catalogs for replay as part of a presentation.

Figure 1.1 Templated graphs



In addition, you can use the GREPLAY procedure to create an automated or user-controlled presentation of graphics output. The GREPLAY procedure enables you to name, arrange, and customize the presentation of graphs that are stored in a catalog. See Chapter 43, “The GREPLAY Procedure,” on page 1237 for a complete description.

Enhancing Graphics Output (graphs and text slides)

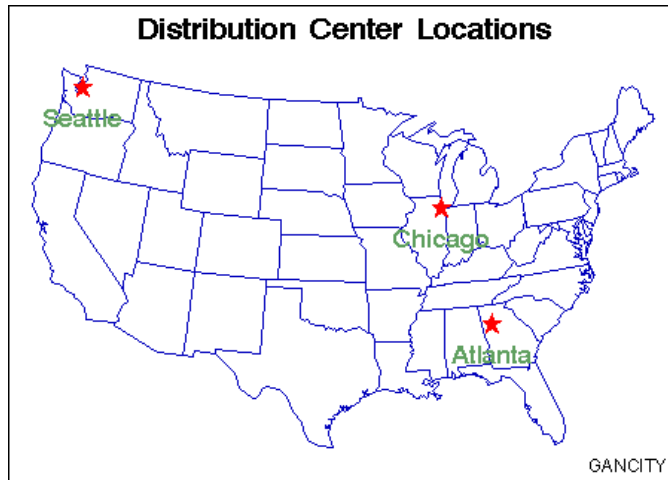
SAS/GRAPH Statements

You also can use *global statements* and *graphics options* in SAS/GRAPH programs. With global statements, you can add titles and footnotes and control the appearance of axes, symbols, patterns, and legends. With graphics options, you can control the appearance of graphics elements by specifying default colors, fill patterns, fonts, text height, and so on.

The Annotate Facility

The Annotate facility enables you to program graphics by using certain variables in SAS data sets. It is often used to add text or special elements to the graphics output of other procedures, although it also can be used to construct custom graphics output. Text and graphics can be placed at coordinates derived from input data, as well as coordinates expressed as explicit locations on the display.

Figure 1.2 Annotated graphs



Creating Custom Graphics

The Annotate facility can also be used to generate custom graphics without using any of the SAS/GRAPH graphing procedures.

The DATA Step Graphics Interface

The DATA Step Graphics Interface provides functions and calls that produce graphics output from the DATA step, rather than from a procedure. The functions and calls are similar in form to those specified by the ISO Graphic Kernal Standard (GKS); however, the interface is not an implementation of the GKS. The form is similar enough that many GKS-compliant programs may be converted easily to run as SAS/GRAPH programs.

Graph-N-Go

To generate presentation graphs without writing any SAS/GRAPH code, you can use Graph-N-Go (not available on mainframes). You can start Graph-N-Go in several ways:

- from the menus in any SAS window, select

Solutions ► Reporting ► Graph-N-Go

- submit either of the following from the SAS command line:

gng

graphngo

- use an Explorer window to directly open a GFORM entry. Double-click (or right-click and choose Open) on a GFORM entry to start a Graph-N-Go session using that entry.

Information on using the application is in Graph-N-Go help, which you can access from the application's main window in either of two ways:

- select

Help ► Using This Window

- press F1 (this may not work in some operating environments).

You can also get help for the application by submitting the following command from the SAS command line:

```
help gng
```

About this Book

This book provides reference information for all facilities, procedures, statements, and options that can be used with SAS/GRAPH software. This chapter describes what you need to know to use SAS/GRAPH software, and what conventions are used in text and example code. To gain full benefit from using this book, you should familiarize yourself with the information presented in this chapter, and refer to it as needed.

Audience

This book is written for users who are experienced in using SAS System software. You should understand the concepts of programming in the SAS language, and you should have an idea of the tasks you want to perform with SAS/GRAPH software.

Prerequisites

The following table summarizes the SAS System concepts that you need to understand in order to use SAS/GRAPH software:

To learn how to	Refer to
invoke the SAS System at your site	instructions provided by the SAS Software Consultant at your site
use base SAS software	<i>SAS Language Reference: Concepts</i> or <i>SAS Language Reference: Dictionary</i>
use the DATA step to create and manipulate SAS data sets	
use the SAS Text Editor to enter and edit text	
allocate SAS data libraries and assign librefs	documentation for using the SAS System under the operating system for the hardware at your site
create external files and assign filerefs	
manipulate SAS data sets using SAS procedures	<i>Base SAS Procedures Guide</i>

Conventions Used in This Book

This section explains the conventions this book uses for text, SAS language syntax, and file and library references. The book uses the following terms in discussing syntax:

keyword is a literal that is a primary part of the SAS language. (A literal must be spelled exactly as shown, although it can be entered in uppercase or lowercase letters.) Keywords in this book are procedure

	names, statement names, macro names, routine names, and function names.
argument	is an element that follows a keyword. It is either literal, or it is user-supplied. It has a built-in value (for example, NODISPLAY), or it has a value assigned to it (for example, COLOR= <i>text-color</i>). Arguments that you must use are <i>required arguments</i> . Other arguments are <i>optional arguments</i> , or simply <i>options</i> .
value	is an element that follows an equal sign. It assigns a value to an argument. It may be a literal, or it may be a user-supplied value.
parameter	is a value assigned to an argument that itself takes a value, for example, the COLOR= parameter of the LABEL= option in a LEGEND statement, as shown in the following statement: <pre>legend label=(color=blue);</pre>

Syntax Conventions

Type styles have special meanings when used in the presentation of SAS/GRAPH software syntax in this book. The following list explains the style conventions for the syntax sections:

UPPERCASE	identifies SAS keywords such as the names of statements and procedures (for example, PROC GCHART). Also identifies arguments and values that are literals, (for example, NOLEGEND and LABEL=NONE).
<i>italic</i>	identifies arguments or values that you supply. Items in italic can represent user-supplied values that are either <ul style="list-style-type: none"> □ nonliteral values assigned to an argument (for example, <i>axis-color</i> in COLOR=<i>axis-color</i>) □ nonliteral arguments (for example, VBAR <i>chart-variable</i>. . . ;). <p>In addition, an item in italics can be the generic name for a list of arguments or parameters from which the user can choose (for example, <i>appearance-options</i>).</p>

The following symbols are used to indicate other syntax conventions:

< > (angle brackets)	identify optional arguments. Any argument not enclosed in angle brackets is required.
(vertical bar)	indicates that you can choose one value from a group. Values separated by bars are mutually exclusive.
. . . (ellipsis)	indicates that the argument following the ellipsis can be repeated any number of times (<i>plot-request</i> <. . . <i>plot-request-n</i> >, for example). If the ellipsis and the following argument are enclosed in angle brackets, they are optional. In SAS/GRAPH software, an ellipsis also indicates a range from which a value is selected (LINE=1 . . . 46, for example).

The following examples illustrate the syntax conventions described in this section. These examples contain selected syntax elements, not complete syntax.

```
PROC GANNO ANNOTATE=Annotate-data-set
<DATASYS>;
```

- PROC GANNO is in uppercase because it is a SAS keyword, the name of a statement. The remaining elements are arguments for the statement.
- ANNOTATE= is not enclosed in angle brackets because it is a required argument. It is in uppercase to indicate that it is a literal and must be spelled as shown.
- *Annotate-data-set* is in italic because it is a value that you must supply; in this case, the value must be a data set name.
- DATASYS is enclosed in angle brackets because it is an optional argument. It is in uppercase to indicate that it is a literal and must be spelled as shown.
- The ending semicolon (;) is required because it is outside the angle brackets for the option.

```
SYMBOL <1 . . . 99>
    <COLOR=symbol-color>
    <MODE=EXCLUDE | INCLUDE>
    <appearance-options>;
```

- SYMBOL is in uppercase because it is a SAS keyword, the name of a statement. The numbers 1 . . . 99 are in angle brackets because they are optional. The ellipsis indicates that you choose one from the range of numbers 1 through 99. The remaining elements are arguments for the statement.
- COLOR= is enclosed in angle brackets because it is an optional argument.
- *Symbol-color* is in italics because it represents a value that you specify.
- MODE= is enclosed in angle brackets because it is an optional argument.
- EXCLUDE and INCLUDE are in uppercase because they are literal values and must be spelled exactly as shown. They are separated by a vertical bar (an *or* bar) because you use one or the other but not both.
- *Appearance-options* is in italics because it is a generic name for a list of options that can be used in the SYMBOL statement.

```
HBAR chart-variable< . . . chart-variable-n>
    </ <PATTERNID=BY | GROUP | MIDPOINT | SUBGROUP>
    <statistic-options>>;
```

- *Chart-variable* is italic because it is an argument that you supply. It is required because it is not in angle brackets.
- *Chart-variable-n* is enclosed in angle brackets because additional user-supplied arguments are optional. The ellipsis before the argument indicates that it can be repeated as many times as desired.
- PATTERNID= is a literal option. The values BY, GROUP, MIDPOINT, and SUBGROUP are literal values that are mutually exclusive. You can use only one, and it must be spelled as shown.
- *Statistic-options* is in italics because it is the generic name of a list of options that affect the chart statistics.

When you are using an option, a statement, or a procedure whose syntax shows arguments or values in italics, you must supply the argument or value. When the argument or value is a font, color, or variable name, SAS/GRAPH software expects valid

font names, color names, and variable names. Consider the following four syntax samples:

```
FONT=font
```

```
COLOR=color
```

```
COLOR=text-color
```

```
PIE chart-variable < . . . chart-variable-n>;
```

- *Font* must be a valid SAS font name. (See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for details.)
- *Color* and *text-color* must be valid SAS/GRAPH colors. (See Chapter 6, “SAS/GRAPH Colors and Images,” on page 91 for details.)
- *Chart-variable* must be a valid SAS variable name. (See *SAS Language Reference: Dictionary* for details.)

Conventions for Examples and Output

Most of the chapters in this book include examples that illustrate some of the features of a procedure or its statements. Each example contains

- a description of the highlights of the example
- the program statements that produce the output
- the actual output from the example
- an explanation of the features of the example.

The output that is shown for the examples in this book was generated in an HP-UX operating environment, using the default XCOLOR display device. If you are using a different operating environment or display device, you may need to make some minor adjustments to the example programs.

The dimensions of the graphics output area vary across devices and when using the GRAPH windows. The dimensions may affect aspects of the graphics output – for example, the appearance of axes or the position of graphics elements that use explicit coordinates in units other than percent. You may need to adjust the dimensions of your graphics output area or the size of graphics elements to correct any differences you see. Most of the examples in this book used a GOPTIONS statement to set the graphics output area to 7 inches by 5 inches, which proved to be a good dimension for generating output displays for this book:

```
goptions hsize=7in vsize=5in;
```

These HSIZE= and VSIZE= settings are not shown in the example code and are not necessary for generating the output, but you may want to use similar settings if your output looks different from the output that is shown in the book.

The examples use explicit color specifications, and the example code contains the names of colors that approximate the colors you see in this book. The colors displayed on your device may differ from those shown in the examples because of differences in device capabilities.

The examples in this book do not specify actual names for the file structures used for SAS data libraries or for external files. This is because different operating environments and different computing installations use different conventions for naming files and directories. Instead, the examples refer to storage locations generically. For example, a LIBNAME statement to assign the libref STORE is shown as

```
libname store 'SAS-data-library';
```

For *SAS-data-library*, you should supply the complete specification for the storage location (for example, directory or z/OS data set) of the data library, using the form required by your operating environment.

Similarly, a FILENAME statement to assign the fileref MYFILE is shown as

```
filename myfile 'external-file';
```

For *external-file*, you should supply a fully qualified filename, using the form required for your operating environment.

If you are unsure of the requirements at your site, see your SAS Software Consultant for more information.

Some examples explicitly specify the font and height for text, the units, and border in procedure statements. For those examples that do not include explicit specifications, the following graphics options were used to produce uniform output:

RESET=GLOBAL	cancels all currently defined AXIS, LEGEND, FOOTNOTE, TITLE, PATTERN, and SYMBOL definitions.
GUNIT=PCT	specifies the unit specification for options is in percent, unless explicitly specified in another SAS statement.
FTEXT=SWISSB	specifies that all text uses the SWISSB font, unless explicitly specified in another SAS statement.
HTITLE=6	specifies that the first title line is 6 percent of the height of the graphics output area, unless explicitly specified in another SAS statement.
HTEXT=3	specifies that text is 3 percent of the height of the graphics output area, unless explicitly specified.
BORDER	causes a border to be drawn around the graphics output area. The border, which appears in most output, represents the boundaries of the graphics output area, whether or not a border is drawn by the program.

Note: The way that output is presented on your device depends on the environment in which you are running SAS/GRAPH software. \triangle

Information You Should Know

This section outlines information you should know before you attempt to run the examples in this book.

Support Personnel

Most sites have personnel available to help users learn to run SAS System software. Record the name of the SAS Software Consultant, SAS Software Representative, and

system administrator at your site. Also record the names of anyone else you regularly turn to for help with running SAS/GRAPH software.

Sample Programs

Most of the chapters in this book provide examples that demonstrate some of the major features of SAS/GRAPH software. To minimize the typing you must do to run the sample code yourself, the code is delivered to you through the SAS Sample Library. Depending on your operating environment, there are up to three ways that you can access the code that is in the sample library:

- if you are viewing the sample code in SAS OnlineDoc, you can copy the code out of the OnlineDoc and paste it into the Program Editor in your SAS session. This alternative is not available if you do not have access to a Web browser in the operating environment where you are running your SAS session.
- in most operating environments (excluding mainframe environments), you can access the sample code through the SAS Help facility. For example, from a SAS window's Help menu, you can choose SAS System Help to enter the help system. You can then choose the link for Sample SAS Programs and Applications, which takes you to the help page for the SAS Sample Library.
- in most operating environments (excluding Windows), the SAS Sample Library may have been installed in your file system. If the SAS Sample Library has been installed at your site, ask your SAS Software Consultant where it is located.

To access the sample programs through SAS System Help or through your file system, you must understand the naming convention used for the samples. The naming convention for SAS/GRAPH samples is *Gpcxxxxx*, where *pc* is the product code and *xxxxx* is an abbreviation of the example title. For example, the code for the first example in the GMAP Procedure chapter, Example 1 on page 1045, is stored in sample member GMPSIMPL. The sample-library member name is sometimes displayed as a footnote in the output's lower-right corner.

- In SAS System Help, the sample programs are organized by product. Within each product category, the samples are sorted alphabetically by title. Thus, to access the code for the first example in the GMAP Procedure chapter, navigate in the help system to the SAS Sample Library page, choose SAS/GRAPH from the list of products, and then scroll to the listing "GMPSIMPL-Producing a Simple Block Map."
- In your file system, the files that contain the sample code have file names that match the sample member names. For example, in a directory-based system, the code for sample member GMPSIMPL is located in a file named GMPSIMPL.SAS.

Note: For WebGraph samples the naming convention is *GWBxxxxx*. △

Note: Some of the examples include LIBNAME and FILENAME statements. You must provide the name of the SAS data library or external file before running the example. △

Table 1.1 Product Codes for SAS/Graph Procedures

Procedure	Code
dsgi	DS
ganno	AN
gchart	CH
gradar	GD
gcontour	CT
gfont	FO
gimport	IP
gkeymap	KY
gmap	MP
goptions	OP
gplot	PL
gprint	PR
gproject	PJ
greduce	RD
gremove	RM
greplay	RE
gslide	SL
gstit	IT
g3d	TD
g3grid	TG

Table 1.2 Product Codes for SAS/Graph Statements

Statement	Code
axis	AX
by	BY
footnote	FO
goptions	ON
legend	LG
note	NO
pattern	PN

Statement	Code
symbol	SY
title	TI

Table 1.3 Product Code for SAS/Graph WebGraphs

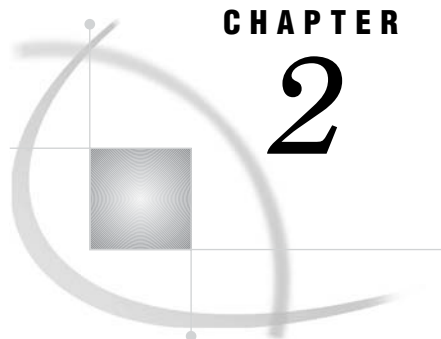
Statement	Code
WebGraph	WB

Map Data Sets

To run the examples that draw maps, you need to know where the map data sets are stored on your system. Depending on your installation, the map data set may automatically be assigned a libref. Ask your SAS Software Consultant or system administrator where the map data sets are stored for your site.

Annotate Macros Data Set

To run the examples using Annotate macros, you need to know where the Annotate macro data set is stored on your system. Depending on your installation, the Annotate macro data set may automatically be assigned a fileref. Ask your SAS Software Consultant or system administrator where the Annotate macro data set is stored for your site.



CHAPTER

2

SAS/GRAPH Programs

<i>Overview</i>	25
<i>Language Elements</i>	26
<i>SAS/GRAPH Procedures</i>	26
<i>SAS/GRAPH Global Statements</i>	27
<i>Annotate DATA Step</i>	27
<i>Other SAS Language Statements</i>	27
<i>FILENAME Statement</i>	28
<i>LIBNAME Statement</i>	29
<i>SAS Data Sets</i>	29
<i>Temporary and Permanent SAS Data Sets</i>	29
<i>Using a Library Reference to Specify a Data Set</i>	30
<i>Using a File Specification to Specify a Data Set</i>	30
<i>Data Set Requirements</i>	31
<i>Automatic Data Set Locking</i>	31
<i>Using Engines with SAS/GRAPH Software</i>	31
<i>Running SAS/GRAPH Programs</i>	31
<i>Modes of Operation</i>	31
<i>Running in Batch Mode</i>	32
<i>RUN-Group Processing</i>	33
<i>With global and local statements</i>	33
<i>With BY statements</i>	33
<i>With the WHERE Statement</i>	33
<i>Procedure Output and the Graphics Output Area</i>	34
<i>About the Graphics Output Area</i>	34
<i>External Dimensions</i>	34
<i>Device Resolution</i>	35
<i>Cells</i>	36
<i>Units</i>	38
<i>Placement of Graphic Elements in the Graphics Output Area</i>	39
<i>How Errors in Sizing Are Handled</i>	40
<i>Making Programs Portable</i>	40

Overview

In general, SAS/GRAPH programs work like other SAS programs: the SAS/GRAPH procedures use data from SAS data sets to produce output – in this case, graphics output. In addition, SAS/GRAPH programs define the output environment and control the format and destination of the graphics output. This chapter discusses SAS/GRAPH programs and explains how they produce graphics output. It describes

- the language elements used by SAS/GRAPH programs

- ways of running SAS/GRAPH programs
- where and how procedure output is produced
- how to control certain attributes of the procedure output.

For information on using and managing SAS/GRAPH output, see Chapter 4, “SAS/GRAPH Output,” on page 47; for information on using SAS/GRAPH programs to create other kinds of graphics output, see “About Exporting SAS/GRAPH Output” on page 59. For information on bringing SAS/GRAPH output to the Web, see Chapter 9, “Introducing SAS/GRAPH Output for the Web,” on page 369.

Language Elements

The language elements used by SAS/GRAPH programs include SAS/GRAPH procedures, SAS/GRAPH statements, and Annotate data sets. In addition to SAS/GRAPH language elements, your SAS/GRAPH program may include Base SAS statements and procedures that you use to process your data or control the destination or format of your program output.

SAS/GRAPH Procedures

SAS/GRAPH procedures create graphics output, process data for other SAS/GRAPH procedures to use, or manage graphics output that has been stored in a catalog. A SAS/GRAPH procedure step typically contains these statements:

PROC statement

starts the procedure. Typically it identifies input and output data sets, and assigns a destination for graphics output. For information on data sets and data requirements, see “SAS Data Sets” on page 29. For information on assigning graphics catalogs, see “Storing Graphics Output in SAS Catalogs” on page 53.

Subordinate statements

perform the work of the procedure; subordinate statements that generate graphs are called *action statements*. For example, the HBAR statement in the GCHART procedure is an action statement.

RUN statement

executes the statements in the procedure step. Use the QUIT statement to end the procedure. See also “RUN-Group Processing” on page 33.

In addition, many SAS/GRAPH procedures can use the following statements:

BY statement

causes the procedure to produce multiple graphs, each corresponding to a BY variable value. Each graph that is produced for a value of a BY variable is stored as a separate catalog entry in either the default catalog, WORK.GSEG, or in the catalog you specify with a GOUT= option in the PROC statement. See “BY Statement” on page 141 for a complete description.

NOTE statement

adds text to the graphics output. See “TITLE, FOOTNOTE, and NOTE Statements” on page 210 for a complete description.

You can also use other SAS language statements with SAS/GRAPH procedures. See “Other SAS Language Statements” on page 27.

SAS/GRAPH Global Statements

SAS/GRAPH has its own set of statements that affect only graphics output that is generated by the SAS/GRAPH procedures and the graphics facilities Annotate and DSGI.

SAS/GRAPH global statements define or modify the titles, footnotes, legends, axes, symbols, and patterns that appear on your graphs, as well as controlling the appearance of the graph, the graphics environment, the destination of the output, and device characteristics.

You can specify these statements anywhere in your program, and they remain in effect until explicitly changed or canceled. These are the SAS/GRAPH global statements:

AXIS

modifies the appearance, position, and range of values of axes in charts and plots.

GOPTIONS

specifies graphics options that control the appearance of graphics elements by specifying characteristics such as default colors, fill patterns, fonts, or text height. Graphics options can also temporarily change device settings.

LEGEND

modifies the appearance and position of legends generated by procedures that produce charts, plots, and maps.

PATTERN

controls the color and fill of patterns that are assigned to areas in charts, maps, and plots.

SYMBOL

specifies the shape and color of plot symbols as well the interpolation method for plot data. It also controls the appearance of lines in contour plots.

TITLE and FOOTNOTE

add titles and footnotes to graphics output.

See Chapter 7, “SAS/GRAPH Statements,” on page 121 for complete descriptions of these statements.

Annotate DATA Step

An Annotate DATA step generates a data set of graphics commands that can be applied to SAS/GRAPH procedure output. See Chapter 24, “Using Annotate Data Sets,” on page 587 for information on building and using Annotate data sets. See Chapter 25, “Annotate Dictionary,” on page 613 for a complete description of all Annotate functions and variables.

Other SAS Language Statements

These SAS language statements can also be used within SAS/GRAPH procedures:

FILENAME statements

identify external files or aggregate file storage locations that you want to use for input or output. See “FILENAME Statement” on page 28 for more information.

FORMAT statement

assigns a format to a variable. SAS/GRAPH procedures use formatted values to determine such aspects of the graph as midpoints, axis labels, tick-mark values, and legend entries.

LABEL statement

assigns a descriptive text string to a variable. Unless other text is specified in the SAS/GRAPH program, the label appears in place of the variable name.

LIBNAME statements

identify SAS libraries that contain SAS data sets or catalogs that you want to use with your SAS/GRAPH programs. See “LIBNAME Statement” on page 29 for more information.

ODS statements

direct the output from certain SAS/GRAPH procedures to the Output Delivery System.

The ODS LISTING statement directs PROC GDEVICE output to the SAS listing file.

The ODS HTML statement is used with the GIF driver to direct graphics output to one or more GIF files and create a variety of HTML files that can display the GIF files in a Web browser. See “ODS HTML Statement” on page 164 for information on using the ODS HTML statement with SAS/GRAPH procedures.

OPTIONS statement

changes the value of one or more SAS system options.

QUIT statement

executes any statements that have not executed and ends the procedure.

WHERE statement

specifies observations from SAS data sets that meet a particular condition. Using a WHERE statement provides an easy way to graph a subset of your data.

For a complete description of these statements, see *SAS Language Reference: Dictionary*.

FILENAME Statement

The FILENAME statement associates a SAS fileref with an external text file or output device. With SAS/GRAPH software, you can use a FILENAME statement to

- point to a text file that you want to use for data input or output.
- assign the destination of a graphics stream file (GSF). This destination can be either a single, specific file or an aggregate file storage location, such as directory or PDS. See “About Exporting SAS/GRAPH Output” on page 59 for information on creating graphics stream files.

You can also use the FILENAME statement to route input to and from other devices. For details, see the SAS documentation for your operating environment.

A FILENAME statement that points to an external file has this general form:

```
FILENAME fileref 'external-file';
```

fileref

is any SAS name.

external-file

is the physical name of the external file or aggregate file storage location you want to reference. For details on specifying the physical names of external files, see the SAS documentation for your operating environment.

For a complete description of the FILENAME statement, see *SAS Language Reference: Dictionary*.

LIBNAME Statement

The LIBNAME statement associates a libref with a SAS data library. A SAS data library can be either temporary or permanent. Typically, SAS data libraries used with SAS/GRAPH software contain

- SAS files for data input and output.
- SAS catalogs that contain maps, fonts, or device entries.
- SAS catalogs that contain graphics output. These catalogs are often stored in permanent libraries. See “Storing Graphics Output in SAS Catalogs” on page 53 for information on storing graphics output in a permanent catalog.

The LIBNAME statement has this general form:

LIBNAME *libref* 'SAS-data-library';

libref

is any SAS name.

SAS-data-library

is the physical name for the SAS data library on your host system. For details on specifying *SAS-data-library*, see the SAS documentation for your operating environment.

The libref WORK is reserved; it always points to an area where temporary data sets and catalogs are kept. The contents of WORK are deleted when you exit a SAS session.

For a complete description of the LIBNAME statement, see *SAS Language Reference: Dictionary*.

SAS Data Sets

Many SAS/GRAPH procedures use SAS data sets as input or output. When a SAS/GRAPH procedure requires an input SAS data set, you usually specify the data set with the DATA= option in the procedure statement, as shown in this example:

```
proc gplot data=reflib.stocks;
```

If you omit the DATA= option, the procedure uses the value of the SAS system option `_LAST_`. The default for `_LAST_` is the most recently created SAS data set (either permanent or temporary) in the current SAS job or session.

If you do not specify a data set and no data set has been created in the current SAS session, an error occurs and the procedure stops.

Most of the procedures that read data sets or create output data sets accept data set options. SAS data set options appear in parentheses after the data set specification, as shown in this example:

```
proc gplot data=reflib.stocks(where=(year=1997));
```

For more information on SAS data sets and other data processing details, see *SAS Language Reference: Concepts*. For a complete discussion of SAS data set options and SAS system options, see *SAS Language Reference: Dictionary*.

Temporary and Permanent SAS Data Sets

SAS data sets are stored in SAS libraries and can be temporary or permanent. You can specify a data set in either of two methods: using a library reference, or using a file specification. A library reference is specified without quotation marks in the form

libref.SAS-data-set-name. A file specification must be enclosed in single quotation marks and uses the file naming conventions of your operating environment.

Using a Library Reference to Specify a Data Set

Typically, temporary SAS data sets are stored in the WORK data library and are referenced with a one-level name. The WORK library is defined automatically at the beginning of the SAS session and is automatically deleted at the end of the SAS session. Procedures assume that SAS data sets that are specified with a one-level name are to be read from, or written to, the WORK data library, unless you specify a USER data library. For example, this statement specifies a temporary data set from the WORK library:

```
proc gplot data=stocks;
```

Typically, permanent SAS data sets have a two-level name of the form *libref.SAS-data-set-name* in which *libref* identifies a storage location on your host system. A LIBNAME statement associates a libref with the storage location. See also “LIBNAME Statement” on page 29. For example, these statements specify a permanent data set:

```
libname reflib 'my-SAS-library';
proc gplot data=reflib.stocks;
```

You can use a one-level name for permanent SAS data sets if you specify a USER data library. In this case, the procedure assumes that data sets with one-level names are in the USER data library instead of in the WORK data library. You can assign a USER data library with a LIBNAME statement or the USER= SAS system option. For example, these statements use a single-level name to specify a permanent data set that is stored in the library identified as the USER library:

```
options user='my-SAS-library';
proc gplot data=stocks;
```

Using a File Specification to Specify a Data Set

To use a file specification for specifying a data set, enclose the file specification in single quotation marks. The specification can be a filename, or a path and filename. The specification must follow the file naming conventions of your operating environment.

For example, the following code creates a file named *mydata* in the default storage location, which is the location where the SAS session was started:

```
data 'mydata';
```

The quotes are required for a file specification; if omitted, SAS treats the specification as a library reference. In the above example, if the quotes are omitted, SAS creates the data set in the temporary WORK catalog and identifies it by the name WORK.MYDATA.

To create the file in a location other than the default location, the quoted file specification must include the full path to the desired location.

You cannot use quoted file specifications for

- SAS catalog names
- MDDDB and FDB references
- PROC SQL
- the `_LAST_` system option.

Data Set Requirements

SAS/GRAPH procedures often have certain requirements for the input data sets they use. Some procedures may expect the input data set to be sorted in a certain way while others may require the data set to contain certain variables or types of information. If necessary, you can use DATA steps and base SAS procedures in your program to manipulate the data appropriately. For specific requirements, see "About the Input Data Set" in the "Concepts" section of the procedure chapter.

Automatic Data Set Locking

All SAS/GRAPH procedures that produce graphics output automatically lock the input data sets during processing. By locking a data set, SAS/GRAPH software prevents another user from updating the data at the same time you are using it to produce a graph. If data in a data set changes while you are using it to draw a graph, unpredictable results can occur in the graph or your program may end with errors.

Using Engines with SAS/GRAPH Software

In the SAS System, procedures use *engines* to access data. Characteristics of these engines vary; generally, they allow SAS procedures to access a data library in a particular way – the expected format for the SAS data file, the type of read/write activity that can occur in SAS data files, and so on. In most cases, you use the default engine for the current SAS version and do not specify an engine. If you are using an engine other than the default, the engine must

- support nonsequential access
- equate observation numbers with internal record IDs (required for the GREduce procedure only)
- disallow shared update or spin a copy of the data set for input processing when a procedure requires multiple passes over the data.

Note: The default engine for Versions 7 and 8 do not work with the GREduce procedure if the input data set is compressed. △

For more information about SAS engines, see *SAS Language Reference: Concepts*.

Running SAS/GRAPH Programs

Modes of Operation

There are several ways to run a SAS program. You can use

- SAS windowing environment that gives you a text editor from which to submit programs, windows for the SAS log and SAS output, and many other facilities
- interactive line mode, in which you submit programs one line at a time
- noninteractive mode, which executes a SAS program (stored in a file) in your current terminal session

- batch mode, which executes a SAS program (stored in a file) in a separate session.

The mode you use determines whether the graphics output displays on your monitor. If you use the SAS windowing environment, interactive line mode, or noninteractive mode, the SAS/GRAPH program can display graphics output on your monitor as well as store the output in a catalog.

If you use batch mode, the graphics output is not displayed on your monitor. In this case, your program must send the graphics output to a hardcopy device, permanent catalog, or a graphics stream file. See Chapter 4, “SAS/GRAPH Output,” on page 47 for more information on the destination of graphics output.

Regardless of how you run your programs, SAS/GRAPH software uses the values stored in the device entry or specified by graphics options in a GOPTIONSChapter 36, “The GOPTIONS Procedure,” on page 1075 statement to determine how to handle the graphics output.

Running in Batch Mode

When you run in batch mode, some SAS/GRAPH device drivers such as device=GIF attempt to use fonts from the X server to annotate the graph output. This can result in a "CANNOT OPEN XDISPLAY" warning if the DISPLAY environment variable has not been set.

To avoid this warning message, you can run in batch mode with the -NOTERMINAL option. Although this suppresses the warning message, it has the disadvantage of not allowing use of the high quality fonts that are available with the X server.

A better solution is to set your DISPLAY environment variable to any available X server. This not only removes warning messages concerning the XDISPLAY, but it also improves the quality of text in GIF output by using fonts available from the X server. The GIF driver does not open any windows on the display referenced by the DISPLAY variable.

An alternative solution does not require a display to be set. FreeType font support can be enabled by setting CHARREC[0] in the device entry to use a TrueType or Type1 font available on your system as follows:

```
libname gdevice0 '.';
proc gdevice c=gdevice0.devices nofs;
  copy gif from=sashelp.devices newname=mygif;
  mod mygif charrec=(0,1,1, 'SAS Monospace', 'Y');
quit;

goptions reset=all dev=mygif ftext='Arial';
title h=5 'Arial';
proc gslide border;run;quit;
```

SAS Monospace is available on all hosts and other fonts can be made available by running PROC FONTREG.

```
proc fontreg fontpath 'directory_containing_TT_or_Type1_fonts';run;
```

FreeType font support is also available with the Universal GIF driver.

```
options dev=sasprtc printerpath=gif;
proc gtestit pic=1;run;
```

RUN-Group Processing

You can use RUN-group processing with the GCHART, GMAP, GPLOT, GREPLAY, and GSLIDE procedures to produce multiple graphs without restarting the procedure every time.

To use RUN-group processing, you start the procedure and then submit multiple RUN-groups. A *RUN-group* is a group of statements that contains at least one action statement and ends with a RUN statement. It can contain other SAS statements such as AXIS, BY, GOPTIONS, LEGEND, TITLE, or WHERE. As long as you do not end the procedure, it remains active and you do not need to resubmit the PROC statement.

To end RUN-group processing, submit a QUIT or RUN CANCEL statement, or start a new procedure.

Note: When using SAS/GRAPH with the ODS statement, it is best to use a QUIT statement after each procedure that uses RUN-group processing, rather than relying on a new procedure to end the processing. Running too many procedures without an intervening QUIT statement can use up so much memory as to crash the system (depending, of course, on how many other processes are running). Also, note that failing to do a QUIT before doing an ODS CLOSE results in the process memory not being freed at all. △

With global and local statements

Global statements and NOTE statements that are submitted in a RUN-group affect all subsequent RUN-groups until you cancel the statements or exit the procedure. For example, each of these two RUN-groups produces a plot and both plots display the title defined in the first RUN-group:

```
/* first run group*/
proc gplot data=sales;
  title1 'Sales Summary';
  plot sales*model_a;
run;

      /* second run group */
      plot sales*model_b;
run;
quit;
```

With BY statements

BY statements persist in exactly the same way. Therefore, if you submit a BY statement within a RUN-group, the BY-group processing produces a separate graph for each value of the BY variable for the RUN-group in which you submit it and for all subsequent RUN-groups until you cancel the BY statement or exit the procedure. Thus, as you submit subsequent action statements, you continue to get multiple graphs (one for each value of the BY variable). For more information, see “BY Statement” on page 141.

With the WHERE Statement

The WHERE statement enables you to graph only a subset of the data in the input data set. If you submit a WHERE statement with a RUN-group, the WHERE definition remains in effect for all subsequent RUN-groups until you exit the procedure or reset the WHERE definition.

Using a WHERE statement with RUN-group processing follows most of the same rules as using the WHERE statement outside of RUN-group processing with these exceptions:

- With the GMAP procedure, the WHERE variable must be in the input data set.
- With a procedure that is using an Annotate data set, the following requirements must be met:
 - The ANNOTATE= option must be included in the action statement.
 - The WHERE statement must be executed before the action statement.
 - The WHERE variable must occur in the Annotate data set.

Procedure Output and the Graphics Output Area

The result of most SAS/GRAPH procedures is the graphic display of data in the form of graphics output. *Graphics output* is made up of commands that tell a graphics device how to draw graphic elements. A *graphics element* is a visual element of graphics output – for example, a plot line, a bar, a footnote, the outline of a map area, or a border.

To generate graphics output, your program uses a device driver that directs the graphics output to a display device (a graphics monitor or terminal), a hardcopy device, or a file. Despite the fact that all graphics devices do not understand the same commands, SAS/GRAPH software can produce graphics output on many types of graphics devices. It does so by producing output in two steps:

- 1 It creates a catalog entry made up of graphics commands in a generic, device-independent format.
- 2 It uses a device driver to translate the commands from the generic format to commands that a particular graphics device understands. This is called device-dependent output.

Your program controls this process as well as the graphics environment in which the graphics appear. This section describes this graphics environment and how you can modify it, how SAS/GRAPH uses it, and how you can make your programs work for different output devices.

About the Graphics Output Area

When SAS/GRAPH software produces graphics output, it draws the graphic elements inside of an area called the *graphics output area*. Characteristics of the graphics output area are determined by the values of certain device parameters within the device entry. You can modify some of these characteristics for a single graph or an entire SAS session by using graphics options to change the values of the device parameters. This section describes changes you can make to the external dimensions, the resolution, the cell size, and the type of units. For a description of the graphics options and device parameters referred to in this section, see Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261.

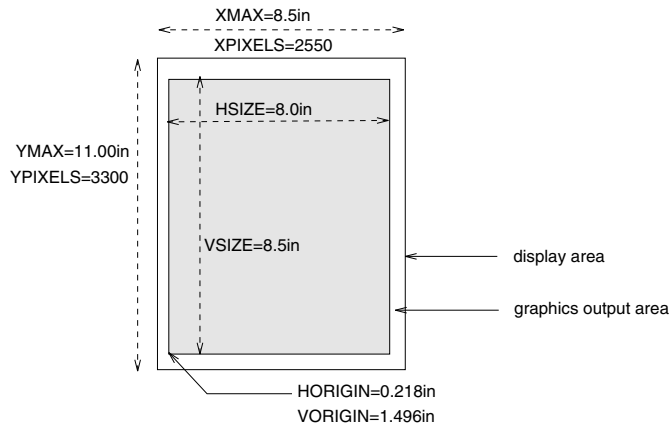
External Dimensions

The graphics output area is contained within the device’s display area. The external dimensions of the device’s display area are controlled by the values of the XMAX and YMAX device parameters. XMAX sets the maximum horizontal dimension; YMAX sets the maximum vertical dimension. The *orientation* of the graphics output area, that is, whether it is landscape or portrait, is determined by whether the larger value is XMAX (orientation is landscape) or YMAX (orientation is portrait).

The external dimensions of the graphics output area are controlled by the values of the HSIZE and VSIZE device parameters.

Typically, the default dimensions of the graphics output area are the same as the dimensions of the device. This is usually true for display devices. For those devices, the default value of HSIZE and VSIZE is 0. However, for hardcopy devices, the XMAX, YMAX values represent the external boundaries of the output medium (such as a sheet of paper). If these devices need a margin, HSIZE, VSIZE and HORIGIN, VORIGIN are assigned default values and the default graphics output area is somewhat smaller than the device's display area. Figure 2.1 on page 35 illustrates such a device.

Figure 2.1 Default Dimensions of the PSCOLOR Device



Note that HORIGIN and VORIGIN define the left margin and bottom margin, respectively. The right margin and top margin are calculated by the device driver as follows:

$$\text{right-margin} = XMAX - (HSIZE + HORIGIN)$$

$$\text{top-margin} = YMAX - (VSIZE + VORIGIN)$$

You cannot specify values for *right-margin* and *top-margin*.

You can change the dimensions of the graphics output area for a SAS session or for a single graph with the HSIZE= and VSIZE= graphics options. Changing the size of the graphics output area does not change the dimensions of the device's display area or affect the resolution. The values of HSIZE= and VSIZE= cannot exceed the maximum dimensions for the device as specified by XMAX and YMAX. Furthermore, you cannot specify values for graphics options HSIZE= and VSIZE= that exceed the HSIZE and VSIZE values in the device entry.

Device Resolution

The resolution of a device is the number of pixels per inch. It is determined by the values of the device parameters XMAX, YMAX, and XPIXELS, YPIXELS, and is calculated by dividing the number of pixels by the corresponding external dimension. For example,

$$x\text{-resolution} = XPIXELS / XMAX$$

Therefore, the X resolution of the PSCOLOR device illustrated in Figure 2.1 on page 35 is 300dpi (dots per inch).

Ordinarily, you do not want to change the device resolution because changing it may distort your image. However, you may want to change the size of the display area. To

do so without changing the resolution, use the GOPTIONS statement to change the values of only XPIXELS= and YPIXELS=, or the values of only XMAX= and YMAX=. Then SAS/GRAPH will automatically calculate the correct value for the unspecified parameters so that the device retains the default resolution.

If you do want to change the device resolution (usually for image files or graphs that are displayed online), specify values for both XMAX= and XPIXELS= (horizontal resolution) or both YMAX= and YPIXELS= (vertical resolution), or all four. In these cases, SAS/GRAPH changes the dimensions and recalculates the device resolution. Table 2.1 on page 36 summarizes the interaction of these options.

Table 2.1 Interactions of Graphics Options That Affect Resolution

<i>If you specify values for</i>	<i>and...</i>	<i>then SAS/GRAPH...</i>
XPIXELS= and YPIXELS=	not XMAX= and YMAX=	changes the dimensions and recalculates the value of XMAX= and YMAX= in order to retain the resolution
XMAX= and YMAX=	not XPIXELS= and YPIXELS=	changes the dimensions and recalculates the value of XPIXELS= and YPIXELS= in order to retain the resolution
XMAX=	XPIXELS=	changes the horizontal dimension and recalculates the resolution
YMAX=	YPIXELS=	changes the vertical dimension and recalculates the resolution

Cells

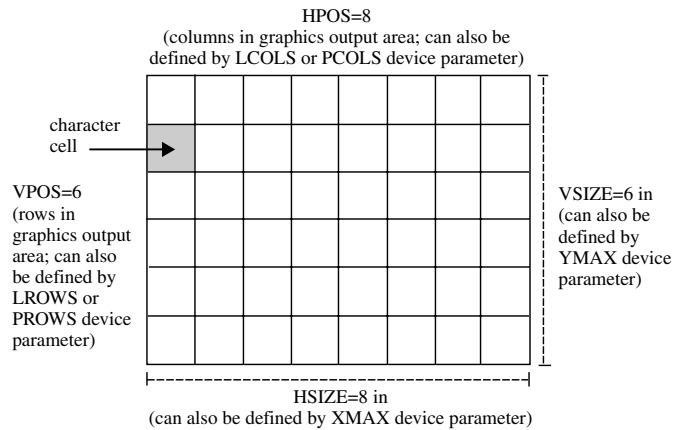
Within the graphics output area, SAS/GRAPH software defines an invisible grid of rows and columns. This grid is made up of *character cells* as shown in Figure 2.2 on page 37.

The size and proportion of these cells affects the size and appearance of graphic elements that are drawn using units of CELLS. The attributes of the cells are determined by both the external dimensions of the graphics output area (controlled by HSIZE and VSIZE) and the number of rows and columns. The number of rows is controlled by the LROWS (if orientation is landscape) or PROWS (if orientation is portrait) device parameter. Similarly, the number of columns is controlled by the LCOLS (landscape) or PCOLS (portrait) device parameter.

You can change the number of rows and columns in the grid with the HPOS= and VPOS= graphics options. HPOS= overrides the value of LCOLS or PCOLS and sets the number of columns in the graphics output area. VPOS= overrides the value of LROWS or PROWS and sets the number of rows in the graphics output area.

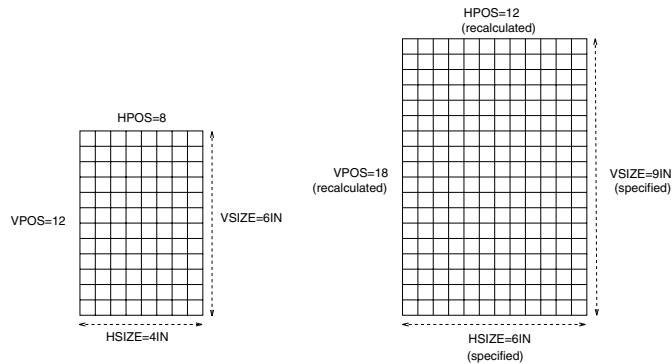
Figure 2.2 on page 37 illustrates the relationship between the graphics options or device parameters that determine the dimensions of the graphics output area and those that determine the number of character cells within the graphics output area.

Figure 2.2 Rows, Columns, and Cells in the Graphics Output Area



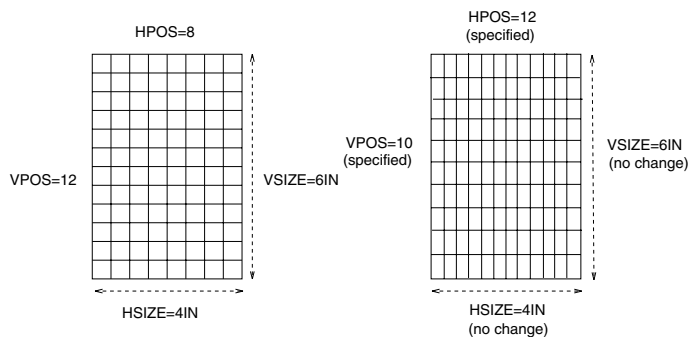
Changing only the external dimensions of the graphics output area (HSIZE= and VSIZE=) retains the cell size but causes SAS/GRAPH to automatically recalculate the number of rows and columns, as illustrated in Figure 2.3 on page 37.

Figure 2.3 Changing HSIZE=, VSIZE= Changes Dimensions and Recalculates Number of Rows and Columns



Changing only the number of rows and columns (HPOS and VPOS) changes the size of the cells without altering the overall size of the output. Figure 2.4 on page 37 shows how increasing the number of rows and columns reduces the size of the individual cells.

Figure 2.4 Changing HPOS= and VPOS= Changes Cell Size



Usually, you should not change the number of rows and columns from the default for your device. However, if you must change them to make a graph fit, note that the size of text in the graphics output will change *if* you specified text size using units of CELLS. If the cells are large (that is, HPOS= and VPOS= have small values), the text may not fit. If the cells are too small, the text may be too small to read. In this case, you can adjust the size of the text with the HEIGHT= statement option or the HTEXT= graphics option.

To change all the attributes of the graphics output area, specify values for all four options, as shown in Figure 2.5 on page 38.

Figure 2.5 Changing HSIZE=, VSIZE= and HPOS=, VPOS= Changes Dimensions and Number and Size of Cells

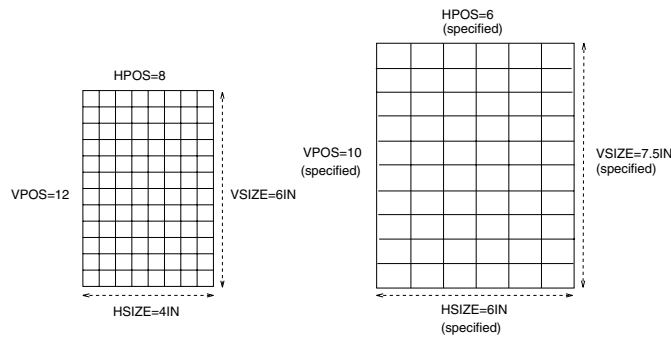


Table 2.2 on page 38 summarizes the interaction of the HSIZE=, VSIZE=, HPOS=, and VPOS= graphics options.

Table 2.2 Interaction of Graphics Options Affecting Cells

<i>If you specify values for ...</i>	<i>and ...</i>	<i>then SAS/GRAPH ...</i>
HSIZE= and VSIZE=	not HPOS= and VPOS= (or specify HPOS=0 and VPOS=0)	changes the external dimensions of the graphics output area and recalculates the number of rows and columns in order to retain cell size and proportions.
HPOS= and VPOS=	not HSIZE= and VSIZE=	keeps the external dimensions but changes the cell size according to the number of rows and columns.
HSIZE= and VSIZE=	HPOS= and VPOS=	changes the dimensions of the graphics output area, the number of rows and columns, and recalculates the cell size.

Units

By default, most graphic elements are drawn using units of CELLS to determine their size. For example, the default character height for the TITLE1 definition is two cells; for all other text the default height is one cell.

Changing the cell size to control the size of one element, such as text, may distort other parts of your graph. Instead, you may want to change the type of units that SAS/GRAPH uses to control the size of the graphic elements. In addition to CELLS and other absolute units such as inches (IN), centimeters (CM), and points (PT), you can often use units of percent of the graphics output area (PCT). This unit specification

allows the height of the graphic elements to change in proportion to the size of the graphics output area.

You can specify the type of unit for individual graphic elements or you can use the GUNIT= graphics option to set the default units that will be used for most height specifications.

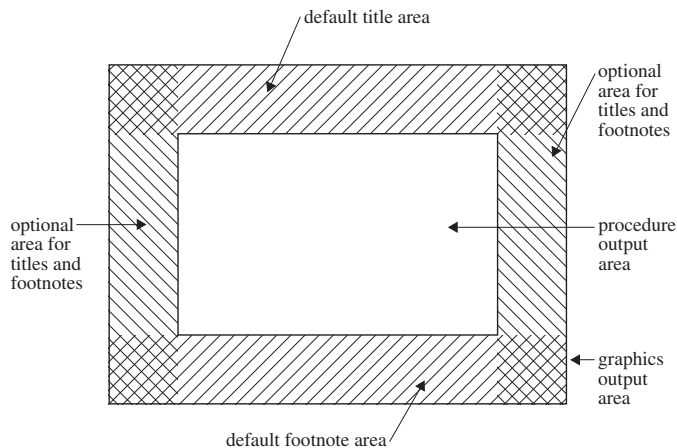
Placement of Graphic Elements in the Graphics Output Area

By default, SAS/GRAPH software positions certain graphics elements in predefined locations in the graphics output area. Figure 2.6 on page 39 shows the graphics output area and the areas within it that are used by the following graphic elements:

- Titles are placed in the title area at the top of the graphics output area.
- Footnotes are placed in the footnote area at the bottom of the graphics output area.
- The graph itself uses the *procedure output area*, which is the area left after the titles and footnotes have been drawn.
- Legends also use the procedure output area and may affect the amount of space available for the graph. By default, space is reserved for the legend below the axis area of a graph and above the footnote area. However, you can position the legend in the part of the procedure output area that is reserved for the graph. For details, see “LEGEND Statement” on page 151.

Note: Titles and footnotes can be positioned elsewhere on the graph as well, with different effects on space allocation. See “TITLE, FOOTNOTE, and NOTE Statements” on page 210 for details. Δ

Figure 2.6 Default Locations for Graphic Elements in the Graphics Output Area



Note: If the titles, footnotes, and legend are very large, they may make the procedure output area too small for the graph. You can control the size of title and footnote text and of most legend elements with statement options. For details, see Chapter 7, “SAS/GRAPH Statements,” on page 121 for a description of the appropriate statement. In addition, “GOPTIONS Statement” on page 146 lists the graphics options that control the size of various graphic elements. See also “Making Programs Portable” on page 40. Δ

How Errors in Sizing Are Handled

Sometimes SAS/GRAPH cannot fit one or more graphic elements on the graph. This can happen if an element is too big for the available space (for example, the title is too long), or if you have too many elements to fit in a given space (for example, a bar chart has too many bars). In these cases, SAS/GRAPH either

- resizes the graphics element and issues a warning explaining what it did
- issues an error message and does not attempt to produce the graph.

For example, it adjusts the size of titles to make them fit but it does not drop bars in order to produce a readable bar chart. If you get unexpected results or no graph, check the SAS log for notes, warnings, and errors.

Making Programs Portable

When you want to write a program that will produce the same graphics output on two different devices, you can use features in SAS/GRAPH software to simplify the process:

- Use percent of the graphics output area (PCT) as the unit of measure when specifying sizes of text and other graphics to make sure that text is proportional in size across devices. A one-inch-high title may be appropriate on a standard piece of paper, but it is almost all of the display area of a slide. To make units of percentage the default for size specifications, use the GUNIT= graphics option:

```
goptions gunit=pct;
```

You can also specify PCT anywhere you specify a size:

```
axis1 label=(height=3 pct 'Year');
```

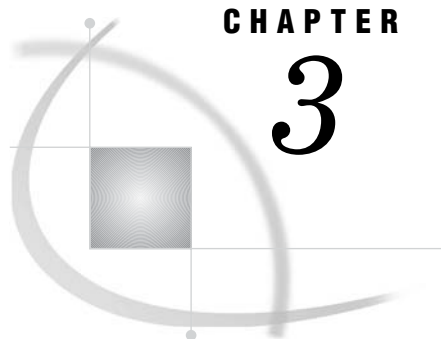
See “GUNIT” on page 309 for a complete description of the GUNIT= graphics option.

- Compare colors and patterns for the devices you will use and use the colors of the device that has the fewest colors. With a slide camera, for example, you can choose from over 16 million colors, but some graphics monitors display only four or eight colors at a time.

If you want to generate a graph on both a color device and a black-and-white device, such as a laser printer or a monochrome graphics monitor, all colors are remapped to black, white, or scales of gray, depending on the device. However, with the PATTERN statement, you can substitute line-patterns for colors.

- Preview the appearance of the output on a different device with the TARGETDEVICE= graphics option. For example, to see how the output will look on a color PostScript printer, specify

```
goptions targetdevice=pscolor;
```



CHAPTER

3

Device Drivers

<i>Overview</i>	41
<i>About Device Drivers</i>	42
<i>Types of Device Drivers</i>	42
<i>How Device Drivers Are Assigned</i>	43
<i>Selecting a Device Driver</i>	43
<i>Viewing the List of Available Device Drivers</i>	44
<i>Browsing the Contents of a Device Entry</i>	44
<i>Specifying a Device Driver in a SAS Session</i>	44
<i>Controlling Output with Device Drivers</i>	45
<i>Modifying Device Parameters Permanently</i>	45
<i>Overriding Device Parameters Temporarily</i>	46
<i>Graphics Options</i>	46
<i>Statement Options</i>	46

Overview

SAS/GRAPH procedures that produce graphics output require a device driver to display the output. This chapter discusses the role of device drivers in generating SAS/GRAPH output, provides directions for selecting and specifying device drivers, and explains how you can change the settings of device parameters.

Other tasks related to device drivers are discussed in Chapter 4, “SAS/GRAPH Output,” on page 47 and Chapter 9, “Introducing SAS/GRAPH Output for the Web,” on page 369 and in the SAS Help facility for SAS/GRAPH. These include

- displaying graphics output (see “Displaying Graphics Output on Monitors or Terminals” on page 49)
- previewing on one device how output will look on another device (see “Previewing Output” on page 52)
- sending graphics output to a printer or other hardcopy device (see “Printing Graphics Output” on page 51)
- creating external files in other graphics formats for use with other applications (see “About Graphics Stream Files” on page 60)
- creating graphics output that can be referenced in an HTML file and viewed with a Web browser (see “About Graphics Stream Files” on page 60)

For a description of device catalogs and for information on creating and modifying device drivers, see Chapter 31, “The GDEVICE Procedure,” on page 915.

About Device Drivers

To produce graphics output on a graphics output device, SAS/GRAPH software uses a device driver. *Device drivers* are the components of SAS/GRAPH software that translate the device-independent output from SAS/GRAPH procedures into the appropriate commands to produce graphics output on particular output devices. Device drivers contain settings that determine the default appearance of the output, such as dimensions and orientation, cell size, colors, and hardware fonts. They can also contain host commands that are issued before and after the driver produces output.

A device driver is composed of two parts:

- a device entry in a SAS catalog
- an executable module.

The *device entry* is a SAS catalog entry of type DEV. It is created and accessed with the GDEVICE procedure and explicitly refers to the executable module. The device entry contains *device parameters* whose settings can control

- the appearance of the output
- the destination to which the output is sent (native SAS/GRAPH drivers only)
- communications between the operating environment and the device
- how the device generates the output.

You can change these settings either by modifying the device parameters using the GDEVICE procedure, or by overriding the parameter settings using graphics options in a GOPTIONS statement. For details, see “Controlling Output with Device Drivers” on page 45.

The *executable module* is a program that produces the device-specific commands from the device-independent output of a SAS/GRAPH procedure. The executable module uses the parameters specified in the device entry to tell it exactly how to do so.

SAS/GRAPH software provides device entries for your operating environment in the Institute-supplied catalog, SASHELP.DEVICES.

If your site has created custom device entries, they may also be stored in SASHELP.DEVICES, although typically custom devices are stored in the catalog GDEVICE0.DEVICES. For more information about custom device entries, see “About Device Catalogs” on page 916 or ask your SAS Support Consultant.

Types of Device Drivers

Most of the device drivers in SASHELP.DEVICES are SAS/GRAPH *native device drivers*, which are those SAS/GRAPH drivers that produce output in the native language of the device. For example, the PS300 driver is a “native device driver” because it directly produces PostScript output.

A special set of *interface drivers* enable you to make route graphics output to the default print device. For OpenVMS, UNIX, and z/OS operating environments, the interface drivers use the Universal Printing subsystem to access a Universal Printer. For Windows, the interface drivers use the Windows Print Manager subsystem to access the printer that is defined as the default.

The interface drivers are:

- | | |
|---------|---------------------|
| SASPRTC | (Color output) |
| SASPRTG | (Grayscale output) |
| SASPRTM | (Monochrome output) |

The WINPRTx and XPRINTx series of drivers are identical to the SASPRTx drivers.

For more information about Universal Printing, see the Base SAS Software section in SAS Help and Documentation. For more information about Windows printing, see the SAS Help facility for SAS/GRAPH.

How Device Drivers Are Assigned

Because many characteristics of the graphics output depend on parameter values that are stored in the device entry, SAS/GRAPH procedures that produce graphics output must know which device driver to use before they begin processing.

Usually SAS/GRAPH automatically selects a device driver for you and you are not required to explicitly specify one. If you use the GRAPH window to display graphics output, SAS/GRAPH selects a device driver that is appropriate for your device. The default device driver for your site may also be selected by your SAS Installation Representative.

If you submit a SAS procedure without specifying a device driver and your display device does not support the GRAPH windows or you are running outside the SAS windowing system, SAS/GRAPH prompts you for a driver name.

Whether or not a default device is assigned, you can always explicitly assign a device driver. See “Selecting a Device Driver” on page 43 for more information.

Selecting a Device Driver

Although SAS/GRAPH software usually selects an appropriate device driver for displaying graphics output on your display device, you may need to select a different device driver if you want to direct your graphics output to another destination.

When you select a device driver, it must be one that is appropriate for your device. The device driver must

- send commands to the device that the device understands. For example, if you are using an X Windows display, the SAS/GRAPH device driver sends the appropriate data stream that can produce graphics output on the display device.
- contain values of device parameters that are appropriate for the device. For example, if you are using a color PostScript printer and you select a device driver for a black and white PostScript printer, your graph will not print in color.

Occasionally the device driver you use is not the one that bears your device’s name. This can happen when

- your graphics device uses a common graphics language (for example, PostScript).
- your graphics device emulates a different graphics device. (For example, there is no SAS device driver for your plotter, but the plotter can be set up to emulate a Hewlett-Packard 7550 plotter.)

In this case, you use the device driver that matches the language that your device understands. For example, if your Hewlett-Packard LaserJet II printer has a PostScript card installed, you would use one of the PostScript device drivers rather than HPLJS2. Similarly, you could use an HP driver for a plotter (for example, HP7550A) when your plotter emulates a Hewlett-Packard 7550 plotter.

You cannot force a device to act as a device with different capabilities by choosing a different device driver.

Viewing the List of Available Device Drivers

You can view the list of device entries in SASHELP.DEVICES or in any other device catalog in the following ways:

- use the SAS Explorer window to display the contents of the device catalog.
- use the GDEVICE procedure to open the GDEVICE DIRECTORY window, which lists all of the device drivers in the current catalog. By default the current catalog is SASHELP.DEVICES. To specify a catalog, include the CATALOG= option, as shown in the following statement:

```
proc gdevice catalog=sashelp.devices;
```

See “Using the GDEVICE Windows” on page 928 for details.

- use GDEVICE procedure statements to write the list of device drivers to the Output window:

```
proc gdevice catalog=sashelp.devices nofs;
  list;
run;
quit;
```

The NOFS on page 921 option in the PROC GDEVICE statement causes the procedure not to use the GDEVICE windows.

If you want to write the list of devices to an external file you can

- save the contents of the Output window.
- use the PRINTTO procedure to redirect the GDEVICE procedure output to an external file. See *Base SAS Procedures Guide* for a description of the PRINTTO procedure.

Once you have generated the list of available device drivers, you can search the list until you find the description that matches your output device. The corresponding name is the name that you specify as the device driver.

Browsing the Contents of a Device Entry

You can also use any of the viewing methods to browse the contents of a device entry. From the GDEVICE Directory window, select the device name to open the GDEVICE Detail window. From there you can move to the other GDEVICE windows for the entry, using either the menus or commands. For details, see “Using the GDEVICE Windows” on page 928.

You can display the contents of a device entry in the Output window by selecting the entry from the Explorer window or by submitting GDEVICE statements. The following statements list in the Output window the contents of the PSCOLOR device entry:

```
proc gdevice c=sashelp.devices nofs;
  list pscolor;
run;
quit;
```

See Output 31.1 for an illustration of the device listing.

Specifying a Device Driver in a SAS Session

You can specify a device driver in these ways:

- use the `DEVICE=` option in a `GOPTIONS` or `OPTIONS` statement. For example,

```
goptions device=pslepf;
```

For details, see “GOPTIONS Statement” on page 146.

- change the device in the System Options window. To do so, type the `OPTIONS` command on the command line, and in the Graphics group, choose "Driver settings." You can then enter a Device value. Use the window's Help button if you need help editing values.
- enter the device name in the `DEVICE` prompt window. The `DEVICE` prompt window opens automatically if you submit a `SAS/GRAPH` program that produces graphics output, no device has been specified, and you are running outside of the SAS windowing system environment.

If you specify a device driver in more than one way, the most recently specified device driver is used. The device driver stays in effect until you specify another device, submit the graphics option `RESET=GOPTIONS` or `RESET=ALL`, or end your SAS session.

If you use the same device driver for most or all of your `SAS/GRAPH` programs, you can put the `GOPTIONS DEVICE=` statement in an `AUTOEXEC` file. See the SAS companion for your operating environment for details on setting up an `AUTOEXEC` file.

You can also specify a device for previewing or printing your output with the `TARGETDEVICE=` graphics option. For details, see “Printing Graphics Output” on page 51.

Controlling Output with Device Drivers

When a `SAS/GRAPH` procedure produces output, it first checks to see what device driver you have specified. It then looks in the device entry for that driver to find the current parameter settings. In general, parameter values control

- the appearance of the graphics output. Device parameters control such aspects as the size of the graphics output, units (such as inches or points) used to draw the output, colors displayed, and text fonts and sizes used.
- how the operating environment communicates with the device. For example, some devices require the graphics commands to be formatted in a specific way or require a particular communications protocol. Others may require that a set of initialization commands precede the graphics commands.
- how the output is produced – that is, how the output is displayed or printed. Some device parameters control the behavior of a hardcopy device, such as the paper feed between graphs as well as the display characteristics, such as orientation.

The parameter values for device entries in `SASHELP.DEVICES` reflect the most common modes of operation of the supported devices. You can control the way the device driver produces output for your device by changing values in the device entry. You can change device parameters either permanently or temporarily.

Modifying Device Parameters Permanently

To change a device parameter permanently, you must make the change in the device entry itself using the `GDEVICE` procedure. The modifications made to a device entry are in effect for all SAS sessions.

The new values that you specify for device parameters must be within the device's capabilities. For example, all devices are limited in the size of the output they can display. Some output devices cannot display color. If you try to increase the size of the

display past the device's capability or if you specify colors for a device that cannot display them, you will get unpredictable results.

Note: If you run SAS/GRAPH software in a multi-user environment, you should not change the device entries in the Institute-supplied catalog, SASHELP.DEVICES, unless you are the system administrator or the SAS Support Consultant. \triangle

If you need to change a device driver in SASHELP.DEVICES, copy it into a personal catalog named DEVICES, and then modify the copy. To use the new device driver, assign the libref GDEVICE0 to the library that contains the modified copy. See "Creating or Modifying Device Entries" on page 934 for details.

Overriding Device Parameters Temporarily

You can temporarily override the settings of device parameters by using graphics options in a GOPTIONS statement or by specifying options in other SAS/GRAPH statements.

Graphics Options

To override device parameter settings with graphics options, simply submit the options in a GOPTIONS statement. For example, the HSIZE= and VSIZE= graphics options control the default size of the graphics output area and override the values of the HSIZE= and VSIZE= device parameters in the current device entry. The following GOPTIONS statement changes the dimensions of the graphics output area:

```
goptions hsize=6in vsize=4in;
```

These new values remain in effect until you change them, use the RESET= graphics option to reset them, or end your SAS session.

The values that you specify for graphics options must be supported by your graphics device. If you use an option that is not supported, SAS/GRAPH software ignores the option. See "GOPTIONS Statement" on page 146 for information about specifying graphics options.

Statement Options

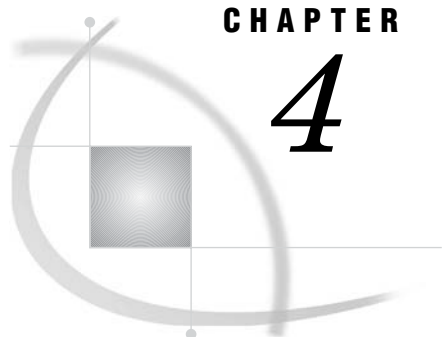
You can also override parameters that control such attributes as text color and font by using options in other SAS/GRAPH statements. For example, this TITLE statement explicitly specifies the text color and font:

```
title color=blue font=zapf 'Production Quality';
```

The COLOR= option overrides the default color selection from the device entry's colors list as well as any graphics options that affect text color. The FONT= option overrides the device's default font and any graphics options that affect the font. As long as the values are supported by your graphics device, the values you specify in the statements override the corresponding device parameters and graphics options when the SAS program is executed.

When you specify options that override device parameter settings, this is the order of precedence that SAS/GRAPH software uses:

- 1 options in a SAS/GRAPH procedure statement or AXIS, FOOTNOTE, LEGEND, NOTE, PATTERN, SYMBOL, or TITLE statement
- 2 graphics options in a GOPTIONS statement
- 3 device parameters in a device entry
- 4 default hardware settings of a device.



CHAPTER

4

SAS/GRAPH Output

<i>About SAS/GRAPH Output</i>	48
<i>What You Can Do With SAS/GRAPH Output</i>	48
<i>Displaying Graphics Output on Monitors or Terminals</i>	49
<i>Displaying Graphs with the GRAPH Window</i>	49
<i>Opening and Closing the GRAPH Window</i>	49
<i>Sizing the GRAPH Window</i>	50
<i>Displaying Graphs with Direct Display</i>	50
<i>Printing Graphics Output</i>	51
<i>Printing Directly to the Device</i>	51
<i>Saving and Printing a Graphics File</i>	51
<i>Printing From a Window</i>	52
<i>Previewing Output</i>	52
<i>Storing Graphics Output in SAS Catalogs</i>	53
<i>Accessing Catalogs from Different Versions of SAS</i>	53
<i>Creating and Specifying Catalogs</i>	54
<i>Names and Descriptions of Catalog Entries</i>	55
<i>Listing and Managing Catalog Entries</i>	55
<i>Modifying SAS/GRAPH Output</i>	55
<i>Transporting and Converting Graphics Output</i>	56
<i>Transporting Catalogs across Operating Environments</i>	56
<i>Example of Transporting GRSEGs</i>	57
<i>Example of Transporting Color Maps and Templates</i>	58
<i>Example of Transporting Fonts</i>	58
<i>Example of Transporting Device Attributes and Device Entries</i>	58
<i>Converting Catalogs to a Different Version of SAS</i>	59
<i>About Exporting SAS/GRAPH Output</i>	59
<i>About Graphics Stream Files</i>	60
<i>About Graphics File Formats</i>	60
<i>Ways to Export SAS/GRAPH Output</i>	61
<i>Exporting SAS/GRAPH Output Interactively</i>	62
<i>Exporting SAS/GRAPH Output with Program Statements</i>	62
<i>General Information</i>	62
<i>Common Requirements</i>	62
<i>Naming the Output</i>	63
<i>Using the NAME= option</i>	63
<i>Using the default output name</i>	63
<i>File extensions</i>	63
<i>Example</i>	63
<i>Saving One Graph to a File</i>	64
<i>Example</i>	65
<i>Saving Multiple Graphs to One File</i>	66

<i>Example</i>	66
<i>Saving Multiple Graphs to Multiple Files</i>	68
<i>Example</i>	68
<i>Replacing Existing External Files</i>	70
<i>Example</i>	71
<i>Other Ways to Assign the Destination</i>	72
<i>Using GACCESS=GSASFILE</i>	72
<i>Using GACCESS= to Explicitly Specify a Destination</i>	72
<i>Exporting SAS/GRAPH Output Using Modified Device Entries</i>	72
73	
<i>The Metagraphics Driver Facility</i>	73

About SAS/GRAPH Output

Most SAS/GRAPH procedures produce *graphics output*, which is distinct from *SAS output*. Whereas SAS output is made up of text, graphics output is made up of commands that tell a graphics device how to draw graphics.

This chapter discusses how to display, print, store, modify, and transport SAS/GRAPH output after you have created it. For information on SAS/GRAPH language elements and programs and on how procedure output is generated, see Chapter 2, “SAS/GRAPH Programs,” on page 25.

What You Can Do With SAS/GRAPH Output

By default, SAS/GRAPH procedures that produce graphics output display the output on your monitor or terminal, using either the GRAPH window or the direct display method. For details, see “Displaying Graphics Output on Monitors or Terminals” on page 49. SAS/GRAPH also can direct graphics output to a variety of other destinations. Graphics output can be

- sent directly to a graphics hardcopy device, such as a printer, plotter, or slide camera. The way you send graphics output to a hardcopy device depends on your hardware, operating environment, and system configuration. For details, see “Printing Graphics Output” on page 51.
- saved in a temporary or permanent SAS catalog entry for later replay. For details, see “Storing Graphics Output in SAS Catalogs” on page 53.
- modified with the graphics editor. You can edit or create graphics output, and save the modified graph to a catalog. For details, see “Modifying SAS/GRAPH Output” on page 55.
- transported in catalogs from one operating environment to another. For details, see “Transporting and Converting Graphics Output” on page 56.
- converted for use with a different version of SAS by converting the catalog containing the graphics output. For details, see “Converting Catalogs to a Different Version of SAS” on page 59.
- exported to external files using different graphics file formats. For example, you can save SAS/GRAPH output in formats such as CGM or PostScript for use with other software packages. For details, see “About Exporting SAS/GRAPH Output” on page 59.

In addition, you can produce graphics output as GIF files and automatically generate HTML files so that you can display your graphics output with a Web browser. For details, see Chapter 9, “Introducing SAS/GRAPH Output for the Web,” on page 369.

Regardless of the other types of output generated, SAS/GRAPH procedures always generate a SAS catalog entry. The entry is stored in the WORK.GSEG catalog unless you specify a different catalog with the GOUT= option in a PROC statement. To generate only catalog entries and suppress all other forms of graphics output, use the NODISPLAY graphics option.

Displaying Graphics Output on Monitors or Terminals

If you want to see your graphics output immediately or preview it before generating a hardcopy, you can send it to your monitor or terminal.

Note: If you are using a terminal or PC that emulates a terminal, it must be a graphics terminal. Δ

In most environments, SAS/GRAPH automatically displays graphics output in the GRAPH window. If your environment does not support the GRAPH window, SAS/GRAPH displays your graphs with the direct display method. See “Displaying Graphs with Direct Display” on page 50.

You can suppress the display of graphics output with the NODISPLAY graphics option. Suppressing a display is useful when you want to create only a catalog entry.

Displaying Graphs with the GRAPH Window

The *GRAPH window*, which is available in the SAS windowing environment, displays catalog entries of type GRSEG. You can use this window to view either the graphics output that you are currently generating or graphics output that has been stored in a catalog. You can scroll backward and forward through the catalog entries.

Some devices allow you to use up to four graph windows: GRAPH1, GRAPH2, GRAPH3, and GRAPH4. By default, the GRAPH1 window displays the graphs in the default catalog, WORK.GSEG, or the catalog that you specified with the GOUT= option.

The GRAPH window acts like other SAS windows: you can resize it and move it, and you can submit global SAS window commands from it. For details on sizing windows, see the SAS documentation for your operating environment. For a description of SAS window commands, see *SAS Language Reference: Dictionary*.

GRAPH window commands control how the graphs appear in the window. For a description of these commands, refer to the SAS Help facility for the GRAPH window.

Opening and Closing the GRAPH Window

The GRAPH Window opens automatically when you submit a procedure that produces graphics output, or when you select a catalog entry from the PROC GREPLAY window or from the SAS Explorer window. You can also open the GRAPH window with the GRAPH n command. The GRAPH n command opens the default catalog WORK.GSEG, or a catalog you specify, or a specific entry.

The GRAPH n command has the following form:

```
GRAPH $n$  <<libref.> catalog-name <.entry-name <.GRSEG>>>
```

n

is a number from 1 to 4 that indicates which GRAPH window to open.

libref

points to the library where the catalog is or will be stored.

catalog-name

is the name of the catalog whose contents you want to view. The default is WORK.GSEG. If the specified catalog does not contain any graphics entries, the window opens but is empty. The catalog assignment is temporary and remains in effect only while the GRAPH window remains open. To change the catalog, resubmit the GRAPH n command.

entry-name

is the name of the catalog entry that you want to view. If you omit *entry-name* or the entry does not exist, the last graph in the catalog is displayed. If you specify *entry-name*, you must also supply the libref and catalog name.

GRSEG

is the type of catalog entry.

To close the GRAPH window, issue the END command.

Sizing the GRAPH Window

The default size of a GRAPH window depends on the display device.

CAUTION:

Resizing the GRAPH window after you have displayed a graph can distort the graphs. If you replay a graph in a GRAPH window that is a different size from the size at which you created the graph, the graph may be distorted. Distortion occurs if the new HSIZE and VSIZE values do not maintain the width-to-height ratio of the original window. \triangle

Doing any of the following may change the HSIZE and VSIZE values and consequently the size of the GRAPH window:

- changing the device driver
- specifying a target device
- specifying dimensions with the HSIZE= and VSIZE= graphics options.

In addition, resizing the GRAPH window can reduce the number of cells available for the output. In order to display some types of graphs in a reduced GRAPH window, you may need to increase the number of cells either by using the HPOS= and VPOS= graphics options or by modifying the device driver. To ensure that graphs have an adequate number of cells, use a full-size GRAPH window when creating the graphs. You can then reduce the window to replay the graphs. For more information on dimensions and cell size, see “About the Graphics Output Area” on page 34.

If you create most or all of your graphs at the same size or with the same aspect ratio, you can avoid distortion if you size the window before you draw the graphs and then use the WSAVE command to save the position and dimensions of the window. If you resize the GRAPH windows and do not use the WSAVE command, the new size is not saved, even in the same SAS session.

Displaying Graphs with Direct Display

Environments that do not support the GRAPH window use the direct display method. With this method, your display is cleared and the graph appears when you run a graphics procedure.

If the procedure produces more than one graph, you are prompted to press ENTER between each graph for the next one to be displayed. To return to your program, press END or ENTER after the last graph.

You can display the graphs automatically and control the amount of time between each graph with the GWAIT= and NOPROMPT graphics options. GWAIT= specifies the

number of seconds before the next graph is drawn. NOPROMPT suppresses the delay between the graphs displayed. In this case, SAS/GRAPH automatically returns to your program after the last graph displays. (See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for a complete description of these graphics options.)

With the direct display method, you cannot scroll to other graphs in the catalog. To redisplay graphs, you must rerun the procedure or replay the catalog entries. For information on replaying graphs, see Chapter 43, “The GREPLAY Procedure,” on page 1237.

Printing Graphics Output

You can print your SAS/GRAPH output on hardcopy devices such as laser printers and plotters. In addition, you can send the output to cameras or film recorders. Regardless of the destination, there are several ways to produce hardcopy of your SAS/GRAPH output:

- Print SAS/GRAPH program output directly to a hardcopy device.
- Print SAS/GRAPH program output by creating an external file of graphics commands called a graphics stream file (GSF), saving it to disk, and printing the file with host commands.
- Print the displayed graph directly from the GRAPH window or the Graphics Editor window.

The following sections provide a general description of these methods.

Operating Environment Information: Whatever way you choose, the exact steps for printing graphics depend on the hardcopy device you are using and on the environment in which you are using it. For complete information on printing graphics output in your operating environment, see the SAS Help facility for SAS/GRAPH and the SAS companion for your operating environment. △

Printing Directly to the Device

You can send graphics output directly to a hardcopy device by sending the graphics commands directly to the device or to a device port.

On most systems you can use any of the following methods to print directly to a device:

- Use a FILENAME statement, a GOPTIONS statement, and a SAS/GRAPH native device driver. The FILENAME statement defines a fileref that points to the print commands. The GOPTIONS statement references the fileref, assigns the device, and specifies any additional parameters.
- Use the GDEVICE procedure to modify a SAS/GRAPH device entry to spool output directly to a printer. See Chapter 31, “The GDEVICE Procedure,” on page 915 for information on adding host commands to a device entry.
- Use the universal printing interface.

For detailed instructions on each of these methods, refer to the SAS Help facility for SAS/GRAPH.

Saving and Printing a Graphics File

There are two steps to printing graphics output from an external file:

- 1 Save your SAS/GRAPH output in an external file. For information on creating external files, see “About Exporting SAS/GRAPH Output” on page 59 and the SAS Help facility for SAS/GRAPH.
- 2 Print the file from your host environment. The host commands vary across operating environments and spooling utilities. See the SAS companion for your operating environment for more information on printing.

You can perform these two steps separately or combine them by incorporating the host printing commands into your program or device driver. In any case, you must choose a graphics file format that is compatible with your hardcopy device. For example, if you are using a PostScript printer, be sure to specify one of the PostScript device drivers supplied in SASHELP.DEVICES. This technique is frequently used on multi-user systems in which the output devices are shared.

You can use any of the following methods to create and print an external file:

- Use the FILENAME and GOPTIONS statements to create the graphics stream file. To route the output directly to the printer, include the print commands in the SAS/GRAPH statements. Otherwise, save the file to disk and use a host command to spool the file to a spooler for the device.
- Use the GDEVICE procedure to modify a SAS/GRAPH device driver to save the output to an external file and spool the output directly to a printer. See Chapter 31, “The GDEVICE Procedure,” on page 915 for information on modifying device entries.
- Use the universal printing interface.

For detailed instructions on each of these methods, refer to the SAS Help facility for SAS/GRAPH.

Printing From a Window

In some operating environments you can print directly from the GRAPH window or the Graphics Editor window by using the PRINT command in conjunction with the TARGETDEVICE= graphics option. To do this, use TARGETDEVICE= to specify the printer driver and use DEVICE= to specify the driver for the terminal or display on which you want to preview the output.

The driver specified by TARGETDEVICE= determines the characteristics of the printed output and sends the graphics output to either an output device or a graphics stream file. The driver specified by DEVICE= controls some characteristics of the output displayed in the window. In some cases, you may need to use a FILENAME and GOPTIONS statements to define the destination of the PRINT command.

Note: If you have not previously specified TARGETDEVICE= when you issue the PRINT command in the window, SAS/GRAPH prompts you for a device for the printed output. However, in this situation the output that is sent to the printer reflects the characteristics of the display device driver. To ensure that your printed output matches what you see on your display, always use TARGETDEVICE= with the PRINT command. \triangle

For details on printing directly from the GRAPH or Graphics Editor window, refer to the SAS Help facility for SAS/GRAPH. See “TARGETDEVICE” on page 355 for a complete description of TARGETDEVICE=.

Previewing Output

If you want to preview how a graph is going to appear on another device before you send it to that device, you can use the TARGETDEVICE= graphics option. For example,

to preview output on your display as it would appear on a color PostScript printer, include TARGETDEVICE= in a GOPTIONS statement and specify the driver for the printer:

```
goptions targetdevice=pcolor;
```

The output is displayed on your screen using

- the orientation of the target device. As a result, the graph may not cover the entire display area of the preview device.
- the values of either the LCOLS and LROWS pair or the PROWS and PCOLS pair, depending on the orientation of the target device.
- the default colors list of the target device.
- the values of the HSIZE and VSIZE device parameters for the target device.
- the value of the CBACK device parameter for the target device.

All other device parameter values, including the destination of the output, come from the current device entry. Therefore, the output displayed by TARGETDEVICE= may not be an exact replication of the actual output, but it is as close as possible.

See “TARGETDEVICE” on page 355 for a complete description of TARGETDEVICE=.

Storing Graphics Output in SAS Catalogs

When you run a SAS/GRAPH procedure that produces graphics output, a copy of the graphics output is always stored in a SAS catalog. A *catalog* is a type of SAS file in which you can store different types of information, called *catalog entries*. Catalog entries of type GRSEG store graphics output. In addition, SAS/GRAPH procedures create other types of catalog entries such as DEV, CMAP, FONT, and TEMPLATE. You can store multiple catalog entries in one catalog.

You can use catalog entries to store graphics output that you want to display again without having to rerun the program. Use the GRAPH window, the GREPLAY procedure, or the SAS Explorer window to redisplay graphics output stored in catalog entries.

SAS catalogs can be either temporary or permanent. *Temporary catalogs* are erased at the end of each SAS session and cannot be retrieved. *Permanent catalogs* are retained until you delete them. Therefore, they exist after the end of a SAS session and can be used in later SAS sessions.

Unless you select another catalog, either temporary or permanent, SAS/GRAPH procedures always store a copy of your graph in the temporary catalog WORK.GSEG, regardless of the other forms of graphics output that you choose. By default, each graph is *appended* to the catalog. The graphs in WORK.GSEG are erased when you end the SAS session.

Accessing Catalogs from Different Versions of SAS

CAUTION:

You can read Version 6 catalogs from Version 7 and Version 8, but you cannot write to them unless you port them. △

In some cases you have to specify an engine to read a Version 6 catalog from the current SAS version:

- If you are running the current version and you assign a libref that points to a library that contains only Version 6 catalogs, the correct engine is automatically

applied and you can view the entries. However, you cannot write to the catalog without porting it.

- If you are running Version 7 or Version 8 and you assign a libref that points to a library that contains both Version 6 and the current version catalogs, the LIBNAME statement must specify the SASEB engine to view the entries in the Version 6 catalog. Even with the engine assigned, you cannot write to the catalog without porting it.

For more information, see “Transporting Catalogs across Operating Environments” on page 56.

All Version 7 and Version 8 applications that support catalog entries that also existed in Version 6 should be able to transparently read those Version 6 catalog entries. That is, if a Version 7 or Version 8 user specifies the SASEB (Version 6 compatibility) engine on the LIBNAME statement, the application should be able to read and process any Version 6 data that the SASEB engine may return.

The Version 7 and Version 8 compatibility engines also support write access to Version 6 catalogs.

Creating and Specifying Catalogs

To create or specify a catalog for graphics output, use the GOUT= option in a PROC statement. The general form is

```
PROC procedure-name GOUT=<libref.> catalog-name;
```

procedure-name

is the graphics procedure you are running.

<libref.>*catalog-name*

is the name of a catalog where you want the output stored. If the specified catalog already exists, the procedure simply adds a catalog entry to the catalog. If the catalog does not exist, it is created.

For a temporary catalog, specify the name of the catalog and omit the libref. (This is a *one-level name*.) The temporary catalog is stored in the WORK library and erased when you end the SAS session.

For a permanent catalog, specify both a libref and a catalog name. (This is a *two-level name*.) *Libref* must already be assigned to a SAS data library that contains or will contain the catalog. For more information on assigning a libref, see “LIBNAME Statement” on page 29.

When you run the procedure, the output is automatically appended to the end of the specified catalog. If the procedure produces multiple graphics, then multiple entries are added to the catalog.

If you want a catalog entry to replace *all* of the existing entries in the catalog, you can use the following statement:

```
goptions goutmode=replace;
```

CAUTION:

Be careful using GOUTMODE=REPLACE. GOUTMODE=REPLACE replaces *all* existing entries in the output catalog. You cannot retrieve previous entries once they have been replaced. See “GOUTMODE” on page 302 for more information. \triangle

Names and Descriptions of Catalog Entries

SAS/GRAPH software always assigns a name and a description to each catalog entry so you can identify it. By default, the names and descriptions are determined by the procedure. For example, a graph produced by the GCHART procedure would be assigned the name GCHART and a description like PIE CHART OF MONTH.

By default, SAS/GRAPH appends each new entry to the catalog. If you create more than one graph with a procedure during a SAS session and the graphs are stored in the same catalog, SAS/GRAPH software adds a number to the end of the name of subsequent catalog entries. This number makes the names unique within the catalog. For example, if you create three graphs with the GCHART procedure during the same SAS session, the catalog entries are named GCHART, GCHART1, and GCHART2. SAS/GRAPH software uses this naming convention whether catalog entries are being stored in a temporary or permanent catalog.

You can supply a name and description when you create the graph by using the NAME= and DESCRIPTION= options. If you create more than one graph of the same name, SAS/GRAPH software increments the specified name just as it does the default names.

Listing and Managing Catalog Entries

You can use the SAS Explorer window or the PROC GREPLAY window to display a list of all of the entries in a catalog. To open the Explorer window for a specific catalog, use the CAT command:

CAT *libref.catalog-name*

To open the PROC GREPLAY window, submit the PROC GREPLAY statement with or without a catalog specification. For details, see Chapter 43, “The GREPLAY Procedure,” on page 1237.

Either method lets you view individual entries once you have displayed the list. For more information, see “Displaying Graphics Output on Monitors or Terminals” on page 49.

There are several ways to manage catalogs and catalog entries:

- The GREPLAY procedure can rename, delete, copy, or group graphics output that is stored in either temporary or permanent catalogs.
- The CATALOG procedure can copy or move an entire catalog or selected entries within a catalog, and can rename or delete catalog entries. For details, see *Base SAS Procedures Guide*.

Modifying SAS/GRAPH Output

The graphics editor is an interactive application that you can use from any GRAPH window to modify graphs produced with SAS/GRAPH software or imported from other graphics software.

You can invoke the graphics editor in several ways:

- To edit a graph that you are viewing in the GRAPH window, select Edit → Edit current graph.
- To open a GRAPH window and invoke the graphics editor without displaying an existing graph, select Tools → Graphics Editor from the SAS Explorer window. After the window appears, you can either open an existing graph or create a graphic image by drawing graphics elements in the window.

- To open a GRAPH window and edit a graph that is stored in a catalog, submit the GRAPH command from the command line of the SAS Explorer window and add EDIT to the end of the command. Separate the two commands with a semicolon (;). For example, the following statement opens the GRAPH3 window, displays the specified graph, and activates the graphics editor:

```
graph3 mylib.grafcat.slide1; edit
```

You can use the graphics editor to change graphics elements already displayed in the graph such as text, lines, and polygons, or you can add graphics elements to the graph. You can create, delete, or duplicate a graphics element and select, change, move, or resize it. You can also change an element's properties, such as its color or fill pattern, or its values, such as the coordinates of one of its points. For instance, you can move a bar from one side of the display area to the other, make a copy of it and place it in yet another place, and change its fill color and pattern. You can undo and redo changes, zoom in or out on the display, move an element to the foreground or background, and read in or link to another graph that has already been created.

After you have made changes, you can save your modifications to a catalog and send the modified graph to an output device, such as a printer.

For a complete description of the graphics editor, refer to the SAS Help facility for SAS/GRAPH.

Note: Modifying your graphics output with the graphics editor does not change the SAS programs that generated that output. Δ

Transporting and Converting Graphics Output

You can use the following methods to transport and convert graphics output within the SAS System:

- Use the CPORT and CIMPORT procedures in base SAS software to transport catalogs that contain graphics output to other operating environments that are running the same version of SAS/GRAPH software.
- Use a LIBNAME statement and the CATALOG procedure to convert catalogs from Version 6 to Version 7 or Version 8.

Transporting Catalogs across Operating Environments

Use the CPORT and CIMPORT procedures to transport catalogs and catalog entries from one machine to another machine running in a different operating environment. In addition to graphics output stored in GRSEG catalog entries, SAS/GRAPH software produces several other files that you can transport from host environment to host environment. These other files include

- colors maps
- templates
- fonts
- device descriptions.

To transport catalog entries that contain graphics output (catalog entries of type GRSEG), follow these steps:

- 1 Use the CPORT procedure to create a transport file from the catalog entries in the current host environment. A transport file is a sequential file that contains the catalog in SAS transport format. To create a transport file, you must specify a catalog to be converted and a fileref for the transport file.

To retain the original order of the GRSEG entries in the catalog, use `SELECT=` in the PROC CPORT statement to export individual graphs in the order they were created. Otherwise, when you use the GREPLAY procedure to list the graphics entries in the imported catalog, the procedure will list the entries in alphabetical order, rather than the order in which they were created.

Note: Only the GREPLAY procedure can list catalog entries in the order they were created. All other procedures list entries in alphabetical order. \triangle

To export a catalog that contains groups of entries created using the GREPLAY procedure, you must use `SELECT=` in the PROC CPORT statement to select the names of the groups, rather than the names of individual graphs, to be included in the transport file. If you export the entire catalog without using `SELECT=`, the groups are not maintained in the catalog created when you import the transport file in the new host environment.

When you use the CPORT procedure, messages in the SAS log identify the catalog entries that have been placed in the transport file. If the catalog entry was created by replaying several graphs into a template, the log messages list the names of all of the entries that contributed to the templated graph.

- 2 Move the transport file to the target machine, if necessary. You must move the transport file in binary format. If you do not move the transport file in binary format, the CIMPORT procedure cannot read the file you create.

Operating Environment Information: Use communications software or tape to move the transport file. Refer to the documentation for your network or standard procedures for using tape files. \triangle

- 3 Once you have moved the transport file to the target machine, import the transport file into a catalog in the new host environment using the CIMPORT procedure. The entries are imported in the order specified in `SELECT=` in the PROC CPORT statement used to create the transport file.

The `SELECT=` option in the PROC CIMPORT statement does not affect the order of the imported entries.

Note: You must use the CIMPORT procedure from the current version of the SAS System. The CIMPORT procedure in a previous release cannot read a transport file created by the CPORT procedure in the current version. For details on using the CPORT and CIMPORT procedures, see the *Base SAS Procedures Guide*. \triangle

Example of Transporting GRSEGs

This example shows how to port three entries from the catalog MYLIB.GRAPHs. First, the CPORT procedure writes selected graphs from MYLIB.GRAPHs to the transport file TRANFILE. The `SELECT` option names the graphs to be ported.

```
libname mylib 'SAS-data-library';
filename tranfile 'external-file';

proc cport file=tranfile
    catalog=mylib.graphs
    select=(GPLOT.GRSEG, GPLOT1.GRSEG, GPLOT3.GRSEG);
run;
```

Once the transport file has been moved to the new host environment using communications software or tape, the CIMPORT procedure creates a new catalog called MYLIB.GRAPHs on the new machine.

```

libname mylib 'SAS-data-library';
filename tranfile 'external-file';

proc cimport catalog=mylib.graphs
    infile=tranfile
    select=(GPLOT.GRSEG, GPLOT1.GRSEG, GPLOT3.GRSEG);
run;

```

Example of Transporting Color Maps and Templates

To transport color maps (catalog entries of type CMAP) and templates (catalog entries of type TEMPLATE) from one host environment to another, use the CPORT and CIMPORT procedures. For example, you could export a color map from the NEWLIB.CMAPS catalog using the following statements:

```

filename tranfile 'external-file';
libname newlib 'SAS-data-library';

proc cport file=tranfile catalog=newlib.cmaps select=(mymap.cmap);
run;

```

After moving the transport file to the new host environment, you can import the color map using the following statements:

```

filename tranfile 'external-file';
libname newlib 'SAS-data-library';

proc cimport infile=tranfile catalog=newlib.cmaps;
run;

```

Example of Transporting Fonts

To transport fonts (catalog entries of type FONT) from one operating system to another, use the CPORT and CIMPORT procedures. For example, you could export a font from the GFONT0.FONTS catalog using the following statements:

```

filename tranfile 'external-file';
libname gfont0 'SAS-data-library';

proc cport file=tranfile
    catalog=gfont0.fonts
    select=(figures.font);
run;

```

After moving the transport file to the new host environment, you can import the font using the following statements:

```

filename tranfile 'external-file';
libname gfont0 'SAS-data-library';

proc cimport infile=tranfile catalog=gfont0.fonts;
run;

```

Example of Transporting Device Attributes and Device Entries

To transport device entries (catalog entries of type DEV) from one operating environment to another, use the CPORT and CIMPORT procedures. For example, you

could export a device entry from the GDEVICE0.DEVICES catalog using the following statements:

```
filename tranfile 'external-file';
libname gdevice0 'SAS-data-library';

proc cport file=tranfile
    catalog=gdevice0.devices
    select=(cgm.dev);
run;
```

After moving the transport file to the new host environment, you can import the device entry using the following statements:

```
filename tranfile 'external-file';
libname gdevice0 'SAS-data-library';

proc cimport infile=tranfile catalog=gdevice0.devices;
run;
```

Converting Catalogs to a Different Version of SAS

To convert catalogs to a different version of SAS, for example from Version 6 to Version 8, use the LIBNAME statement and the CATALOG procedure.

Note: You will not be able to use your old catalogs without transporting them first. △

Before using PROC CATALOG, you must assign librefs to both catalogs and specify the Version 6 Compatibility Engine (saseb) on the input catalog libname. Then use PROC CATALOG with a COPY statement to convert a catalog from Version 6 to Version 7 or Version 8. For details on using the CATALOG procedure, see the *Base SAS Procedures Guide*.

For example, the following statements can be submitted from Version 8 to assign the Version 6 Compatibility Engine and convert a catalog from Version 6 to Version 8.

```
libname v6lib saseb 'SAS-data-library';
libname v8lib 'SAS-data-library';

proc catalog catalog=v6lib.v6cat;
    copy out=v8lib.v8cat;
run;
```

About Exporting SAS/GRAPH Output

By default, SAS/GRAPH output is stored in SAS catalogs as catalog entries of type GRSEG. These entries can be viewed and manipulated within the SAS System and, in some operating environments, can be printed directly as hardcopy. However, you may want to use your SAS/GRAPH output outside of the SAS System. For example, you may want to

- import your graphs into other software packages
- use host system commands or applications to print or manage your graphs
- run batch processes to create and print multiple copies of your graphs
- create graphics output and HTML files that enable you to display SAS/GRAPH output on the Web.

In order to do these kinds of things with your SAS/GRAPH output, you must export your graphs from SAS/GRAPH, using a different graphics file format, such as CGM, GIF, or TIFF, and store them in external files.

For information on creating SAS/GRAPH output for Web publishing, see Chapter 9, “Introducing SAS/GRAPH Output for the Web,” on page 369. For information on SAS/GRAPH language elements and programs and on how procedure output is generated, see Chapter 2, “SAS/GRAPH Programs,” on page 25. For information on using and managing SAS/GRAPH output, see Chapter 4, “SAS/GRAPH Output,” on page 47.

About Graphics Stream Files

When you export SAS/GRAPH output, you run the output through a device driver that creates a *graphics stream file* or GSF. A GSF is an external file that contains graphics commands. Typically, you select a device driver that produces the type of graphics file format that you want, such as CGM, PS or EPS, GIF, or TIFF, although you can select a driver that sends the output directly to a printer or other hardcopy device without creating an external file. You can specify the exact name and location of each file or assign a default location to which all files are sent.

Note: You can also use the Output Delivery System (ODS) or SAS/GRAPH device drivers to generate SAS/GRAPH output as HTML and GIF files that you can view with a Web browser. Details are discussed in Chapter 9, “Introducing SAS/GRAPH Output for the Web,” on page 369. \triangle

Once you have created a GSF, you can

- print the file using host commands
- view the file with an appropriate viewer or browser
- edit the file with the appropriate editing software
- import the file into other software packages.

Note: A GSF is different from a SAS/GRAPH catalog entry. A GSF is an external file that is independent of SAS, and a catalog entry is a type of SAS file. Consequently, you use host commands to manipulate a GSF independent of the SAS System, whereas you must use the SAS System to manipulate SAS catalog entries. For example, to view graphics output stored in a catalog you must use the GREPLAY procedure or the GRAPH window. \triangle

About Graphics File Formats

You can export your SAS/GRAPH output in many different graphics file formats. These are some of the most common formats that SAS/GRAPH software supports:

BMP	Windows bitmap
CGM	Computer graphics metafile
EPS	Adobe’s encapsulated PostScript language
GIF	GIF format
HP-GL	Hewlett Packard’s Graphics Language (plotter control language)
JPEG	JPEG format
PBM	Portable bitmap

PDF	Portable Document Format
PNG	Portable Network Graphics format
PS	Adobe's PostScript language
PPM	Portable pixmap
TIFF	Tagged Image Format File

The type of graphics file format that you choose depends on how you are going to use the output. If you are planning to import the graph into other software products, such as Microsoft Excel or Word Perfect, you may prefer to create a CGM file. These vector-based files are usually smaller than bitmapped files, and they can be edited. In addition, they use hardware fonts and provide a clear image on high-resolution devices.

If you want to display the graph on a Web page, or import it into software that cannot accept vector graphics like CGM, you will need to create a bitmapped file using a format such as GIF or TIFF.

Note: The HTML and WEBFRAME drivers generate both HTML files and GIF files specifically for use with a Web browser. For details, see Chapter 9, "Introducing SAS/GRAPH Output for the Web," on page 369. △

Most software packages that process graphics input can accept one or more of these file formats. Check the documentation for the hardware or software product to which you want to send the graph to determine what file format or formats it can use.

For a complete list of graphics file formats available with SAS/GRAPH in your operating environment, refer to the Device Help for SAS/GRAPH in the SAS Help facility.

Ways to Export SAS/GRAPH Output

There are several ways to send SAS/GRAPH output to an external file in a different graphics file format. You can export graphics output in these ways:

- From the GRAPH window or the Graphics Editor window, use menu selections to select the type of file format and specify a destination for the output that is displayed in the window. When you export SAS/GRAPH output in this way, you are limited to the types of file formats that the Export as Image window supports in your operating environment. For details see "Exporting SAS/GRAPH Output Interactively" on page 62.

Operating Environment Information: In Windows operating environments, WMF formats cannot be exported using this method. △

- Use SAS/GRAPH program statements to direct the output to a graphics stream file. When you use program statements to create a GSF, you explicitly specify a SAS/GRAPH device driver. This driver can be one of the drivers supplied with SAS/GRAPH software and stored in SASHELP.DEVICES, or a custom driver that you have created. For details see "Exporting SAS/GRAPH Output with Program Statements" on page 62.
- Create a custom device driver that contains all of the commands for producing a GSF. For details see "Exporting SAS/GRAPH Output Using Modified Device Entries" on page 72.
- Use the SAS/GRAPH web drivers or the Output Delivery System to create HTML and GIF files. For details see Chapter 9, "Introducing SAS/GRAPH Output for the Web," on page 369.

Exporting SAS/GRAPH Output Interactively

You can export SAS/GRAPH output interactively from either the Graph window or from the Graphics Editor window. To export a graph from one of these windows, follow these steps:

- 1 Open the window and display the graph.
- 2 From the **File** menu select **Export as Image**
- 3 In the Export as Image window, select a file type and specify a destination or file name.
- 4 Close the window by choosing Save or OK (button text depends on the operating environment).

Because you can export only one catalog entry at a time, this method is most useful for quickly exporting a few graphs.

Although you can use this method to create many types of graphics stream files, you have a much larger choice of device drivers when you use SAS/GRAPH program statements to create a GSF.

Exporting SAS/GRAPH Output with Program Statements

When you use program statements to create external files for your SAS/GRAPH output, you use one of these processes:

- create one file that contains one graph
- create one file that contains multiple graphs
- create multiple files that each contain one graph.

You can send the graphics output to external files either at the time you run the program that creates the graphs, or later when you replay them from the catalog in which they are stored. For this reason, these methods are most useful for processing large quantities of output. In addition, using program statements allows you to specify exactly the device driver you want and is therefore a more flexible and powerful way of exporting SAS/GRAPH output.

The following sections provide some information common to all the processes and then describes each process in detail.

General Information

Common Requirements

Regardless of the process you use to create a GSF from a SAS/GRAPH program, you must specify the following:

- a destination for the output. This can be an aggregate file storage location (for example, a directory or a partitioned data set) or a specific file. Typically you specify the destination with a `FILENAME` statement and one or more graphics options. For more information, see “`FILENAME` Statement” on page 28.
- a device driver that creates the type of graphics output that you want.
- whether SAS/GRAPH should replace an existing file or append new records to it. By default, SAS/GRAPH replaces an existing file with newly created output of *the same name*. For details, see “Replacing Existing External Files” on page 70.

Each requirement is explained in detail in the individual process descriptions.

Naming the Output

When you are working with both catalog entries and external files, you should understand how both types of output are named.

Using the NAME= option

You can use the NAME= option in the SAS/GRAPH procedure to specify a name for the catalog entry that the procedure generates. How this name is used depends on whether the FILENAME statement points to a specific external file or to an aggregate file storage location.

- If you specify a specific filename for the external file and also use the NAME= option, the external file is assigned the name specified in the FILENAME statement and NAME= controls only the name given to the created catalog entry. When you specify the filename, you should include the appropriate file extension, such as .CGM, .GIF, or .PS).
- If you specify an aggregate file storage location instead of a specific filename, and also use the NAME= option, the name of the external file is built from the name of the catalog entry, which is determined by the value of NAME=. In this case, SAS/GRAPH supplies the appropriate file extension.

See Table 4.1 on page 64 for examples.

Using the default output name

If you omit NAME=, SAS/GRAPH uses the default naming convention to name the catalog entry, and in some cases the external file. This convention uses up to eight characters of the procedure name as the base name for the catalog entry. If the name generated by the procedure duplicates an existing entry, the name is incremented, for example, GCHART, GCHART1, GCHART2, and so forth. For details, see the description of the NAME= option for a specific procedure.

- If you specify a specific filename for the external file and omit the NAME= option, the external file uses the name specified in the FILENAME statement and the catalog entry uses the default name. When you specify the filename, you should include the appropriate file extension, such as .CGM, .GIF, or .PS.
- If you specify an aggregate file storage location instead of a specific filename, and omit the NAME= option, both the catalog entry and the external file use the default name and SAS/GRAPH supplies the appropriate file extension.

See Table 4.1 on page 64 for examples.

File extensions

When you send SAS/GRAPH output to an aggregate file storage location, SAS/GRAPH generates the name of the external file by taking the catalog entry name and adding the appropriate file extension. Most drivers provide a default extension. If a driver does not generate an extension, SAS/GRAPH uses the default extension .GSF. To specify a different extension from the one SAS/GRAPH provides, use the EXTENSION= graphics option. (For details, see “EXTENSION” on page 288).

Example

illustrates how SAS/GRAPH generates names for catalog entries and external files, depending on 1) whether the NAME= option is used, and 2) on the fileref specification. This illustration assumes the GSLIDE procedure and DEV=GIF:

Table 4.1 How SAS/GRAPH Generates Entry Names and File Names

<i>If...</i>	<i>And...</i>	<i>Then</i>
NAME='FRED'	fileref points to a file named 'MYSLIDE.GIF'	catalog entry name: FRED external file name: MYSLIDE.GIF
NAME='FRED'	fileref points to a storage location (for example, a directory)	catalog entry name: FRED external file name: FRED.GIF
NAME= (not specified)	fileref points to a file named 'MYSLIDE.GIF'	catalog entry name: GSLIDE external file name: MYSLIDE.GIF
NAME= (not specified)	fileref points to a storage location (for example, a directory)	catalog entry name: GSLIDE external file name: GSLIDE.GIF

Note: When the fileref points to an aggregate file storage location, the name of the catalog entry *always* determines the name of the external file. It does not matter whether the catalog entry name is the default name or a name assigned by NAME=. \triangle

CAUTION:

If the graph created by the program already exists in the catalog, a new catalog entry with an incremented name will be created and a new external file may be created rather than updating the existing file. \triangle

You cannot replace individual entries in a catalog; therefore, to replace an entry you must first delete the entry and then re-create it. Therefore, even though the contents of the external file are replaced, the catalog entry is not. Each time you submit the program, a new entry is created and the catalog entry name is incremented.

Saving One Graph to a File

The simplest way to save one graph to a file is to use the FILENAME statement, the GSFNAME= graphics option, and the default setting GSFMODE=REPLACE to create one graphics stream file. These steps describe the general process:

- 1 Use a FILENAME statement to define a fileref for the external file where you want to send the output. The file name must be the complete physical name of the external file and should include a file extension that indicates what type of graphics file you are creating, for example .GIF for a GIF file.
- 2 Assign the fileref to the GSFNAME= graphics option.
- 3 Specify the device driver with the DEVICE= graphics option.
- 4 Use the default setting GSFMODE=REPLACE so that SAS/GRAPH produces only one graph per file (unless BY-group processing is in effect). Because REPLACE is the default setting, you can omit GSFMODE=.
- 5 Submit the SAS/GRAPH program.

Note: The GSF remains open while the SAS/GRAPH procedure is running. Be sure to end the procedure by submitting another procedure step, DATA step, or QUIT statement. To be really safe, you can submit a FILENAME *fileref* CLEAR; statement to explicitly close the GSF.

Operating Environment Information: On certain systems, other graphics options may be required. For more information on creating a graphics stream file, refer to the SAS Help facility for SAS/GRAPH Device Drivers for your operating environment. Δ

Δ

Example

This example creates one GSF that contains one text slide created by a group of TITLE and FOOTNOTE statements and the GSLIDE procedure.

Define the fileref. The FILENAME statement associates the fileref GRAFOUT with the external file that is the destination for the GSF. The file extension .PS indicates that the graphics output is PostScript.

```
filename grafout 'mygraph.ps';
```

Specify graphics options for the GSF. RESET=ALL resets all global statements and graphics options. DEVICE= specifies a PostScript device driver. GSFNAME= assigns the fileref GRAFOUT as the destination for the GSF. GSFMODE=REPLACE (the default) causes the contents of the external file to be replaced each time the graphics procedure is run.

```
goptions reset=all
        device=pscolor
        gsfname=grafout
        gsfmode=replace
        ftext=swissb;
```

Produce one text slide. NAME= specifies the name that is assigned to the catalog entry created by the procedure. If you omit NAME=, SAS/GRAPH uses the default naming convention to name the entry.

```
proc gslide border name='proposal';
  title1 h=4 'Proposed Design Improvements: ';
  title2 h=3 '* Increase Stability';
  title3 h=3 '* Increase Speed';
  title4 h=3 '* Reduce Weight';
  footnote h=2 j=1 'ABC Company';
run;
quit;
```

When you submit these statements, SAS/GRAPH does the following if no graphs of the same name exist in the catalog:

- Creates one catalog entry named PROPOSAL in WORK.GSEG.
- Creates one external file that contains the output from the GSLIDE procedure and sends a message to the LOG reporting the number of records and the name of the file to which they were written. The file name is the one specified in the FILENAME statement.

Because the destination is a specific file and because GSFMODE=REPLACE, each time you run the program it replaces the contents of the external file. Therefore, this method is particularly useful when you want to update an external file by resubmitting an existing program.

However, if there is more than one run of a graphics procedure in this program, the file would contain only the graphics output from the last procedure run because this program replaces the external file each time a graphics procedure is run.

Note: Even though the contents of the external file are replaced, the catalog entry is not. Unless you explicitly delete the existing entry named PROPOSAL, each time you

submit the program, a new entry is created and the catalog entry name is incremented. This table illustrates what happens if you submit the above program three times without deleting the catalog entries: \triangle

Table 4.2

Pass	Catalog Entries	File Name
1	PROPOSAL	mygraph.ps
2	PROPOSA1	mygraph.ps
3	PROPOSA2	mygraph.ps

Note that each new catalog entry replaces the contents of the external file, in this case, mygraph.ps. For more information, see “Replacing Existing External Files” on page 70.

For a complete description of the graphics options used in this example, see Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261.

Saving Multiple Graphs to One File

If your program creates multiple graphs that you want to store in one file, you follow the same steps as those for saving one graph to one file except you specify `GSFMODE=APPEND` to add each new graph to the end of the file instead of replacing the file.

Example

This example stores several text slides in one external file. The program is the same as the previous example except the `GOPTIONS` statement specifies `GSFMODE=APPEND` and the `GSLIDE` procedure uses `RUN-group` processing to create multiple slides. Each slide includes the current `TITLE` statement and all previously defined `TITLE` and `FOOTNOTE` statements.

Define the fileref.

```
filename grafout 'mygraph.ps';
```

Specify graphics options for the GSF. `GSFNAME=` assigns the fileref `GRAFOUT` as the destination for the GSF. `GSFMODE=APPEND` adds each new piece of output to the end of the external file.

```
goptions reset=all
        device=pscolor
        gsfname=grafout
        gsfmode=append
        ftext=swissb
        rotate=landscape;
```

Produce four text slides. Each `RUN-group` generates a new catalog entry. `NAME=` specifies the base name for each catalog entry created by the procedure.

```
proc gslide border name='proposal';
    footnote h=2 j=1 'ABC Company';
    title1 h=4 'Proposed Design Improvements:';
```



```

run;
  title1 h=3 '* Increase Stability';
run;
  title1 h=3 '* Increase Speed';
run;
  title1 h=3 '* Reduce Weight';
run;
quit;

```

When you submit these statements, SAS/GRAPH does the following if no graphs of the same name exist in the catalog:

- Creates a new catalog entry for each graph, beginning with the name PROPOSAL. After the first graph is created, the entry name is incremented so that subsequent graphs are called PROPOSA1, PROPOSA2, and so forth.
- Creates one external file that contains all of the output from the GSLIDE procedure. The file name is the one specified in the FILENAME statement. Each time a graph is added to the file, SAS/GRAPH sends a message to the LOG reporting the number of records and the name of the file to which they were appended.

Note: Because the destination is a specific file and because the GSFMODE= setting is APPEND, each time you run the program SAS/GRAPH adds the new graphs to the external file. If you want the file to contain only the newly created graphs, delete it before resubmitting the program. Δ

In addition, if you resubmit the program without deleting the original catalog entries, SAS/GRAPH will create four new entries that will be added to the catalog entry and four new graphs appended to the external file, both of which will contain all eight graphs.

This table illustrates what happens if you submit this program twice without deleting the catalog entries or the external file:

Table 4.3

Pass	Catalog Entries	File Name	File Contents
1	PROPOSAL PROPOSA1 PROPOSA2 PROPOSA3	mygraph.ps	PROPOSAL, PROPOSA1, PROPOSA2, PROPOSA3
2	PROPOSAL PROPOSA1 PROPOSA2 PROPOSA3 PROPOSA4 PROPOSA5 PROPOSA6 PROPOSA7	mygraph.ps	PROPOSAL, PROPOSA1, PROPOSA2, PROPOSA3, PROPOSA4, PROPOSA5, PROPOSA6, PROPOSA7

For more information, see “Replacing Existing External Files” on page 70.

Saving Multiple Graphs to Multiple Files

When you want your SAS/GRAPH program to create multiple files that each contain one graph, you can either

- use a FILENAME statements for each PROC statement to explicitly specify a destination file for each graph. To do this, simply repeat as many times as necessary the process described in “Saving One Graph to a File” on page 64.
- use one FILENAME statement to specify an aggregate file storage location as the destination and let SAS/GRAPH automatically name and create the files for you.

These steps describe the general process:

- 1 Use a FILENAME statement to define a fileref for the aggregate file storage location, such as a directory or PDS, where you want to send the output. Do not point to a specific file.
- 2 Assign the fileref to the GSFNAME= graphics option.
- 3 Specify the device driver with the DEVICE= graphics option.
- 4 Use the default setting GSFMODE=REPLACE so that SAS/GRAPH produces only one graph per file. Because REPLACE is the default setting, you can omit GSFMODE=.
- 5 Submit the SAS/GRAPH program.

Although the general technique is the same, there are significant differences between directing your graphics output to a specific file and directing it to an aggregate file storage location. These differences are mostly concerned with how the file is named.

When the destination is an aggregate file storage location, SAS/GRAPH not only creates the external file, it also names it. When SAS/GRAPH names an external file, it always uses the name of the entry in the output catalog. This name is either

- the name you specify in the NAME= option in the procedure
- the default name supplied by SAS/GRAPH, such as GSLIDE.

In addition, SAS/GRAPH automatically appends the correct file extension to the external file name. For example, if the output is named Q1SALES and the external file is a GIF file, the file name is Q1SALES.GIF.

This technique of building the file name from the catalog entry name affects what you do when you want to replace the contents of a file created in this way. For details, see “Replacing Existing External Files” on page 70.

Example

This example creates four text slides and stores each one in a separate external file. The program is similar to the previous examples except the fileref points to an aggregate storage location instead of to a specific file.

Define the fileref. The FILENAME statement assigns an aggregate file storage location as the destination for the files.

```
filename grafout 'external-file-location'; /* such as a directory */
```

Specify graphics options for the GSF. GSFNAME= assigns the fileref GRAFOUT as the destination for the GSF. GSFMODE=REPLACE (the default) replaces the contents of the external files with catalog entries of the same name.

```
goptions reset=all
         device=pscolor
         gsfname=grafout
```

```

gsfmode=replace
ftext=swissb
rotate=landscape;

```

Produce four text slides. Each RUN-group generates a new catalog entry. NAME= specifies the base name for each catalog entry that is generated by the procedure. This name is also the base name for the external files.

```

proc gslide border name='proposal';
  footnote h=2 j=1 'ABC Company';
  title1 h=4 'Proposed Design Improvements: ';
run;
  title1 h=3 '* Increase Stability';
run;
  title1 h=3 '* Increase Speed';
run;
  title1 h=3 '* Reduce Weight';
run;
quit;

```

When you submit these statements, SAS/GRAPH does the following if no graphs of the same name exist in the catalog:

- Creates a new catalog entry for each graph, beginning with the name PROPOSAL. After the first graph is created, the entry name is incremented so that subsequent graphs are called PROPOSA1, PROPOSA2, and so forth.
- Creates one external file for each catalog entry. The name of the file is the same as the entry name plus the extension. In this case, the files are named PROPOSAL.PS, PROPOSA1.PS, and so forth. For each file created, SAS/GRAPH sends a message to the LOG reporting the number of records and the name of the file to which they were written.

Note: Because you cannot replace individual entries in a catalog, each time you run the program SAS/GRAPH creates new catalog entries and consequently new files. If you want to replace the files, you must delete the corresponding catalog entries before resubmitting the program. \triangle

This table illustrates what happens if you submit this program twice without deleting the catalog entries:

Table 4.4

Pass	Catalog Entries	File Name
1	PROPOSAL	proposal.ps
	PROPOSA1	proposa1.ps
	PROPOSA2	proposa2.ps
	PROPOSA3	proposa3.ps
2	PROPOSA4	proposa4.ps
	PROPOSA5	proposa5.ps
	PROPOSA6	proposa6.ps
	PROPOSA7	proposa7.ps

Because the catalog names increment, there is never a matching file that the new catalog can replace. Therefore, unless you delete the existing entries, the program continues to create new files. To delete the existing entries, first run a GREPLAY procedure with the DELETE option:

```
proc greplay igout=work.gseg nofs;
  delete proposal proposa1 proposa2 proposa3;
```

Replacing Existing External Files

When you are working with aggregate file storage locations and automatic file naming, it is important to remember that GSFMODE=REPLACE replaces a file *only* if the name of the catalog entry is the same as the name of the file and you are using the same type of driver. For example, to replace a file named Q1SALES.PS, your program must create a catalog entry named Q1SALES, and you must also be using a PostScript driver. If the entry named Q1SALES already exists in the catalog, SAS/GRAPH will increment the name to Q1SALES1, and either create a new file with the incremented name or replace an existing file whose name matches the new incremented name.

Therefore, to replace the contents of existing external files with a new set of graphs, you must be sure that the catalog you are using does not already contain entries of the same name. There are several ways to assure that a catalog does not contain entries with the same names as your files:

- Use a temporary catalog, such as the default WORK.GSEG, to store the output and start a new SAS session. Initially, the catalog is empty.
- Use a temporary or permanent catalog and use the GREPLAY procedure to do either of the following:
 - delete the entire contents of the catalog before you submit your program
 - delete specified entries before you submit your program.

One additional method for replacing catalog entries is rarely recommended because it is easy to accidentally delete catalog entries that you did not intend to delete. If you want to replace the entire contents of the catalog and if you are running only one procedure, you can use the graphics option GOUTMODE=REPLACE. Whenever a new procedure starts, GOUTMODE=REPLACE replaces the *entire contents* of the current catalog with the new entries; it does not replace individual entries.

Example

The following example uses the GREPLAY procedure to explicitly delete specified catalog entries so that you can re-create them and replace the corresponding external files. This example uses the permanent catalog MYLIB.GRAFCAT.

- The GREPLAY procedure explicitly deletes existing catalog entries that have the same name as the entries to be created. If no entries exist, PROC GREPLAY issues a message and the program continues.
- The GSLIDE procedure generates four text slides and stores them in the catalog specified by GOUT=. NAME= specifies PROPOSAL as the base name for the catalog entries created by the procedure.

Define the libref for the permanent catalog.

```
libname mylib 'SAS-data-library';
```

Define the fileref. The FILENAME statement assigns an aggregate file storage location as the destination for the files.

```
filename grafout 'external-file-location';
```

Specify graphics options for the GSF. GSFNAME= assigns the fileref GRAFOUT as the destination for the GSF. GSFMODE=REPLACE (the default) replaces the contents of the external files with catalog entries of the same name.

```
goptions reset=all
         device=pscolor
         gsfname=grafout
         gsfmode=replace
         ftext=swissb
         rotate=landscape;
```

Delete existing catalog entries of the same name. The GREPLAY procedure deletes the specified catalog entries. These are the catalog names generated by the NAME= option in the procedure. If the entries do not exist, PROC GREPLAY issues a message and the program continues.

```
proc greplay nofs igout=mylib.grafcat;
  delete proposal proposal proposa2 proposa3;
run;
```

Produce four text slides. Each RUN-group generates a new catalog entry. NAME= specifies the base name for each catalog entry generated by the procedure. This name is also the base name for the external files.

```
proc gslide border gout=mylib.grafcat name='proposal';
  footnote h=2 j=1 'ABC Company';
run;
  title1 h=3 '* Increase Strength';
run;
  title1 h=3 '* Reduce Drag';
run;
  title1 h=3 '* Increase Resistance to Sheer';
run;
quit;
```

When you submit these statements, SAS/GRAPH does the following:

- deletes the specified entries from the catalog MYLIB.GRAFCAT.

- creates a new catalog entry in MYLIB.GRAFCAT for each slide, and increments the entry names: PROPOSAL, PROPOSA1, PROPOSA2, and PROPOSA3.
- creates one external file for each catalog entry. The file name is built from the catalog entry name. If a file of the same name already exists, SAS/GRAPH replaces the contents of the file.

Other Ways to Assign the Destination

You can use the GACCESS= graphics option to assign the destination for a graphics stream file. There are two ways to do this.

Using GACCESS=GSASFILE

This method is similar to the GSFNAME= method described in the previous sections.

- Use a FILENAME statement and assign a destination to the fileref GSASFILE. When you use GACCESS, the fileref must be named GSASFILE. The destination can be either a specific file or an aggregate file storage location.
- Assign GSASFILE to the GACCESS= graphics option instead of to GSFNAME=.

For example, these statements define and assign the fileref for an aggregate file storage location:

```
/* define a fileref for the destination */
filename gsasfile 'external-file-location';

/* assign the fileref and specify a device */
goptions reset=all gaccess=gsasfile device=gif;
```

Using GACCESS= to Explicitly Specify a Destination

You can also use GACCESS= to assign the destination and omit the FILENAME statement. In this case, you must also include the SASGASTD output format and quote the entire value. The destination can be either a specific file or an aggregate file storage location.

For example, this statement assigns a specific file location as the destination for the graphics stream file:

```
/* assign the fileref and specify a device */
goptions reset=all
gaccess='sasgastd > my-graph-file.gif'
device=gif;
```

Exporting SAS/GRAPH Output Using Modified Device Entries

If you frequently send graphics output to the same file or device, you may want to create a custom device entry that automatically sends your output to that destination. This simplifies the process by eliminating several graphics options from the GOPTIONS statement and allowing you to create the GSF by simply specifying the custom device entry with the DEVICE= graphics option.

To modify a device entry, use the GDEVICE procedure using either code-based statements or the GDEVICE windows. In either case, copy the original entry from the SASHELP.DEVICES catalog to your personal catalog (typically, GDEVICE0.DEVICES).

Then simply change the device parameters to create an entry that produces graphics stream files by default.

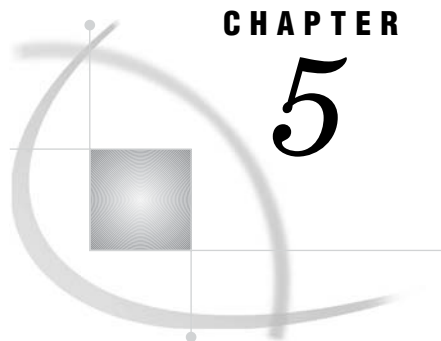
Often these device parameters correspond to the graphics options in your program. For example, if your program uses the fileref GRAFOUT, you can enter a value of **Grafout** in the **Gsfname:** field in the Host File Options window.

See Chapter 31, “The GDEVICE Procedure,” on page 915 for more information on modifying device entries. Refer to the Device Help for SAS/GRAPH in the SAS Help facility for lists of available drivers and for specific information on modifying device entries in your operating environment.

The Metagraphics Driver Facility

You can use the SAS/GRAPH Metagraphics facility to write your own device drivers to create files in other formats for use by other graphics software. The Metagraphics driver facility can be used to develop driver support for virtually any device you choose. It is device-intelligent and can support most hardware features. A user-written program is required to translate the Metagraphics metafile into the appropriate graphics language recognized by the hardware product.

For assistance in creating a Metagraphics driver, contact Technical Support.



CHAPTER

5

SAS/GRAPH Fonts

<i>Overview</i>	75
<i>Specifying Fonts in SAS/GRAPH Programs</i>	75
<i>Font Specifications</i>	76
<i>Default Fonts</i>	77
<i>Font Locations</i>	77
<i>Using Hardware Fonts</i>	78
<i>Default Hardware Fonts</i>	78
<i>Using a GOPTIONS Statement to Change the Default Hardware Font</i>	79
<i>Using the GDEVICE Procedure to Change the Default Hardware Font</i>	79
<i>Specifying the Full Font Name</i>	80
<i>Specifying Alternative Hardware Fonts</i>	80
<i>Specifying Special Characters</i>	81
<i>Using SAS/GRAPH Software Fonts</i>	82
<i>Rendering Fonts</i>	82
<i>Font Lists</i>	82

Overview

The SAS/GRAPH software has a variety of type styles that you can use to display text and special characters in your procedure output. These type styles are stored in *fonts* that you can specify when you want to select type for the text on your graphs or choose plot symbols.

This chapter explains how to specify a font, how to change the default hardware font, how to produce special characters, and how to select a software font.

After the FONTREG procedure runs, the SASSEM device driver and graphs in the ODS PDF destination can use FreeType fonts without any modification. However, you will need to modify the GIF, JPEG, PNG, and TIFFFP device drivers before these can access FreeType fonts. For details about using FreeType fonts, refer to The FONTREG Procedure.

Note: In some operating environments, you can access the System or TrueType fonts that are available to the host print driver that is currently set for your printer. For more information, see the SAS documentation for your operating environment. △

Specifying Fonts in SAS/GRAPH Programs

When you select a type style for text or plot symbols, you use statement options to assign the font. You can assign fonts for any amount of text from a single character in a

title to all the text in your output. When the SAS/GRAPH software encounters an explicit font specification in a SAS statement, it uses the font that you specify rather than a font that is specified in a GOPTIONS statement or the device's hardware font.

With SAS/GRAPH you can select existing hardware or software fonts or create your own font. Hardware fonts are fonts that exist on hardcopy output devices (such as printers or plotters) or on a computer. PC TrueType and UNIX system fonts are examples of hardware fonts that exist on computers. Software fonts are the fonts delivered with SAS. The software fonts are the entries in the SASHELP.FONTS catalogue. If you do not make a font assignment, in most cases the SAS/GRAPH software uses the default hardware font for your device.

Note: Java and ActiveX devices do not support software fonts. △

Font Specifications

A *font specification* is an argument that takes a font name as its value. Font specifications typically take the following form:

FONT=*font*

F=*font*

where *font* is a valid SAS name. The specified font can be

- a software font that is either
 - a catalog entry that is supplied by SAS Institute (for example, CENTB), or that is user-generated (for example, MYFONT generated by the GFONT procedure)
 - a system font that is available in your operating environment.

Note: Font names greater than eight characters in length must be enclosed in quotation marks. △

To see a list of available software fonts, issue the FONTLIST command from the SAS command line. See *Base SAS Software Help* for more information on the FONTLIST command. The resulting window enables you to copy a font name into the copy buffer so that you can paste the name into your SAS program. The window's Family box lists the software fonts that are supplied by SAS Institute. Choose the System button to see a list of the system fonts that are available in your operating environment. Choose the Help button for help on using the window. See "Font Lists" on page 82 for more information on the valid font names for Institute-supplied software fonts.

- a hardware font that is either in the form of
 - HWxxxxnnn
 - *hardware-font-name*.

See "Specifying Alternative Hardware Fonts" on page 80 for details.

For example, the following statement specifies the Century Bold font in a title:

```
title font=centb 'This is a Title';
```

However, there are other arguments that also take *font* as a value. For example, the FTEXT= option in the following GOPTIONS statement specifies the Century Bold font for all text that does not have a more explicit font specification:

```
goptions ftext=centb;
```

Note: In this chapter, the argument FONT= is used to represent any argument that takes *font* as its value. △

Default Fonts

When a font is needed, the SAS/GRAPH software looks first for a font specification in the statement or procedure that produces the output, and then it looks in the GOPTIONS statement. If no font specification is found, the SAS/GRAPH software uses one of the following:

- for TITLE1 statements, the default font is SWISS.
- for all other text, the default font is NONE. The NONE font specifies the default hardware font for the output device.

In some cases, the device's hardware font cannot be used and the SIMULATE font is used instead. The SIMULATE font is a software font that simulates the device's hardware characters by allowing the same amount of space for the text that the hardware characters use. The SIMULATE font is used whenever the default hardware font is unavailable, including the following situations:

- FONT=NONE or FONT=HWxxxnnn or no font is specified, *and* one of the following conditions or sets of conditions is also met:
 - GOPTIONS NOCHARACTERS is specified.
 - The device driver does not support hardware text.
 - You request a hardware font for a different device.
 - You specify an angle or rotation for the characters that the device does not support.
- The device does not have scalable hardware characters (that is, hardware characters can be generated only in the proportions specified with the font), *and* one of the following conditions is also met:
 - The values of the HPOS= and VPOS= graphics options do not match the values displayed in the LCOLS or PCOLS field or the LROWS or PROWS field in the Detail window of the device entry.
 - The HSIZE= or VSIZE= graphics option is set to values that are not the default.
 - You replay a graph in a template that is not the same size as the full size of the graphics output area, or you use a device driver other than the one you used to create the graph.
 - The target device and the display device have different values for the HPOS= and VPOS= graphics options.
 - You use any height specification, including the HEIGHT=, HTEXT=, HTITLE=, and HBY= graphics options, that is not equal to 1.

You should never delete the SIMULATE font from the fonts catalog.

Note: You can change the font that is used as the SIMULATE font with the SIMFONT= graphics option. If you use the SIMFONT= option, it is better to specify a uniform font. Do not specify a hardware font as a substitute for SIMULATE. See "SIMFONT" on page 351 for more information on the SIMFONT= option. △

Font Locations

SAS/GRAPH software fonts are stored in catalogs. The SAS/GRAPH software looks only into catalogs with certain librefs and names to find fonts. By default, SAS/GRAPH

searches for the font in the catalog SASHELP.FONTS, which contains Institute-supplied fonts, key maps, and device maps.

If you want to specify fonts that you have created locally, submit a LIBNAME statement that associates the libref GFONT0 with the location of your font catalog. If you have specified more than one libref in the sequence GFONT0 through GFONT9, the SAS/GRAPH software performs a sequential search of these catalogs when locating the font that you have specified.

When you specify a font name, the SAS/GRAPH software searches for the font in the following order:

- 1 If a SAS data library with the libref GFONT0 exists, then the SAS/GRAPH software looks there for a catalog named FONTS. If GFONT0.FONTS exists, it is checked for the specified font. If the font is not there, then the SAS/GRAPH software looks next for a library with the libref GFONT1 and for a catalog named FONTS in that library. The search is repeated for the sequence of librefs through GFONT9.
- 2 If the SAS/GRAPH software fails to find the specified font in any FONTS catalog in the libraries GFONT0 to GFONT9, or if it finds a GFONT n libref without a FONTS catalog, or if it encounters an undefined libref in that sequence before locating the specified font, then it searches for the font in SASHELP.FONTS. (SASHELP is one of the standard librefs defined automatically whenever you start your SAS session; you do not need to issue a LIBNAME statement to define it.)
- 3 If the specified font is not found in SASHELP.FONTS, then a warning is issued and the SIMPLEX font is used. The SIMPLEX font is the default software font and should never be deleted from the fonts catalog.

See Chapter 32, “The GFONT Procedure,” on page 939 for additional information on specifying the libref GFONT0.

Using Hardware Fonts

There are four ways to use hardware fonts with SAS/GRAPH output:

- By using the CHARTYPE= graphics options in a GOPTIONS statement to assign the number of a font listed in the Chartype window of your device entry as the default hardware font. See “Using a GOPTIONS Statement to Change the Default Hardware Font” on page 79 for details.
- By using the GDEVICE procedure to specify the number of the font you want to use as the default hardware font. See “Using the GDEVICE Procedure to Change the Default Hardware Font” on page 79 for details.
- By specifying the full font name as it appears on the Chartype window of the device driver entry. See “Specifying the Full Font Name” on page 80 for details.
- By explicitly specifying a hardware font name of the type HWxxxxnnn. See “Specifying Alternative Hardware Fonts” on page 80 for details.

There are several advantages to using hardware fonts instead of software fonts. Hardware fonts often are produced faster than software fonts and produce smaller output files. Also, some devices, such as laser printers with resident hardware fonts, may produce better quality output with hardware fonts than with software fonts.

Default Hardware Fonts

SAS/GRAPH software uses a device’s default hardware font to draw characters when both of the following conditions are true:

- No font specification is made in the SAS/GRAPH program, or FONT=NONE is specified.
- The hardware font can be used. See “Default Fonts” on page 77 for details on when hardware fonts cannot be used.

Every available hardware font for a particular device has a number associated with it. This number and the corresponding font name are listed in the Chartype window of the device entry for your device. The default hardware font is the font whose number is entered in the Chartype field in the Parameters window of the device entry. When FONT=NONE or no font is specified, SAS/GRAPH software uses the font assigned to this field.

If your device has more than one hardware font, there are two ways you can assign a different default hardware font:

- By specifying the font with the CHARTYPE= option in a GOPTIONS statement. See “Using a GOPTIONS Statement to Change the Default Hardware Font” on page 79
- By using the GDEVICE procedure to modify the value of the Chartype field in the Parameters window of your device entry. See “Using the GDEVICE Procedure to Change the Default Hardware Font” on page 79 for more details.

If your device has only one hardware font (this is often the case), the Chartype field has a value of 0.

Using a GOPTIONS Statement to Change the Default Hardware Font

To assign the default hardware font for your current SAS session, use the CHARTYPE= option in a GOPTIONS statement. Assign it the actual number of the hardware font as listed in the Chartype field in the Chartype window of the device entry for your device.

Using the CHARTYPE= option only changes the default font for the duration of your SAS session; using the CHARTYPE= option does not change the value of the field in the device entry. (See “CHARTYPE” on page 269 for a complete description of the CHARTYPE= option.)

When you specify a hardware font by using the graphics option CHARTYPE=*n* and the font specification NONE, the size of the character cells is determined by the current values for the HPOS= and VPOS= options. This means that the font is drawn using the current cell size. As a result, the aspect ratio of the displayed font may be different and the height of the characters, if displayed in cells, may be affected.

CAUTION:

Specifying a nonscalable hardware font with the CHARTYPE= option may cause the SIMULATE font to be used. Δ

In addition, if the font selected with CHARTYPE= is not scalable and if the values of HPOS= and VPOS= do not match the values of the Rows and Cols fields in the Chartype window, then the SIMULATE font is substituted.

Using the GDEVICE Procedure to Change the Default Hardware Font

To change the default hardware font with the GDEVICE procedure, change the Chartype field in the Parameters window for the device:

- 1 Invoke the GDEVICE procedure and select the entry for your device.
- 2 Go to the Chartype window and review the available fonts.

- 3 Note the number of the font that you want to use as the default font and go to the Parameters window.
- 4 Enter the number of the font in the Chartype field.
- 5 Close the window and exit the procedure.

Note: If you change the number in the Chartype field in the Parameters window of the device entry, the change is permanent and remains in effect from one SAS session to another until you change the entry again. △

(See Chapter 31, “The GDEVICE Procedure,” on page 915 for information on viewing device entries and changing device parameters.)

Specifying the Full Font Name

You can specify the full font name in any SAS statement where a font specification is valid (such as for the `FTEXT=font` graphics option or the `FONT=font` specification on a `TITLE` statement). For the value font, specify the full font name exactly as it appears in the Chartype window of the device driver entry. For example, to specify the Times-Roman font on a `TITLE` statement when you use the PS300 device driver, specify:

```
title font='Times-Roman' 'Testing the Times-Roman font';
```

The SAS System allows up to 255 characters for the font name. The font name may contain spaces. If the font name is longer than 40 characters, PROC GDEVICE in fullscreen mode only displays the first 37 characters, followed by an ellipsis (...). To see the complete font name when the name is longer than 40 characters, use PROC GDEVICE with the NOFS (no fullscreen) option as follows:

```
proc gdevice c=sashelp.devices nofs;
  list driver-name;
run;
quit;
```

When a font is quoted, the SAS System will first look at the Chartype window of the device driver entry to see if it is a valid hardware font. If the font is not found in the Chartype window, the SAS System will then check to see if the quoted font is a valid SAS/GRAPH software font. If the font is not recognized as either a valid hardware font or a valid SAS/GRAPH software font, the SIMPLEX font will be used.

Specifying Alternative Hardware Fonts

An alternative hardware font can be specified in any SAS statement where a font specification is valid. You can use more than one hardware font in a single graph (or even in a single statement), as long as all of the fonts that you specify exist on your device. If you specify a hardware font, make sure that the font is available on the device and that there is a corresponding Chartype value for the font. If you request a hardware font that does not have a Chartype defined, SAS/GRAPH software substitutes the SIMULATE font.

These are the three ways to specify alternative hardware fonts:

- In the font specification, explicitly assign a hardware font using the following form:

```
HWxxxnnn
```

HW

identifies the font as a hardware font. The font name must begin with the characters HW.

- xxx* are the last two or three characters of the module name in the Module field in the Detail window of your device entry. If the module name has eight characters (SASGDPSL, for example), use the last three characters (PSL). If the module name has only seven characters (SASGDVT, for example), use the last two characters (VT).
- nnn* is the Chartype number of the hardware font that you want to use as listed in the Chartype window in the device entry. This value should be a three-digit decimal number, with leading zeros if necessary.
- In the font specification, explicitly assign a hardware font using the following form:

hardware-font-name

- identifies the name of the hardware font that is listed in the Chartype window of the device entry. *Hardware-font-name* must be enclosed in quotation marks and the maximum length is 256 characters. The specified font name will be converted internally to the HW*xxxnnn* name. Note that in Annotate, the specified font name must be enclosed in both double quotes and single quotes (see Chapter 25, “Annotate Dictionary,” on page 613 for details).
- Assign one of the fonts listed in the Chartype window of your device entry as the default hardware font with the CHARTYPE= graphics option. You can also change the default hardware font by modifying the value of the Chartype field in the Parameters window of your device entry. Then you can use FONT=NONE in your SAS/GRAPH procedure or statement to specify the new default hardware font.

When you specify FONT=HW*xxxnnn* or *hardware-font-name*, the size of the character cells is determined by the values in the Rows and Cols fields in the Chartype window of the device entry, and the values of the HPOS= and VPOS= options are ignored for the font. Consequently, the font retains its original proportions. In addition, with this method the font catalog is checked for proportional spacing information. This information is used by the software to determine how much space to reserve for proportional text. See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for additional information.

Specifying Special Characters

Some fonts contain characters that are not mapped to the keyboard and cannot be typed directly into a text string. To display these special characters, substitute a character code or a hexadecimal value in the text string.

Character codes include the letters, numbers, punctuation marks, and symbols that are commonly found on a keyboard. They are usually associated with symbols or national alphabets. These codes enable you to display the character by specifying the font and using the keyboard character in the text string.

For example, to produce the character ζ, assign the Greek font and specify the character code **z** in the text string.

```
title font=greek 'z';
```

Hexadecimal values are any two-digit hexadecimal numbers enclosed in quotation marks and followed by the letter x, for example, '3D'x. (In double-byte character sets, the hexadecimal values contain four digits, for example, '4E60'x.)

You display characters with hexadecimal values the same way that you display them with character codes, that is, by specifying the font that contains the special character and placing the hexadecimal value in the text string. For example, this TITLE

statement uses hexadecimal 18 to produce £ in the Zapf type style. (This example assumes a U.S. key map).

```
title font=zapf '18'x;
```

Note: The character code or hexadecimal value associated with any character in any font is dependent on the key map that is currently being used. \triangle

In addition, you can use a key map to map selected characters to your keyboard. For example, if you want to be able to type e directly, you could create a key map that maps e to the key that usually generates the asterisk (*) and a device map that maps e to your output device. Then, when you press the * key, although you see * on your display, e is produced on your graphics device. See Chapter 34, “The GKEYMAP Procedure,” on page 983 for details.

Using SAS/GRAPH Software Fonts

Rendering Fonts

The SAS/GRAPH software includes methods of storing rendered versions of Bitstream fonts, along with three graphics options to control how the fonts are rendered.

When your graphics output uses one of the Bitstream fonts that are provided in the SAS/GRAPH software, SAS/GRAPH must process information contained in corresponding FONT catalog entries to determine how to draw characters of the specified size and typeface. The process of calculating the character shapes and sizes is known as *rendering* the font. Bitstream fonts that are available in the SAS/GRAPH software include the Century, Swiss, and Zapf families.

The SAS/GRAPH software can store rendered versions of the Bitstream fonts in memory or in special SAS files. Using these rendered versions of the fonts can provide a speed improvement when characters of the same size and style are used again during the SAS session. The SAS/GRAPH software can read the rendered version of the characters from memory or from the rendered font file rather than having to perform the rendering calculations again each time the characters are used. If you store the rendered fonts in files in a permanent SAS data set, the SAS/GRAPH software can use the rendered font files again in subsequent SAS sessions.

Note: Because the rendered font files use a special utility member type, they do not appear in the list of library members that is displayed when you issue a DIRECTORY command for the SAS data library in which the font files are stored. \triangle

You control whether and how rendered versions of fonts are stored using the FONTRES=, RENDER=, and RENDERLIB= graphics options. See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for details.

Font Lists

The fonts available with the SAS/GRAPH software are listed in the following tables.

All of the software fonts are stored in the catalog SASHELP.FONTS. For many fonts, the last letter or letters of the font name indicates weight or spacing of the font:

- | | |
|---|--|
| B | bold (thicker) |
| E | empty (outline) versions of their counterparts |

I	italic (slanted)
L	light (thin)
U	uniformly spaced versions of their counterparts; most of the SAS/GRAPH fonts that do not end in U are proportionately spaced; however, the Kanji fonts are always uniform.
X	expanded (wider characters and extra space between characters).

CAUTION:

Empty and uniform versions of fonts cannot be used if you have deleted their filled or proportionally spaced counterparts. △

If the label of a font in SASHELP.FONTS is "Depends on," it is possible to delete it. However, empty and uniform versions of fonts are generated from their regular, bold, or italic counterparts. Therefore, if you delete any of these fonts, you cannot use the uniform or empty version of that font. For example, you must have the CENTB (Century Bold) font in order to use the CENTBE (Century Bold Empty) font.

Figure 5.1 Roman Alphabet Text Fonts

Type Style	Font Name	Type Sample	Uniform Font
Brush	BRUSH	<i>ABCabc123</i>	
Century			
Bold	CENTB	ABCabc123	CENTBU
Bold Empty	CENTBE	ABCabc123	
Bold Italic	CENTBI	<i>ABCabc123</i>	CENTBIU
Bold Italic Empty	CENTBIE	<i>ABCabc123</i>	
Expanded	CENTX	ABCabc123	CENTXU
Expanded Empty	CENTXE	ABCabc123	
Expanded Italic	CENTXI	<i>ABCabc123</i>	CENTXIU
Expanded Italic Empty	CENTXIE	<i>ABCabc123</i>	
German	GERMAN	Œ ß € abc123	GERMANU
German Italic	GITALIC	Œ ß € abc123	GITALICU
Hershey			
Sans Serif	SIMPLEX	ABCabc123	SIMPLEXU
Sans Serif Bold	DUPLEX	ABCabc123	DUPLEXU
Serif	COMPLEX	ABCabc123	COMPLEXU
Serif Bold	TRIPLEX	ABCabc123	TRIPLEXU
Serif Bold Italic	TITALIC	<i>ABCabc123</i>	TITALICU
Serif Italic	ITALIC	<i>ABCabc123</i>	ITALICU
Old English	OLDENG	Æ Œ € abc123	OLDENGU
Script	SCRIPT	<i>ABCabc123</i>	
Cscript	CSCRIPT	<i>ABCabc123</i>	
Swiss	SWISS	ABCabc123	SWISSU
Empty	SWISSE	ABCabc123	
Bold	SWISSB	ABCabc123	SWISSBU
Bold Empty	SWISSBE	ABCabc123	
Bold Italic	SWISSBI	<i>ABCabc123</i>	SWISSBIU

(continued)

Figure 5.2 Roman Alphabet Text Fonts—continued

Type Style	Font Name	Type Sample	Uniform Font
Bold Italic Empty	SWISSBIE	<i>A B C a b c 1 2 3</i>	
Expanded	SWISSX	A B C a b c 1 2 3	SWISSXU
Expanded Empty	SWISSXE	<i>A B C a b c 1 2 3</i>	
Expanded Bold	SWISSXB	A B C a b c 1 2 3	SWISSXBU
Expanded Bold Empty	SWISSXBE	<i>A B C a b c 1 2 3</i>	
Italic	SWISSI	A B C a b c 1 2 3	SWISSIU
Italic Empty	SWISSIE	<i>A B C a b c 1 2 3</i>	
Light	SWISSL	A B C a b c 1 2 3	SWISSLU
Light Empty	SWISSLE	<i>A B C a b c 1 2 3</i>	
Zapf	ZAPF	A B C a b c 1 2 3	ZAPFU
Empty	ZAPFE	<i>A B C a b c 1 2 3</i>	
Bold	ZAPFB	A B C a b c 1 2 3	ZAPFBU
Bold Empty	ZAPFBE	<i>A B C a b c 1 2 3</i>	
Bold Italic	ZAPFBI	<i>A B C a b c 1 2 3</i>	ZAPFBIU
Bold Italic Empty	ZAPFBIE	<i>A B C a b c 1 2 3</i>	
Italic	ZAPFI	A B C a b c 1 2 3	ZAPFIU
Italic Empty	ZAPFIE	<i>A B C a b c 1 2 3</i>	

Table 5.1 Non-Roman Alphabet Fonts

Type Style	Font Name	Uniform Font Name
Arabic	ARABIC	
Arabic Empty	ARABICE	
Cyrillic	CYRILLIC	CYRILLIU
David	DAVID	
Davidb	DAVIDB	
Fsong	FSONG	FSONGU
Greek	GREEK	GREEKU
Greek (serif)	CGREEK	CGREEKU
Hebrew	HEBREW	
Hebrew	NHEBREW*	
Hebrewb	HEBREWB	
Hebrew Empty	HEBREWE	
Hei	HEI	HEIU
Hiragana	HIRA	
Hiragana	NHIRA*	
Kanji	KANJI	

Type Style	Font Name	Uniform Font Name
Kanji	KANSJIS	
Kanji Subset		
Kanji 1	KAN1	
Kanji 2	KAN2	
Kanji 3	KAN3	
Kanji 4	KAN4	
Kanji 5	KAN5	
Kanji 6	KAN6	
Kanji 7	KAN7	
Kanji 8	KAN8	
Katakana	KATA	
Katakana	NKATA*	
Mincho	MINCHO	MINCHOE

*This font requires a special keyboard and is host-dependent. If you are not equipped to use this font, use the host-independent version listed directly above.

Table 5.2 Symbol Fonts

Type Style	Font Name	Uniform Font Name
Cartographic	CARTOG	CARTOGU
Electronic	ELECTRON	ELECTROU
Marker	MARKER	
Marker	MARKERE	
Empty	*	
Math	MATH	MATHU
Music	MUSIC	MUSICU
Special	SPECIAL	SPECIALU
Weather	WEATHER	WEATHERU

*MARKERE is not displayed in the figures.

Figure 5.3 Cartographic Font

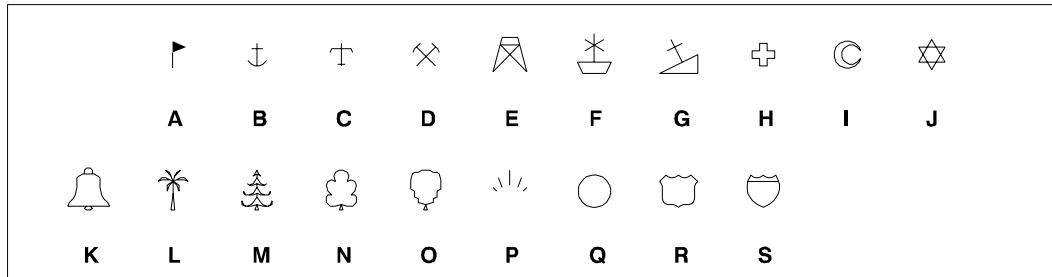
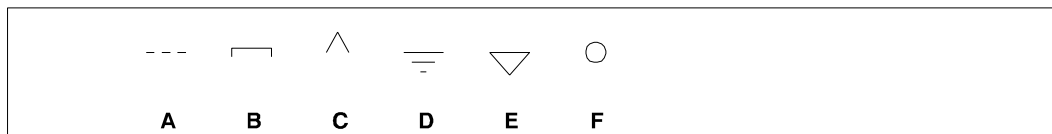


Figure 5.4 Electronic Font



Note: Figure 5.5 on page 87 shows the MARKER font. The MARKERE font produces the same symbols but in empty (outline) form. △

Figure 5.5 Marker Font

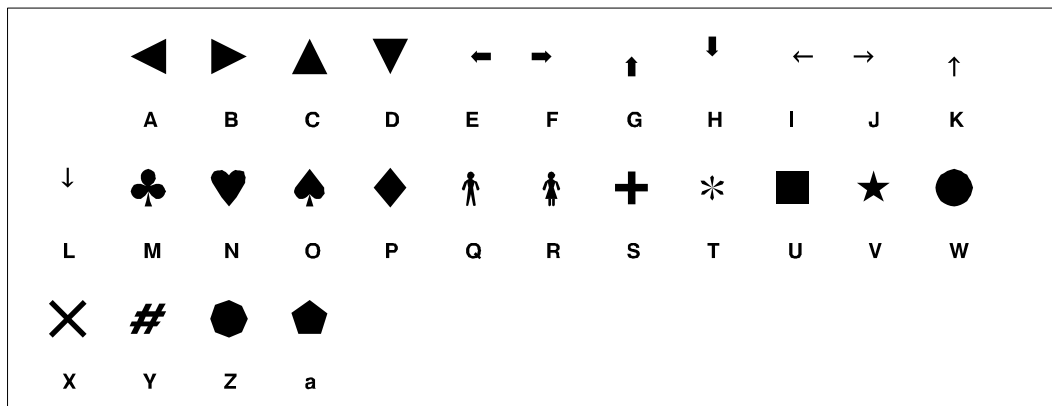


Figure 5.6 Math Font

\parallel	\perp	\angle	\therefore	$<$	$>$	\pm	\mp	\div	\neq	\equiv
A	B	C	D	E	F	G	H	I	J	K
\leq	\geq	∞	\sim	\surd	\subset	\cup	\supset	\cap	\in	\rightarrow
L	M	N	O	a	b	c	d	e	f	g
\leftarrow	\downarrow	∂	∇	\int	\oint	∞	\exists	\prod	Σ	
i	j	k	l	m	n	o	p	q	r	

Figure 5.7 Music Font

\cdot	\smile	\frown	\circ	\circ	\bullet	\sharp	\natural	b	—
A	B	C	D	E	F	G	H	I	J
—	—	—	—	—	—	—	—	—	—
K	L	M	N	O	P	Q	R	S	

Figure 5.8 Special Font

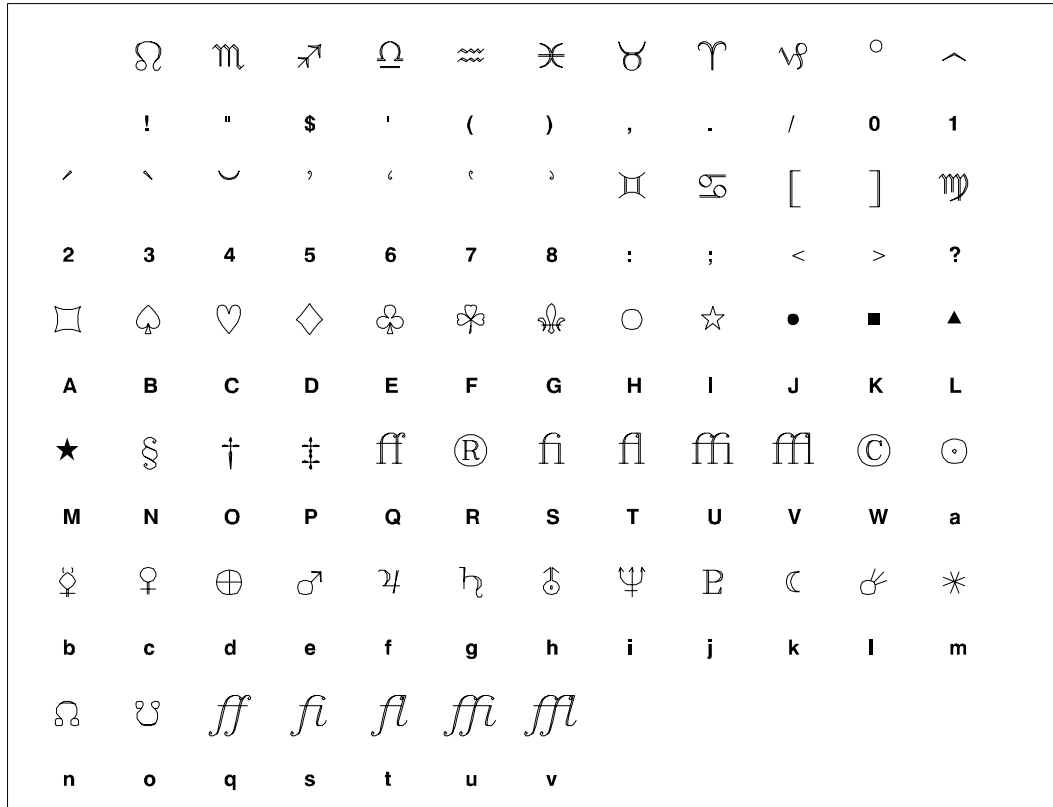
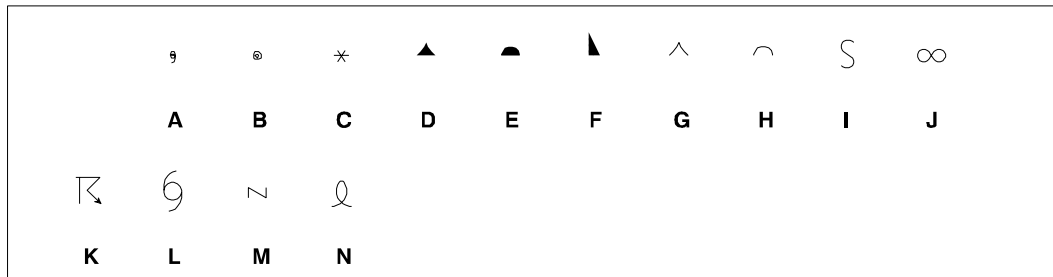
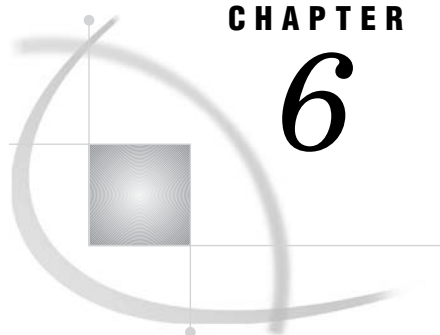


Figure 5.9 Weather Font





CHAPTER

6

SAS/GRAPH Colors and Images

<i>Using SAS/GRAPH Colors and Images</i>	92
<i>Specifying Colors in SAS/GRAPH Programs</i>	92
<i>Specifying Default Colors in a GOPTIONS Statement</i>	92
<i>Defining and Using a Colors List</i>	93
<i>Building a Colors List</i>	93
<i>Using a Device's Default Colors List</i>	94
<i>Overriding the Default Colors List</i>	94
<i>Resetting the Colors List to the Default</i>	94
<i>Applying ODS Styles</i>	94
<i>Color-Naming Schemes</i>	95
<i>Selecting a Color-Naming Scheme</i>	95
<i>Hardware-Oriented Color-Naming Schemes Overview</i>	95
<i>RGB Color Codes</i>	95
<i>CMYK Color Codes</i>	96
<i>User-Oriented Color-Naming Schemes Overview</i>	96
<i>HLS Color Codes</i>	97
<i>HSV (or HSB) Color Codes</i>	98
<i>Gray-Scale Color Codes</i>	99
<i>SAS Color Names and RGB Values</i>	99
<i>CNS Color Names</i>	99
<i>Using the Color Utility Macros</i>	100
<i>Colors and Device Capabilities</i>	103
<i>Devices That Do Not Support User-defined Colors</i>	103
<i>Devices That Support User-defined Colors</i>	104
<i>Pen Plotters</i>	104
<i>Limitations</i>	104
<i>Specifying Images in SAS/GRAPH Programs</i>	106
<i>Image File Types Supported by SAS/GRAPH</i>	106
<i>Reading and Writing Image File Types</i>	107
<i>Including the FORMAT= attribute</i>	108
<i>Image Formats for Reading</i>	108
<i>Image Formats for Writing</i>	110
<i>Image File Types Supported Only on Certain Hosts</i>	113
<i>Placing a Background Image</i>	113
<i>Placing a Backplane Image on Graphs with Frames</i>	115
<i>Placing Images on the Bars of Two-Dimensional Bar Charts</i>	116
<i>Using Annotate to Display an Image</i>	118
<i>Using DSGI to Display an Image</i>	119
<i>Disabling and Enabling Image Output</i>	120

Using SAS/GRAPH Colors and Images

SAS/GRAPH software lets you set colors or apply images to your graphics output, as described in “Specifying Colors in SAS/GRAPH Programs” on page 92 and in “Specifying Images in SAS/GRAPH Programs” on page 106.

Specifying Colors in SAS/GRAPH Programs

SAS/GRAPH software lets you set color

- in any procedure that generates graphics output (refer to the chapter for the individual procedure).
- in global statements that enhance procedure output: `AXIS`, `FOOTNOTE`, `LEGEND`, `NOTE`, `PATTERN`, `SYMBOL`, and `TITLE` (see Chapter 7, “SAS/GRAPH Statements,” on page 121).
- in the options on the `GOPTIONS` statement that defines default colors for graphics elements (see “Specifying Default Colors in a `GOPTIONS` Statement” on page 92).
- in the `COLORS=` option of the `GOPTIONS` statement to define a colors list (see “Defining and Using a Colors List” on page 93).
- in the colors list of the current device driver. (See “Colors and Device Capabilities” on page 103 for general information about device capabilities. See Example 1 on page 936 for information on how you can view or modify a device’s color list.)

These specifications, alone or in combination, give SAS/GRAPH software the colors it needs to generate graphics output. Colors can be specified using color names, such as `RED`, or color codes, such as `CXFF0000`. Color names must not exceed 64 characters. Color codes must not exceed eight characters. All color values must be in a valid SAS color-naming scheme (see “Color-Naming Schemes” on page 95).

Color specifications are searched for in the following order:

- 1 colors specified in the procedure itself
- 2 colors specified in global statements
- 3 color options in `GOPTIONS`
- 4 the `COLORS` list in the `GOPTIONS` statement
- 5 the color list in the current device driver.

SAS/GRAPH will search the color specifications in the order outlined above until a valid color is found. If no valid colors are specified, the color will be retrieved from the color list of the current device driver.

Specifying Default Colors in a `GOPTIONS` Statement

The `GOPTIONS` statement has several graphics options that specify default colors for graphics elements:

Option	Sets default color for
<code>CBACK=</code>	background for graphics output
<code>CBY=</code>	BY lines in graphics output
<code>CPATTERN=</code>	<code>PATTERN</code> statements

Option	Sets default color for
CSYMBOL=	SYMBOL statements
CTEXT=	all text and the border in graphics output
CTITLE=	border, plus all titles, footnotes, and notes

SAS/GRAPH software uses these values if you do not explicitly select colors in other statements. Refer to Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for complete information about each of these graphics options.

If you have not explicitly specified a color in a SAS statement or set a default color for a graphic element in the graphics output, SAS/GRAPH software searches for a colors list.

Defining and Using a Colors List

If SAS/GRAPH software does not find a color specification on a procedure or global statement, and a default color for the graphic element is not specified in the GOPTIONS statement, then SAS/GRAPH uses colors from the following colors list:

- 1 the COLORS= option on a GOPTIONS statement
- 2 the color list of the current device driver. (Use the GTESTIT procedure Example 2 on page 1292 to view the color list for a device driver.)

The color selected from the colors list varies depending on the procedure using the color and the graphics element that is being drawn. Usually, the first color in the list is used; however, certain procedures may select other colors. For example, if the CAXIS= option is not specified in the GCONTOUR procedure’s PLOT statement, the procedure selects the second color from the colors list to draw the axes. See the documentation for an individual procedure for more information.

Building a Colors List

To build a colors list, use the COLORS= option on the GOPTIONS statement. This specified colors list overrides the colors list of the current device driver. Building a colors list is useful for selecting a subset of colors to be used in a specific order for graphics output. For example, to ensure that the colors red, green, and blue are available in that order, you can specify:

```
goptions colors=(red green blue);
```

Each value you specify in a color list must be either a valid color name or a valid color code. Color names must not exceed 64 characters and color codes must not exceed eight characters. For an explanation of SAS color names and codes, see “Color-Naming Schemes” on page 95.

SAS provides, in the SAS Registry, a set of ordinary-language color names that you can use for colors that are common to most Web browsers. For a list of these names, showing both the ordinary-language name and its associated RGB value, you can run the following code and view the output in the Log window:

```
proc registry list
  startat='COLORNAMES';
run;
```

You can also modify entries in this list and add your own names and their associated RGB values. For more information on viewing and modifying the list of color names, see *Using the SAS Registry to Control Color* in *SAS Language Reference: Concepts*.

Note: The `COLORS=` graphics option only provides a default lookup table. Any time you explicitly select any other colors in your SAS/GRAPH program, those colors are used to draw the graphics elements for which you have specified them. \triangle

For a pen plotter, SAS/GRAPH software uses the order of the colors in the `COLORS=` graphics option to define the order of pens for a multiple pen plotter, or to make a pen plotter prompt you to change the pen when a graph uses more colors than the plotter has loaded.

Using a Device's Default Colors List

If you do not define a colors list with the `COLORS=` graphics option, then SAS/GRAPH software uses the colors list from the current device driver. This colors list is found in the device entry of the specified device driver (for details on specifying a device driver, refer to Chapter 36, “The GOPTIONS Procedure,” on page 1075). The colors list will change if you select a different device driver during a SAS session and have not specified the `COLORS=` graphics option.

To view and modify the device's default colors list, use the GDEVICE procedure (for details, refer to Chapter 31, “The GDEVICE Procedure,” on page 915).

Overriding the Default Colors List

If you do not specify colors for certain graphics elements or do not specify a colors list with the `COLORS=` graphics option, then the SAS/GRAPH procedures assign colors from the colors list of the current device driver. In some procedures, this assignment takes up some of the 256 colors that you can specify for a graph. For example, if no colors are specified, the G3D procedure uses the first three colors from the colors list to draw the text, the plot axes, and the plot symbols. These colors take up three of the 256 colors that you can specify for the graph. Therefore, you can specify 253 additional colors for your graph before a warning is issued and the colors are remapped.

To use only the colors that you explicitly specify in your SAS program, submit

```
goptions colors=(none);
```

The colors from any elements that have a color explicitly specified in your SAS program are combined to form a color list. This color list is used to assign colors to the elements that do not have a color explicitly specified. If no colors are explicitly specified, black will be used. This setting is useful if you want to generate graphics output with the maximum of 256 colors, and you do not want to use any of the default colors from the current device driver.

Resetting the Colors List to the Default

If you have specified a colors list with the `COLORS=` graphics option and you want to reset it back to the default colors list for the current device driver, then specify

```
goptions colors=();
```

Applying ODS Styles

If you are using ActiveX or Java device drivers to produce your graphic output, then you can specify Output Delivery System (ODS) styles to be used for your output's colors. The ODS styles contain predefined color schemes that can be used to create

professionally styled graphic output. The ODS styles are available for both bar graphs and pie charts. For more information on ODS Styles see “Using ODS Styles” on page 488 and *SAS Output Delivery System: User’s Guide*.

Color-Naming Schemes

The valid color-naming schemes are

- 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)
- the SAS Color Naming System (CNS).

For the CMYK color scheme, color specifications must be enclosed in quotation marks. For the SAS color names and CNS names, quotation marks are required if the color name contains spaces. In all other instances, quotation marks are optional.

You can freely intermix colors using different color-naming schemes in your programs. However, depending on your device capabilities, the color that is displayed may not be the color that you expect. See “Colors and Device Capabilities” on page 103 for information on how to tell if your device supports user-defined colors.

Selecting a Color-Naming Scheme

Each of the color-naming schemes supported by SAS/GRAPH offer their own set of advantages and disadvantages based on how the color-naming scheme and desired color will be implemented. RGB and CMYK are older hardware-oriented color-naming schemes. Creating specific RGB or CMYK colors may be less intuitive than creating colors using the user-oriented color-naming schemes: HLS, HSV, gray scale, SAS named colors, or CNS colors. For both hardware-oriented and user-oriented color-naming schemes, the color utility macros allow you to create colors for a specific color-naming scheme and convert color values between color-naming schemes (see “Using the Color Utility Macros” on page 100 for more information).

Hardware-Oriented Color-Naming Schemes Overview

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. With this color system, a color is defined by its red, green, and blue components. Individual amounts of each color are added together to create the desired result. All the colors combined together create white and the absence of all color is black.

CMYK is a special color-naming scheme used in four-color printing. Whereas the RGB scheme is based upon the principles of light, the CMYK scheme is based upon the principles of objects reflecting light. Cyan, magenta, and yellow absorb red, green, and blue light, respectively. When cyan is set at maximum, for example, all the red light is absorbed. Combining equal values of cyan, magenta, and yellow produces black, but this color may appear brown when printed. Therefore, the black component (K) of CMYK may 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.

RGB Color Codes

You can use the RGB color-naming scheme to specify a color in terms of its red, green, and blue components. Color names are of the form `CXrrggb`, where

- \square CX indicates that this is an RGB color specification
- \square *rr* is the red component
- \square *gg* is the green component
- \square *bb* is the blue component.

The components are given as hexadecimal numbers in the range 00 through FF (0% to 100%), where lower values are darker and higher values are brighter. This allows for up to 256 levels of each color component (over 16 million different colors). For example, bright red is specified as CXFF0000, white as CXFFFFFF, black as CX000000, and green as CX00FF00.

Any combination of the color components is valid. Some combinations will match the color produced by predefined SAS color names. See *SAS Language Reference: Concepts* for information on viewing the RGB combinations that match predefined SAS color names.

Note: When printed, RGB color values are automatically converted to the CMYK color values so that the colors display appropriately in the output. \triangle

CMYK Color Codes

To specify the colors from a printer's Pantone Color Look-Up Table, you can use the CMYK color-naming scheme to specify colors in terms of their cyan, magenta, yellow, and black components. Color names are of the form '*ccmmyykk*', where

- \square *cc* is the cyan component
- \square *mm* is the magenta component
- \square *yy* is the yellow component
- \square *kk* is the black component.

The components are given as quoted hexadecimal numbers in the range 00 through FF, where higher values are darker and lower values are brighter. This scheme allows for up to 256 levels of each color component. For example, red is specified as '00FFFF00', green as 'FF00FF00', process black (using cyan, magenta, and yellow ink) as 'FFFFFF00', and pure black (using only black ink) as '000000FF'. For CMYK color specifications, the quotes are required.

CMYK color specifications should only be used for devices that support four colors. If a CMYK color is used on a three-color device, the color specification will be mapped to a color that the device supports, but the resulting colors may not be appealing. Moreover, different CMYK colors may map to the same device color because a four-color space supports more colors than a three-color space.

Note: You can specify a CMY value by making the *kk*, the color's black component, zero (00). \triangle

User-Oriented Color-Naming Schemes Overview

The HLS color-naming scheme follows the Tektronix Color Standard illustrated in Figure 6.1 on page 98. To make the HLS color model consistent with the HSV coordinate system, Tektronix places blue at zero degrees. With the HLS color naming-scheme you specify the hue, lightness, and saturation levels.

With the HSV color-naming scheme, you specify the hue, saturation, and value (brightness) levels.

The gray scale color-naming scheme allows you to specify the lightness or darkness of gray using the word GRAY and a lightness value.

A predefined list of the SAS color names and their accompanying RGB values are contained in the SAS Registry Editor. The SAS Registry Editor also allows you to add your own SAS color names. With these colors, you can specify the name itself or the RGB value associated with that color name.

With CNS, you develop your color value by selecting and combining valid lightness, saturation, and hue terms. The CNS colors are based on the HLS color model and will display fairly uniform transitions between color values.

Note: Invalid color values will be replaced by the next valid color value on the foreground color list. Messages are written to the SAS log detailing the colors substituted for invalid color values. △

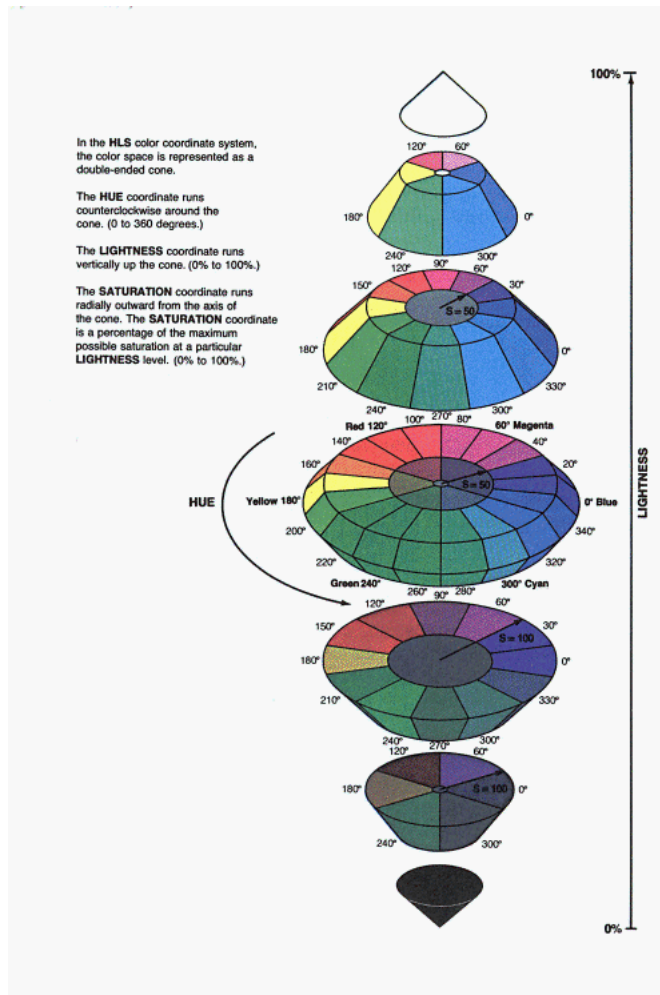
HLS Color Codes

You can use the HLS color-naming scheme to specify colors in terms of hue, lightness, and saturation components. SAS/GRAPH software uses an HLS color scheme that is modeled directly after the Tektronix Color Standard illustrated in Figure 6.1 on page 98. HLS color names are of the form *Hhhllss*, where

- H indicates that this is an HLS color specification
- *hhh* is the hue component
- *ll* is the lightness component
- *ss* is the saturation component.

The components are given as hexadecimal numbers. The hue component has the range of 000 through 168 hexadecimal (168 hexadecimal is equivalent to 360 decimal). Both the lightness and saturation components are hexadecimal and scaled to a range of 0 to 255 expressed with values of 00 through FF (0% to 100%). Thus, they provide 256 levels for each component. For example, blue is specified as H00080FF and light gray as H000BB00. 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 HxxxFF00 and black as Hxxx0000, where *xxx* can be any hue.

Figure 6.1 Tektronix Color Standard



HSV (or HSB) Color Codes

You can use the HSV color-naming scheme to specify colors in terms of hue, saturation, and value (or brightness) components. HSV color names are of the form $Vhhhssvv$, where

- V indicates that this is an HSV color specification
- hhh is the hue component
- ss is the saturation component
- vv is the value or brightness component.

The components are given as hexadecimal numbers. The hue component has the range of 000 through 168 hexadecimal (168 hexadecimal is equivalent to 360 decimal). Both the saturation and value (brightness) components are hexadecimal and scaled to a range of 0 to 255 expressed with values of 00 through FF (when the saturation is set to 00, the color is a shade of gray determined by the value). Thus, they provide 256 levels for each component.

For example, blue is specified as $V0F0FFFF$, light gray as $Vxxx00BB$, and white as $Vxxx00FF$, where xxx can be any hue. For white and black, the value component determines the intensity of gray level.

Gray-Scale Color Codes

Gray-scale color names are of the form GRAY*ll*. The value *ll* is the lightness of the gray and is given as a hexadecimal number in the range 00 through FF. This scheme allows for 256 levels on the gray scale. For example, GRAYFF is white, GRAY00 is black, and GRAY4C is a dark gray.

SAS Color Names and RGB Values

The SAS Registry Editor contains valid color names and RGB values. The predefined color names and RGB values in the SAS Registry are common to most web browsers. In addition to viewing predefined SAS color names and RGB values, the SAS Registry Editor also allows you to create and define your own color names and RGB values. See *SAS Language Reference: Concepts* for more information.

Note: Hardware characteristics may cause some colors with different color definitions to appear the same. Also, the same predefined color is likely to appear different on different devices and may not appear correctly on some devices. Δ

CNS Color Names

You can specify a CNS color value's lightness, saturation, and hue using the following terms:

Table 6.1

Lightness	Saturation	Hue
Black	Gray	Blue
Very Dark	Grayish	Purple
Dark	Moderate	Red
Medium	Strong	Orange/Brown
Light	Vivid	Yellow
Very Light		Green
White		

CNS values should be written in the following order: *lightness saturation hue*. The color names may be written without space separators between words, with an underscore to separate words, or with a space to separate words. The following list contains examples of valid color names:

```

verylightmoderatepurplishblue
very_light_moderate_purplish_blue
"very light moderate purplish blue"

```

Color names containing spaces must be enclosed in quotation marks.

Note: The %CNS macro only accepts CNS color names where a space is used to separate the words in the color name. See Table 6.5 on page 101 for more information Δ

If a CNS color name is also a color name in the SAS Registry, the SAS Registry color value will be used. 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, place quotes around the color name.

The lightness values black and white should not be used with saturation or hue values. If not specified, 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 CNS value is black, white, or some form of gray, at least one hue value must be used.

One or two hue values can be used in the CNS color name. When using two hue values, the hues must be adjacent to form a color. The hues are located in the following circular order: blue, purple, red, orange/brown, yellow, green, and then returning to blue. When two hues are used, the resulting color is a combination of both colors. Use the suffix “ish” to reduce the effect of a hue when two hues are combined. For example, reddish purple is less red than red purple. If you are using a color with an “ish” suffix, this color must precede the color without the “ish” suffix.

Using the Color Utility Macros

The %COLORMAC macro contains several subcomponent macros that can be used to construct and convert color values for the different color-naming schemes supported by SAS. The %HELPCLR macro provides information about the %COLORMAC subcomponent macros. The following table shows information that will appear in your SAS log when you call the %HELPCLR macro from the command-line:

Table 6.2 Using the %HELPCLR macro

Use...	To...
%HELPCLR;	List the color utility macro names.
%HELPCLR(ALL);	Display the short descriptions and examples for each of the color utility macros.
%HELPCLR(<i>macroname</i>);	Obtain a short description and an example of a specific color utilities macro. Replace <i>macroname</i> with the name of the color utility macro you are interested in.

When the color utility macros are invoked, the calculated color value can be directed to the SAS log or perform in-place substitutions in the code. The following tables list and describe the color utility macros and provide an usage example of each macro:

Table 6.3 %CMY(*cyan, magenta, yellow*);

Description	Usage Example
Replace <i>cyan, magenta, yellow</i> with numeric values to create a RGB color value. The numeric values that are used in place of <i>cyan, magenta, yellow</i> indicate the percentage (0 to 100) of each of these colors to be included in the RGB value.	<p>Entering the following code into your Program Editor:</p> <pre>%COLORMAC; data _null; put ' '%CMY(100,0,100)''; run;</pre> <p>Returns the RGB value CX00FF00 which is green.</p>

Table 6.4 %CMYK(*cyan, magenta, yellow, black*);

Description	Usage Example
Replace <i>cyan, magenta, yellow, black</i> with numeric values to create a CMYK color value. The numeric values that are used in place of <i>cyan, magenta, yellow, black</i> indicate the percentage (0 to 100) of each of these colors to be included in the CMYK color value. See “CMYK Color Codes” on page 96 for more information on the color value produced by using this macro.	<p>Entering the following code into your Program Editor:</p> <pre>%COLORMAC; data _null_; put %CMYK(0,46,16,31); run;</pre> <p>Returns the CMYK value 0075294F which is purple.</p>

Note: In the PUT statement, %CMYK(*cyan, magenta, yellow, black*), should not be placed in quotations. △

Table 6.5 %CNS(*colorname*);

Description	Usage Example
Replace <i>colorname</i> with a color-naming scheme color name to create a HLS color value. See “HLS Color Codes” on page 97 for more information on HLS color values. For more information on valid color-naming scheme color names see “CNS Color Names” on page 99 or enter the following into the command-line of the Program Editor: %HELPCLR(CNS);	<p>Entering the following code into your Program Editor:</p> <pre>%COLORMAC; data _null_; put '%CNS(GRAYISH REDDISH PURPLE)'; run;</pre> <p>Returns the HLS value H04B8040 which is grayish reddish purple.</p>

Table 6.6 %HLS(*hue, lightness, saturation*);

Description	Usage Example
Replace <i>hue, lightness, saturation</i> with numeric values to create a HLS color value. <i>Hue</i> should be replaced with any value from 0 to 360. <i>Lightness</i> and <i>saturation</i> indicate a percentage (0 to 100) to be included in the HLS color value. See “HLS Color Codes” on page 97 for more information.	<p>Entering the following code into your Program Editor:</p> <pre>%COLORMAC; data _null_; put '%HLS(0,50,100)'; run;</pre> <p>Returns the HLS value H00080FF which is blue.</p>

Table 6.7 %HSV(*hue, saturation, value*);

Description	Usage Example
Replace <i>hue, saturation, value</i> with numeric values to create a HLS value from HSV components. <i>Hue</i> should be replaced with any value from 0 to 360. <i>Saturation</i> and <i>value</i> (brightness) indicate a percentage (0 to 100) to be included in the HLS color value. See “HSV (or HSB) Color Codes” on page 98 and “HLS Color Codes” on page 97 for more information.	<p>Entering the following code into your Program Editor:</p> <pre>%COLORMAC; data _null_; put '%HSV(0,100,75)'; run;</pre> <p>Returns the HSV value V000FFBF which is dark red.</p>

Table 6.8 %RGB(*red, green, blue*);

Description	Usage Example
Replace <i>red, green, blue</i> with numeric values to create a RGB color value from RGB color components. The numeric values that are used in place of <i>red, green, blue</i> indicate the percentage (0 to 100) of each of these colors to be included in the RGB color value. See “RGB Color Codes” on page 95 for more information.	<p>Entering the following code into your Program Editor:</p> <pre>%COLORMAC; data _null_; put '%RGB(100,100,0)'; run;</pre> <p>Returns the RGB value CXFFFF00 which is yellow.</p>

Table 6.9 %HLS2RGB(*hls*);

Description	Usage Example
Replace <i>hls</i> with a HLS color value to create a RGB color value. See “HLS Color Codes” on page 97 and “RGB Color Codes” on page 95 for more information.	<p>Entering the following code into your Program Editor:</p> <pre>%COLORMAC; data _null_; put '%HLS2RGB(H04B8040)'; run;</pre> <p>Returns the RGB value CX9F5F8F which is grayish reddish purple.</p>

Table 6.10 %RGB2HLS(*rgb*);

Description	Usage Example
Replace <i>rgb</i> with a RGB color value to create a HLS color value. See “RGB Color Codes” on page 95 and “HLS Color Codes” on page 97 for more information.	<p>Entering the following code into your Program Editor:</p> <pre>%COLORMAC; data _null_; put ' '%RGB2HLS(CX9F5F8F)'' ; run;</pre> <p>Returns the HLS value H04C7F40 which is grayish reddish purple.</p>

Note: Round-trip conversions using HLS2RGB and then RGB2HLS macros or vice versa, may produce ultimate output values that differ from the initial input values. For example, converting CXABCDEF (a light blue) using %RGB2HLS (CXABCDEF) produces H14ACDAD. Converting this value back to RGB using %HLS2RGB (H14ACDAD) returns CXAACCEE. While not identical, the three colors are very similar on the display and when printed. Δ

For additional information on color-naming schemes, see *Effective Color Displays: Theory and Practice* by David Travis and *Computer Graphics: Principles and Practice* by Foley, van Dam, Feiner, and Hughes.

Colors and Device Capabilities

Your graphics output device determines the colors that you can use. SAS/GRAPH software translates the color that you specify to the color definition system on your device. It then checks to see if the color is available. If the color is not available, SAS/GRAPH software produces a message in the Log and remaps the color either to a color that closely matches the color you specified or to the next available color in the colors list. The remapping behavior depends on the capabilities of your device.

Graphics devices can be grouped into the following categories:

- devices that do not support user-defined colors
- devices that support user-defined colors
- pen plotters.

You can determine whether your device supports user-defined colors in two ways:

- Check the documentation for your graphics device.
- Run the GTESTIT procedure and display picture 1. The OPTS= string indicates if the device supports user-defined colors. If the second digit of the OPTS= string is an odd integer or d, b, or f, then the driver supports user-defined colors.

Devices That Do Not Support User-defined Colors

Devices that do not support user-defined colors are those that come with predefined nonalterable color palettes. These devices have a palette of between 1 and 256 predefined colors, but the actual number of colors that can be displayed at one time may be less than that.

When using devices that do not support user-defined colors, you can specify colors using any color-naming scheme; however, any unsupported colors that you specify in RGB, HLS, or gray-scale format are remapped to colors in your device's default colors

list. Some devices try to match the user-defined color with the closest color in the device's color palette. Others merely remap the color to the next available color in the device's color palette. To avoid having colors remapped, use the colors in your device's default colors list.

Devices That Support User-defined Colors

Devices that support user-defined colors are graphics devices that allow you to configure the colors from the device's color palette. A device in this category may have a palette of over 16 million colors, but it may only be able to display a subset of colors from the color palette at any one time.

On a device that supports user-defined colors, you can use any color-naming scheme to specify colors. If you specify a color that your device does not support, SAS/GRAPH software remaps the color to an available color that is the closest match.

Pen Plotters

The colors you can use with a pen plotter are determined by your set of pens. You specify the color names in the GOPTIONS statement's COLORS= option, then place pens with those color names in the plotter when the following message appears (the message does not appear if you specify the NOPROMPT graphics option):

```
Please mount the following pens: . . .
```

By default, SAS/GRAPH software tries to keep a standard set of pens in the plotter's carousel. If colors in the standard set of pens are named in the COLORS= graphics option, a mount request is issued for the entire standard set, even if all of them are not used in the output. This minimizes the number of times pens must be reshuffled as a graph is being drawn. If GOPTIONS COLORS=(NONE) is used, you are only prompted to mount pens for the colors that are actually used in the output.

Note: You can specify any valid SAS name for a color when using a pen plotter. It does not have to be a predefined or user-defined color name. For example, you can specify COLOR=PEN3, and you will be prompted to mount PEN3 or SAS/GRAPH software will assume a pen of that color is in the appropriate slot. This feature is often helpful if you are using transparency pens or other special pens. \triangle

Limitations

Using colors in SAS/GRAPH software is limited by the number of colors that you can use in one graph and by the capabilities of your device. The following sections discuss these limitations.

Maximum Number of Colors Displayable in SAS/GRAPH Software

You can use a maximum of 256 unique colors on each graph, including the background color (specified with the CBACK= graphics option or the CBACK device parameter). If you use more than 256 unique names in a program, SAS/GRAPH software issues a warning and remaps the 257th and any subsequent colors to existing color names, beginning with the first color name in the existing color list.

Note: If you specify a color using two different color-naming schemes for the same graph (for example, WHITE and RGB white, CXFFFFFF), SAS/GRAPH software counts them as two color specifications out of the 256 colors that you are allowed to use on one graph. \triangle

Maximum Number of Colors Displayable on a Device

The number of colors that you can display is limited by the type of graphics output device that you have. If you generate a graph with more colors than the device can

display, the colors that cannot be displayed are mapped to an existing color. You may also receive a note in the SAS log telling you when a color is mapped to another color in the colors list and what color will be used instead.

Although your device may support 256 colors, it may not let you use all of them at once. The MAXCOLORS device parameter tells SAS/GRAPH software the maximum number of colors that can be displayed at one time. MAXCOLORS is the number of foreground colors plus the background color and has a range of two to 256. If you use more than the number of colors set by the MAXCOLORS device parameter, the excess colors are remapped.

Note: The MAXCOLORS device parameter defaults to the number of displayable colors on the basic model of each graphics device supported. If your graphics device can display more colors than the base model, use the PENMOUNTS= graphics option to specify the number of colors that can be displayed. Optionally, you can use PROC GDEVICE to modify the value of the MAXCOLORS device parameter. △

For pen plotters, you use the PENMOUNTS= graphics option to indicate the number of pen holders on the plotter. PENMOUNTS is the number of foreground colors and has a range of one to 255. PENMOUNTS does not include the background color. Using this graphics option does not limit the number of colors that you can specify for a graph that is produced on a pen plotter. If you use more colors than the plotter has pen holders, you will be prompted to change pens unless you have also specified the NOPROMPT graphics option.

Replaying Graphs on a Device That Displays Fewer Colors

You can use the GREPLAY Procedure to display graphs previously generated. Sometimes you may need to replay the graphs on a device that cannot display as many colors as the device on which the graph was originally developed. Use “CMAP” on page 270 to control some of the remapping.

When you replay graphs on devices that display fewer colors than are in the graph, two situations may cause problems:

- Colors are specified that the device does not support.
- More colors are specified than the device can display at one time.

If you specify colors on a device that does not support those colors, the colors are remapped to those available for that device. You may also receive a note in the SAS log telling you when a color is mapped to another color in the colors list and what the new color is.

The number of colors that your device can display affects the actual color displayed. If your graphics output device can generate a maximum of 64 distinct colors and your graph contains 256 colors, the 65th through the 256th color specifications are remapped to the device’s available colors and may not display as the color you specify.

You can use the TARGETDEVICE= graphics option to preview the way a graph is going to look on a different device. You set this graphics option to the device entry name of the device driver that will be used later. The graph is displayed as close as possible to the way it will display when the other device is used.

Note: When you use the TARGETDEVICE= graphics option, SAS/GRAPH software uses the colors list of the target device as the default colors list; any color that you explicitly use is displayed when you preview the graph, although the target device may cause the color to be mapped. Refer to “TARGETDEVICE” on page 355 for complete information about the TARGETDEVICE= graphics option. △

Trueness of Color Displayed on a Device

The size of the color palette in your device determines the trueness of the color that is actually produced. For example, a device with a palette of 64 colors can

only produce colors that contain a combination of four shades of red (including *no* red), four shades of green (including *no* green), and four shades of blue (including *no* blue). Consequently, ranges of color will be remapped to available colors. For example, color specifications CX008080, CX258080, and CX3F8080 will all look the same (no red). CX408080 through CX7F8080 will look redder; CX808080 through CXBF0000, redder still; and CXC08080 through CXFF8080, the reddest. Larger palettes have more color resolution but may not have more simultaneous colors.

Specifying Images in SAS/GRAPH Programs

SAS/GRAPH software enables you to display images as part of your graph. You can place an image in the background area of a graph, in the backplane of graphs that support frames, or on the bars of two-dimensional bar charts. You can also apply images at specified graph-coordinate positions using the Annotate facility or the DATA Step Graphics Interface (DSGI).

The images you add to your graphs can be SAS files or external files in a wide range of image formats. If you wish to withhold images from your graphics output, you can specify the NOIMAGEPRINT graphics option.

Image File Types Supported by SAS/GRAPH

For displaying images in your graphs, SAS/GRAPH software supports the following image file types:

File Type	Description
BMP (Microsoft Windows Device Independent Bitmap)	supports color-mapped and true color images stored as uncompressed or run-length encoded. BMP was developed by Microsoft Corporation for storing images under Windows 3.0.
CAT (SAS Catalog IMAGE entry)	supports color-mapped images as well as true color images. The images can be optionally compressed.
DIB (Microsoft Windows Device Independent Bitmap)	see the description of BMP.
EMF (Microsoft NT Enhanced Metafile)	supported under Windows 95, Windows 98, Windows 2000, and Windows NT.
GIF (Graphics Interchange Format)	supports only color-mapped images. GIF is owned by CompuServe, Inc.
JPEG (Joint Photographic Experts Group)	supports compression of images with the use of JPEG File Interchange Format (JFIF) software. JFIF software is developed by the Independent Joint Photographic Experts Group.
PBM (Portable Bitmap Utilities)	supports gray, color, RGB, and bitmap files. The Portable Bitmap Utilities is a set of free utility programs that were primarily developed by Jeff Poskanzer.

File Type	Description
PCD (Photo CD)	Kodak Photo CD format which supports multiple image resolutions.
PCL (Printer Control Language)	developed by HP.
PCX (PC Paintbrush)	supports bitmap, color-map, and true color images. PCX and PC Paintbrush are owned by Zsoft Corporation.
PNG (Portable Network Graphic)	supports truecolor, gray-scale, and 8-bit images.
PS (PostScript Image File Format)	the Image classes use only PostScript image operators. A level II PS printer is required for color images. PostScript was developed by Adobe Systems, Inc.
TGA (Targa)	supports both true color images and color-mapped images; however, the current release of the Image classes supports only true color TGA files. Targa is owned by Truevision, Inc.
TIFF (Tagged Image File Format)	internally supports a number of compression types and image types, including bitmap, color-map, gray-scale, and true color. TIFF was developed by Aldus Corporation and Microsoft Corporation and is used by a wide variety of applications.
WMF (Microsoft Windows Metafile)	supported only under Microsoft Windows operating systems.
XBM (X Window Bitmaps)	supports bitmap images only. XBM is owned by MIT X Consortium.
XPM (X Window Pixmap)	is an extended version of XBM that supports color bitmaps.
XWD (X Window Dump)	supports all X visual types (bitmap, color-map, and true color.) XWD is owned by MIT X Consortium.

Reading and Writing Image File Types

The image file types supported by SAS/GRAPH may be supported for reading or writing. In addition, all SAS/GRAPH supported file types can be printed. The printed output's appearance is dependent on the driver that is selected. See "Image Formats for Reading" on page 108 and "Image Formats for Writing" on page 110 for more information on specific file type capabilities and attributes.

When you are reading some images, the `FORMAT=` attribute is required. See "Including the `FORMAT=` attribute" on page 108 for more information. Some of the file types that require the `FORMAT=` attribute are only supported by certain hosts. "Image File Types Supported Only on Certain Hosts" on page 113 contains information on these file types, the reader and writer attributes, and the host support.

If you are using SAS/GRAPH on the z/OS platform, you must be running the UNIX System Services Hierarchical File System (HFS) to read and write image files.

Including the **FORMAT=** attribute

When you are reading images, include the **FORMAT=** attribute if

- you are reading a format supported only on certain hosts. See “Image File Types Supported Only on Certain Hosts” on page 113 for more information.
- images reside in SAS catalogs.
- images are read from a system pipe.

FORMAT= is not required in other cases, but it is always more efficient to specify it.

Image Formats for Reading

The following table describes the attributes for the image readers that are supported by SAS/GRAPH:

File Type	Reader Attributes	Comment
BMP	FORMAT=BMP	
	COMPRESS=NONE	is the default. Sets compression to
	COMPRESS=RLE	run-length encoded.
CAT	FORMAT=CAT	
DIB	FORMAT=DIB	is supported only by Windows NT, Window 2000, and Windows XP.
EMF	FORMAT=EMF	is supported only by Windows NT, Window 2000, and Windows XP.
GIF	FORMAT=GIF	
JFIF	FORMAT=JFIF	required for reading JPEG files that use JPEG File Interchange Format (JFIF).
	DCT= <i>mode</i>	selects specific type of Discrete Cosine Transform (DCT) to use when processing the image; <i>mode</i> can be <ul style="list-style-type: none"> <input type="checkbox"/> INT — an integer DCT <input type="checkbox"/> FAST — a faster and less accurate integer DCT <input type="checkbox"/> FLOAT — a slightly more accurate method that can be slower unless the host has very fast floating point hardware.
	GRAYSCALE	produces a gray-scale image even if the JPEG file is in color. This is useful for viewing on monochrome displays. The reader runs noticeably faster in this mode.
	VERSION	prints the version number and copyright messages for the Independent JPE Group’s JFIF software to the log.
	FAST	enables certain recommended processing options for fast, low quality output; equivalent to enabling ONEPASS, DITHER=ORDERED, COLORS=216, NOSMOOTH, and DCT=FAST.

File Type	Reader Attributes	Comment												
	NOSMOOTH	uses a faster, lower quality, upsampling routine.												
	ONEPASS	uses a one-pass color quantization instead of the standard two-pass quantization. The one-pass method is faster and needs less memory, but it produces a lower-quality image. This attribute is ignored unless you also specify the COLORS attribute. ONEPASS is always enabled for gray-scale output.												
	COLORS= <i>n</i>	reduces the number of colors in the image to at most <i>n</i> colors; <i>n</i> must be in the range 2...256.												
	SCALE_RATIO= <i>n</i>	scales the output image by a factor of $1/n$. Currently the scale factor must be 1/1, 1/2, 1/4, or 1/8. This is useful when processing a large image and only a smaller version is needed, as the reader is much faster when scaling down the output.												
	DITHER= <i>mode</i>	selects the specific type of dithering to use color quantization; <i>mode</i> can be <ul style="list-style-type: none"> <input type="checkbox"/> FS — Floyd-Steinberg dithering <input type="checkbox"/> ORDERED — ordered dithering <input type="checkbox"/> NONE — no dithering. 												
PBM	FORMAT=PBM													
PCD	FORMAT=PCD	specifies photo CD format. FORMAT=PCD RES= specifies the image resolution to be read. Photo CD images have multiple resolution images in each image. Values are: <table style="margin-left: 20px;"> <tr> <td>BASE/64</td> <td>64x96</td> </tr> <tr> <td>BASE/16</td> <td>128x192</td> </tr> <tr> <td>BASE/4</td> <td>256x384</td> </tr> <tr> <td>BASE</td> <td>512x768 (default)</td> </tr> <tr> <td>4BASE</td> <td>1024x1536</td> </tr> <tr> <td>16BASE</td> <td>2048x3072.</td> </tr> </table>	BASE/64	64x96	BASE/16	128x192	BASE/4	256x384	BASE	512x768 (default)	4BASE	1024x1536	16BASE	2048x3072.
BASE/64	64x96													
BASE/16	128x192													
BASE/4	256x384													
BASE	512x768 (default)													
4BASE	1024x1536													
16BASE	2048x3072.													
PCX	FORMAT=PCX	is not supported for writing.												
PNG	FORMAT=PNG													
TGA	FORMAT=TGA													
TIFF	FORMAT=TIFF													
XBM	FORMAT=XBM													

File Type	Reader Attributes	Comment
XPM	FORMAT=XPM	is supported only under the X Windows System under UNIX.
XWD	FORMAT=XWD	

Image Formats for Writing

The following table describes the attributes for the image writers that are supported by SAS/GRAPH:

File Type	Writer Attributes	Comment
BMP	FORMAT=BMP	
CAT	FORMAT=CAT	
	COMPRESS=G3FAX	sets compression to FAX CCITT Group 3 for monochrome black-and-white images (depth of 1) only.
	COMPRESS=G4FAX	sets compression to FAX CCITT Group4 for monochrome black-and-white images (depth of 1) only.
DIB	DESC=	enables description for catalog description
	FORMAT=DIB	is supported only by Windows NT, Window 2000, and Windows XP.
EMF	FORMAT=EMF	is supported only by Windows 95, Windows 98, Windows 2000, and Windows NT.
GIF	FORMAT=GIF	
JFIF	FORMAT=JFIF	is required for writing JPEG files that use JPEG File Interchange Format (JFIF).
	DCT= <i>mode</i>	selects specific type of Discrete Cosine Transform (DCT) to use when processing the image; <i>mode</i> can be <ul style="list-style-type: none"> <input type="checkbox"/> INT — an integer DCT <input type="checkbox"/> FAST — a faster and less accurate integer DCT <input type="checkbox"/> FLOAT — a slightly more accurate method that can be slower unless the host has very fast floating point hardware.
	GRAYSCALE	produces a gray-scale image even if the JPEG file is in color. This is useful for viewing on monochrome displays. The writer runs noticeably faster in this mode.
	VERSION	prints the version number and copyright messages for the Independent JPE Group's JFIF software to the log.

File Type	Writer Attributes	Comment
	FAST	enables certain recommended processing options for fast, low quality output; equivalent to enabling ONEPASS, DITHER=ORDERED, COLORS=216, NOSMOOTH, and DCT=FAST.
	NOSMOOTH	uses a faster, lower quality, upsampling routine.
	ONEPASS	uses a one-pass color quantization instead of the standard two-pass quantization. The one-pass method is faster and needs less memory, but it produces a lower-quality image. This attribute is ignored unless you also specify the COLORS attribute. ONEPASS is always enabled for gray-scale output.
	COLORS= <i>n</i>	reduces the number of colors in the image to at most <i>n</i> colors; <i>n</i> must be in the range 2...256.
	SCALE_RATIO= <i>n</i>	scales the output image by a factor of $1/n$. Currently the scale factor must be 1/1, 1/2, 1/4, or 1/8. This is useful when processing a large image and only a smaller version is needed, as the writer is much faster when scaling down the output.
	DITHER= <i>mode</i>	selects the specific type of dithering to use color quantization; <i>mode</i> can be <ul style="list-style-type: none"> <input type="checkbox"/> FS — Floyd-Steinberg dithering <input type="checkbox"/> ORDERED — ordered dithering <input type="checkbox"/> NONE — no dithering.
PBM	FORMAT=PBM	
	COMPRESS=NONE	creates a text PBM file.
	COMPRESS=BINARY RAW	if either of these values is specified, creates a PBM file of reduced size by packing the pixels as binary data (when FORMAT=PBM is specified, output is produced by default as if one of these values is specified).
PCL	FORMAT=PCL	
	DPI= <i>num</i>	specifies the number of dots per inch to use to calculate the visual size of the image on the PostScript page in the output file. The default is 300. For example, a 600-pixel by 600-pixel image appears as a 2-inch by 2-inch image on the PostScript page if you use the default setting.
	EPS	does not reset the printer margins; use to embed an image into another PCL document

File Type	Writer Attributes	Comment
PNG	FORMAT=PNG	
PS	FORMAT=PS	
	COMPRESS=NONE	is the default for color images.
	COMPRESS=RLE	sets compression to run-length encoded; default for gray-scale images.
	DPI= <i>num</i>	specifies the number of dots per inch to use to calculate the visual size of the image on the PostScript page in the output file. The default is 300. For example, a 600-pixel by 600-pixel image appears as a 2-inch by 2-inch image on the PostScript page if you use the default setting.
	EPS	does not reset the printer margins; used to embed an image in another PCL document.
	PREVIEW	specifies whether a scaled-down, 1-bit, black-and-white preview image is written into the encapsulation header. The preview image enables this file to be written by software (such as SAS) that doesn't support a real PostScript writer.
	PREWIDTH= <i>x</i>	sets the size of the preview image in pixels if PREVIEW is specified (default: 25% of original size).
	PREHEIGHT= <i>y</i>	
	XSCALE	directly sets width scaling in points (1/72 inch). Default: calculate it.
	YSCALE	directly sets height scaling in points (1/72 inch). Default: calculate it.
	PAGEX	sets output page width in points (1/72 inch). Default: 612 (a typical 8.5-inch page).
	PAGEY	sets output page height in points (1/72 inch). Default: 792 (a typical 11-inch page).
	NOFIT	turns off the default of scaling an oversized image down to fit the page. Must be used with XSCALE and YSCALE. Although this option is still supported in this release, the PAGEFIT option will replace it.
PAGEFIT	<p>0 = Image size is not adjusted (equivalent of NOFIT).</p> <p>1 = Image size is adjusted only if the image exceeds the page size (default).</p> <p>2 = Image size is always adjusted to fill the page.</p> <p>This option replaces the NOFIT option.</p>	
TIFF	FORMAT=TIFF	
	COMPRESS=NONE	is the default

File Type	Writer Attributes	Comment
	COMPRESS=G3FAX	sets compression to FAX CCITT Group 3 for monochrome black-and-white (depth of 1) images only.
	COMPRESS=G4FAX	sets compression to FAX CCITT Group 4 for monochrome black-and-white (depth of 1) images only.
WMF	FORMAT=WMF	is supported only under Windows operating systems.
XBM	FORMAT=XBM	is supported for writing only from interactive windows under UNIX.
XPM	FORMAT=XPM	is supported only under the X Windows System under UNIX.

Image File Types Supported Only on Certain Hosts

Some file types are only supported by certain hosts. You must include a `FORMAT=` attribute when you are reading or writing the following image file types shown in the following table:

File Type	Reader Attributes	Writer Attributes	Host
DIB	FORMAT=DIB	FORMAT=DIB	Windows NT, Window 2000, and Windows XP
EMF	FORMAT=EMF	FORMAT=EMF	Windows NT, Window 2000, and Windows XP
WMF		FORMAT=WMF	Windows operating systems
XBM	FORMAT=XBM		interactive windows under UNIX
XPM	FORMAT=XPM	FORMAT=XPM	X Windows System under UNIX

Placing a Background Image

Any SAS/GRAPH procedure that produces a picture can place an image on the graph's background area. To place an image on the graph background, use the `IBACK=` option on a `GOPTIONS` statement. On `IBACK=`, specify either the full path to the image file in quotation marks, or a fileref that has been defined to point to the image file as follows:

```
goptions iback='external-image-file';
```

“Image File Types Supported by SAS/GRAPH” on page 106 shows the image file formats that you can use. “Disabling and Enabling Image Output” on page 120 shows how to suppress the image output without removing the imaging code from your

SAS/GRAPH program; for example, you might want to suppress the image when printing the graph.

By default, the image is tiled on the background, which means that the image is copied as many times as needed to fill the background area. You can specify `IMAGESTYLE=FIT` on the `GOPTIONS` statement to stretch the image so that a single image fits within the entire background area:

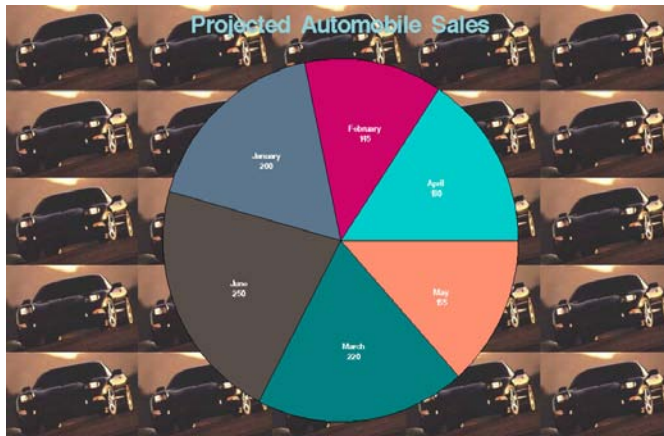
```
goptions iback='external-image-file'
        imagestyle=fit;
```

After fitting the image, you can tile subsequent images by resetting the `GOPTIONS` statement or by specifying `IMAGESTYLE=TILE`. The following graphs illustrate the use of tiled versus stretched images.

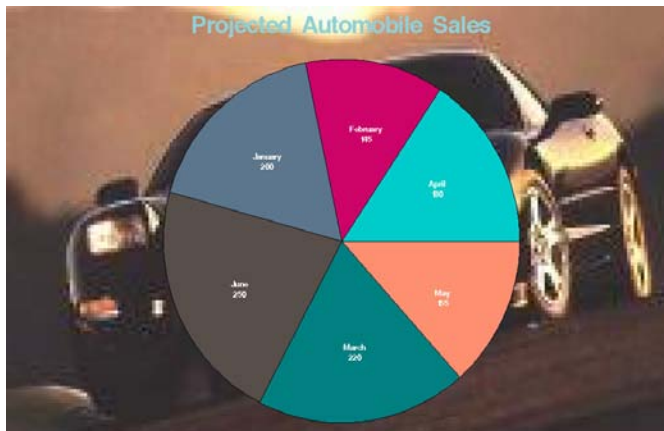
The following example displays an image behind a pie chart. Because the `IMAGESTYLE` option is not used, the image is tiled in the background area.

```
goptions reset=all ctitle=cx90d0d9 ftitle=swissb
        ctext=white htext=0.85 htitle=2.5 ftext=swissb
        colors=(cx00cccc cxcd0369 cx5b768d
                cx594f4a cx008080 cxff8f71)
        iback='external-image-file';
title 'Projected Automobile Sales';
data sales;
    length month $ 9;
    input month amount;
    datalines;
January    200
February   145
March      220
April      180
May        155
June       250
;
proc gchart;
    pie month / freq=amount value=inside
              noheading coutline=black;
run;
quit;
```

The preceding program generates the following graph, which illustrates the tiling of an image to fill an area.

Figure 6.2 Pie Chart with a Tiled Image in the Background

Adding the `IMAGESTYLE=FIT` option to the preceding program generates the following graphics output, where a single instance of the image is stretched to fit the background of the graph:

Figure 6.3 Pie Chart with a Stretched Image in the Background

Placing a Backplane Image on Graphs with Frames

Procedures `GCHART`, `GLOT`, `GRADAR`, and `GSLIDE` support frames, which are the backplanes behind the graphs. Each of these procedures enables you to place an image on the backplane.

To place an image on the backplane of a graph that supports frames, specify the `IFRAME=` option on the procedure that generates the graph. On the `IFRAME=` option, specify either the full path to the image file in quotation marks, or a fileref that has been defined to point to the image file as follows:

```
iframe=fileref | 'external-image-file'
```

“Image File Types Supported by SAS/GRAPH” on page 106 shows the image file formats that you can use. “Disabling and Enabling Image Output” on page 120 shows how to suppress the image output without removing the imaging code from your

SAS/GRAPH program; for example, you might want to suppress the image when printing the graph.

By default, the image is tiled on the backplane, which means that the image is copied as many times as needed to fill the backplane.

To stretch a single instance of the image to fill the backplane, specify `IMAGESTYLE=FIT` as follows:

```
iframe=fileref | 'external-image-file'
imagestyle=fit;
```

To switch from a single fitted image back to a series of tiled images, you can specify `IMAGESTYLE=TILE`.

The following example displays an image on the backplane of a horizontal bar chart. Because the `IMAGESTYLE=` option is not used, the image is tiled by default.

```
goptions reset=all ctitle=cx000080 ftitle=swissb
        ctext=black htext=0.85 htitle=2.5 ftext=swissb
        cback=cxf7e1c2;
title 'Projected Automobile Sales';
data sales;
    length month $ 9;
    input month amount;
    datalines;
January    200
February   145
March      220
April      180
May        155
June       250
;
pattern1 value=solid color=cxcd0369;
axis1 width=1.5 major=(width=1.5)
      label=( h=1 'Number of Cars') noplane;
axis2 width=1.5 major=(width=1.5)
      label=( h=1 'Month') noplane;

proc gchart;
    hbar3d month / freq=amount
            nostats
            axis=axis1
            maxis=axis2
            iframe='external-image-file'
            coutline=black;

run;
quit;
```

Placing Images on the Bars of Two-Dimensional Bar Charts

Using the `PATTERN` statement, you can place images on the two-dimensional bars of graphs that are generated by the `GCHART` procedure's `HBAR` or `VBAR` statements. To place an image on a two-dimensional bar, use the `IMAGE=` option on a `PATTERN` statement. On the `IMAGE=` option, specify the image file as follows:

```
pattern image=fileref | 'external-image-file';
```

“Image File Types Supported by SAS/GRAPH” on page 106 shows the image file formats that you can use. “Disabling and Enabling Image Output” on page 120 shows how to suppress the image output without removing the imaging code from your SAS/GRAPH program; for example, you might want to suppress the image when printing the graph.

By default, the image is tiled on the bar, which means that the image is copied as many times as needed to fill the bar area. You can specify `IMAGESTYLE=FIT` on the `PATTERN` statement to stretch a single instance of the image to fit the dimensions of the bar, as follows:

```
pattern image='external-image-file'
        imagestyle=fit;
```

After fitting the image, you can tile subsequent images by resetting the `PATTERN` statement or by specifying `IMAGESTYLE=TILE`.

Note: Images are only supported on the bars that are generated by the `HBAR` and `VBAR` statements. If an image is specified on a `PATTERN` statement that is used with another type of chart, then the `PATTERN` statement is ignored and default pattern rotation is affected. For example, if you submit a `PIE` statement when an image has been specified on `PATTERN`, the default fill pattern is used for the pie slices; however, rather than rotating that pattern through the colors list, each slice in the pie displays the fill pattern in the same color. △

The following example generates a bar chart that shows the sales for different automobile manufacturers. The bars that represent the sales figures for each manufacturer display a model vehicle for that manufacturer. Because `IMAGESTYLE=FIT` is not specified, each image is tiled on the bar that displays it.

```
goptions reset=all ctitle=cx000080 ftitle=swissb
        ctext=black htext=0.85 htitle=2.5
        ftext=swissb cback=cxf7e1c2;
title 'Projected Automobile Sales';
data sales;
    length Month $ 9 Manufacturer $ 10;
    input Month amount Manufacturer;
    datalines;
January    100 Nissan
February   80 Nissan
March     210 Nissan
April     201 Nissan
January   400 Dodge
February  90 Dodge
March    220 Dodge
April    202 Dodge
January  300 Cheverolet
February 70 Cheverolet
March   230 Cheverolet
April   203 Cheverolet
January 200 Ford
February 100 Ford
March   240 Ford
April   204 Ford
;
run;
pattern1 image='external-image-file'; /* corvette image */
pattern2 image='external-image-file'; /* viper image   */
```

```

pattern3 image='external-image-file'; /* mustang image */
pattern4 image='external-image-file'; /* nissan image */

axis1 label=( h=1 'Number of Cars');

proc gchart;
  vbar month / freq=amount
           coutline=black
           subgroup=Manufacturer
           axis=axis1
           cframe=olive;

run;
quit;

```

Using Annotate to Display an Image

The Annotate facility enables you to display an image at the coordinate location that you specify with the X and Y variables. To display an image, specify the file specification for the image file in quotation marks on the `IMGPATH` variable, set the image coordinates with the X and Y variables, and then call the `IMAGE` function, as shown in the following example. One corner of the image is located by the current X and Y position, and the opposite corner is located by the X and Y variables that are associated with the `IMGPATH` variable.

```

x=10; y=5; function='move'; output;
x=35; y=15; imgpath='external-image-file';
style = 'fit';
function='image'; output;

```

The code above draws an image from (10,5) to (35,15).

By default, the image is tiled, which means that it is copied as many times as needed to fill the area. To stretch the image so that a single image fits within the area, set the `STYLE` variable equal to 'fit', as shown in the code above.

“Image File Types Supported by SAS/GRAPH” on page 106 shows the image file formats that you can use. “Disabling and Enabling Image Output” on page 120 shows how to suppress the image output without removing the imaging code from your SAS/GRAPH program; for example, you might want to suppress the image when printing the graph.

Here is a complete example:

```

goptions reset=all cback=olive;
data wrldtotl;
  length company $ 10;
  input company $ 1-10 mean 12-15;
  datalines;
Nissan      550
Chevrolet  571
Ford       137
Dodge     273
Honda     546
Saturn    430
;
run;

data wrldanno;
  length function style color $ 8 text $ 20 image $ 50;

```

```

retain line 0 xsys ysys '2' hsys '3' x 8;
set wrldtotl end=end;
function='move'; x=x+8; y=20; output;
if company='Nissan' then
  imgpath='external-image-file'; /* Nissan image */
else if company='Chevrolet' then
  imgpath='external-image-file'; /* Corvette image */
else if company='Ford' then
  imgpath='external-image-file'; /* Mustang image */
else if company='Dodge' then
  imgpath='external-image-file'; /* Viper image */
else if company='Honda' then
  imgpath='external-image-file'; /* Honda image */
else if company='Saturn' then
  imgpath='external-image-file'; /* Saturn image */
function='image'; y=y+(mean); x=x+9;
output;
function='label'; y=0; x=x-4; size=3.5;
position='E'; style='swiss';
color='blue'; text=company; output;
function='move'; y=y+(mean)-3; output;
function='label'; x=x-1; text=left(put(mean,3.));
position='5'; style='swiss'; size=7; color='red'; output;
if end then do;
  function='move'; x=10; y=20; output;
  function='draw'; x=90; y=20; line=1;
  size=.5; color='blue'; output;
  function='label'; x=50; y=95; text='Projected Sales';
  xsys='3'; ysys='3'; position='5'; style='swissb';
  size=7; color=' '; output;
  x=92; y=5; size=3; style='swiss'; output;
  function='frame'; color='blue'; when='b';
  style='empty'; output;
end;
run;
proc ganno annotate=wrldanno
  datasys;
run;
quit;

```

Using DSGI to Display an Image

Using the DATA Step Graphics Interface (DSGI), you can display an image in a designated position. To display an image, specify the file specification for the image file in quotation marks on the GDRAW('IMAGE',...) function as follows:

```
rc=gdraw('image', 'external-image-file', 20, 20, 40, 40, 'fit');
```

The code above displays the image in the screen coordinates (20, 20) to (40, 40). The last parameter indicates how to display the image. The following keywords are available:

fit fits the image within the specified area. This stretches the image, if necessary.

tile tiles the image within the specified area. This copies the image as many times as needed to fit the area.

“Image File Types Supported by SAS/GRAPH” on page 106 shows the image file formats that you can use. “Disabling and Enabling Image Output” on page 120 shows how to suppress the image output without removing the imaging code from your SAS/GRAPH program; for example, you might want to suppress the image when printing the graph.

Here is a complete example:

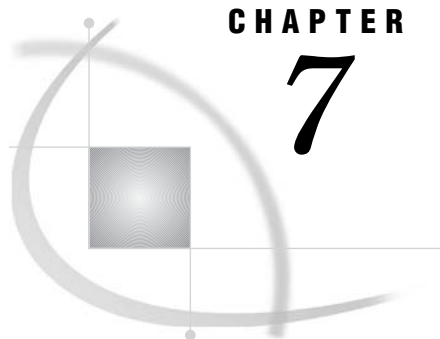
```
goptions reset=all;
title 'DGSI with image';
footnote ' dsqi with image option';
data image;
  rc=ginit();
  rc=graph('clear');
  rc=gdraw('image','external-image-file',
          5, 5, 90, 90,'fit');
  rc=graph('update');
  rc=gterm();
run;
quit;
```

Disabling and Enabling Image Output

The NOIMAGEPRINT graphics option disables image output as follows:

```
goptions noimageprint;
```

NOIMAGEPRINT can be useful for printing output without images. To enable image output, reset the GOPTIONS statement or specify the IMAGEPRINT graphics option.



CHAPTER

7

SAS/GRAPH Statements

<i>Overview</i>	123
<i>AXIS Statement</i>	124
<i>Description</i>	124
<i>Syntax</i>	125
<i>Options</i>	126
<i>Text Description Suboptions</i>	135
<i>Using Text Description Suboptions</i>	139
<i>Tick Mark Description Suboptions</i>	139
<i>Using the AXIS Statement</i>	140
<i>Assigning AXIS Definitions</i>	140
<i>BY Statement</i>	141
<i>Description</i>	141
<i>Syntax</i>	141
<i>Required Arguments</i>	142
<i>Options</i>	142
<i>Preparing Data for BY-Group Processing</i>	142
<i>Controlling BY Lines</i>	143
<i>Suppressing the BY line</i>	143
<i>Suppressing the name of the BY variable</i>	143
<i>Controlling the appearance of the BY line</i>	143
<i>Naming the Catalog Entries</i>	143
<i>Using the BY Statement</i>	143
<i>With the GCHART Procedure</i>	144
<i>With the GMAP Procedure</i>	144
<i>With the GLOT Procedure</i>	144
<i>With the RUN Groups</i>	144
<i>With the Annotate Facility</i>	145
<i>With TITLE, FOOTNOTE, and NOTE Statements</i>	145
<i>With PATTERN and SYMBOL Definitions</i>	145
<i>FOOTNOTE Statement</i>	146
<i>GOPTIONS Statement</i>	146
<i>Description</i>	146
<i>Syntax</i>	146
<i>Options</i>	150
<i>Using the GOPTIONS Statement</i>	150
<i>Graphics Option Processing</i>	150
<i>LEGEND Statement</i>	151
<i>Description</i>	151
<i>Syntax</i>	151
<i>Options</i>	152
<i>Text Description Suboptions</i>	158

<i>Using Text Description Suboptions</i>	161
<i>Using the LEGEND Statement</i>	161
<i>Positioning the Legend</i>	162
<i>Positioning the Legend on the Graphics Output Area</i>	162
<i>Using POSITION= and OFFSET=</i>	162
<i>Using ORIGIN=</i>	163
<i>Relating Legends to Other Graphic Elements</i>	163
<i>Interactions Between POSITION= and MODE=</i>	163
<i>Creating Drop Shadows and Block Effects</i>	163
NOTE Statement	164
ODS HTML Statement	164
Description	164
Syntax	164
Required Arguments	165
<i>Using the ODS HTML Statement</i>	168
<i>Specifying a Destination for ODS HTML Output</i>	168
About Anchors	168
PATTERN Statement	169
Description	169
Syntax	169
Options	170
<i>Using the PATTERN Statement</i>	176
<i>Altering or Canceling PATTERN Statements</i>	177
About Default Patterns	177
<i>How Default Patterns and Outlines Are Generated</i>	178
<i>Things That Affect Default Patterns</i>	178
Working with PATTERN Statements	179
<i>Explicitly Specifying Patterns</i>	179
<i>Generating Multiple Pattern Definitions</i>	179
<i>Selecting an Appropriate Pattern</i>	180
<i>Controlling Outline Colors</i>	180
<i>The Effect of the CPATTERN= Graphics Option</i>	180
<i>Specifying Version 6 Patterns</i>	181
Specifying Device-Dependent Hardware Patterns	181
GDDM Drivers	181
TEK42xx Series Terminal Drivers	181
HPLJxxxx Drivers	181
Metagraphics Drivers	181
Understanding Pattern Sequences	182
Generating Pattern Sequences	182
Repeating Pattern Sequences	183
SYMBOL Statement	183
Description	183
Syntax	184
Options	184
<i>Using the SYMBOL Statement</i>	202
<i>Altering or Canceling SYMBOL Statements</i>	203
Controlling Consecutive SYMBOL Statements	203
Setting Definitions for PROC GPLOT	204
Specifying Plot Symbols	205
Specifying a Default Interpolation Method	205
Sorting Data with Spline Interpolation	205
Using Color	206
Specifying Colors with SYMBOL Statements	206

Specifying Color with CSYMBOL=	207
Specifying Line Types	207
Using Generated Symbol Sequences	208
Default Symbol Sequences	209
Symbol Sequences Generated from SYMBOL Statements	209
TITLE, FOOTNOTE, and NOTE Statements	210
Description	211
Syntax	212
Options	212
Using TITLE and FOOTNOTE Statements	224
Using the NOTE Statement	224
Using Multiple Options	224
Setting Defaults	225
Using Options That Can Reset Other Options	225
Substituting BY Line Values in a Text String	226
Example 1. Ordering Axis Tick Marks with SAS Datetime Values	226
Example 2. Specifying Logarithmic Axes	229
Example 3. Rotating Plot Symbols through the Colors List	231
Example 4. Creating and Modifying Box Plots	233
Example 5. Filling the Area between Plot Lines	236
Example 6. Enhancing Titles	238
Example 7. Using BY-group Processing to Generate a Series of Charts	240
Example 8. Creating a Simple Web Page with the ODS HTML Statement	245
Example 9. Combining Graphs and Reports in a Web Page	248
Example 10. Creating a Bar Chart with Drill-down for the Web	255
Details	259
Building an HREF value	259
Creating an image map	259
Referencing SAS/GRAPH output	260
See Also	260

Overview

SAS/GRAPH programs can use some of the SAS language statements that you typically use with the base SAS procedures or with the DATA step, such as LABEL, WHERE, and FORMAT. These statements are described in the *SAS Language Reference: Dictionary*.

In addition, SAS/GRAPH has its own set of statements that affect only graphics output generated by the SAS/GRAPH procedures and the graphics facilities Annotate and DSGI. Most of these statements are *global statements*. That is, they can be specified anywhere in your program and remain in effect until explicitly changed or canceled. These are the SAS/PH global statements:

AXIS

modifies the appearance, position, and range of values of axes in charts and plots.

FOOTNOTE

adds footnotes to graphics output. This statement is like the TITLE statement and is described in that section.

GOPTIONS

submits graphics options that control the appearance of graphics elements by specifying characteristics such as default colors, fill patterns, fonts, or text height. Graphics options can also temporarily change device settings.

LEGEND

modifies the appearance and position of legends generated by procedures that produce charts, plots, and maps.

NOTE

adds text to the graphics output. This statement is an exception because it is not global but local, meaning that it must be submitted within a procedure. Otherwise, NOTE is like the TITLE statement and is described in that section.

PATTERN

controls the color and fill of patterns assigned to areas in charts, maps, and plots.

SYMBOL

specifies the shape and color of plot symbols as well the interpolation method for plot data. It also controls the appearance of lines in contour plots.

TITLE

add titles to graphics output. The section describing the TITLE statement includes the FOOTNOTE and NOTE statements.

These statements are described in this chapter, which also includes two Base language statements that have a special effect when used with SAS/GRAPH procedures:

BY

processes data according to the values of a classification (BY) variable and produces a separate graph for each BY-group value.

ODS HTML

generates one or more files written in Hyper Text Markup Language (HTML). If you use it with SAS/GRAPH procedures, you can specify one of the device drivers GIF, ACTIVEX, or JAVA (ACTIVEX and JAVA are only available with GCHART, GCONTOUR, GMAP, GPLOT, and G3D). With the GIF device driver, the graphics output is stored in GIF files. With the ACTIVEX device driver, graphics output is stored as XML input to ActiveX controls. With the JAVA device driver, graphics output is stored as XML input to Java applets. The HTML files that are generated reference the graphics output. When viewed with a Web browser, the HTML files can display graphics and non-graphics output together on the same Web page.

AXIS Statement

The **AXIS** statement controls the location, values, and appearance of the axes in plots and charts.

Used by:

GBARLINE, GCHART, GCONTOUR, GPLOT, and GRADAR procedures

Global

Description

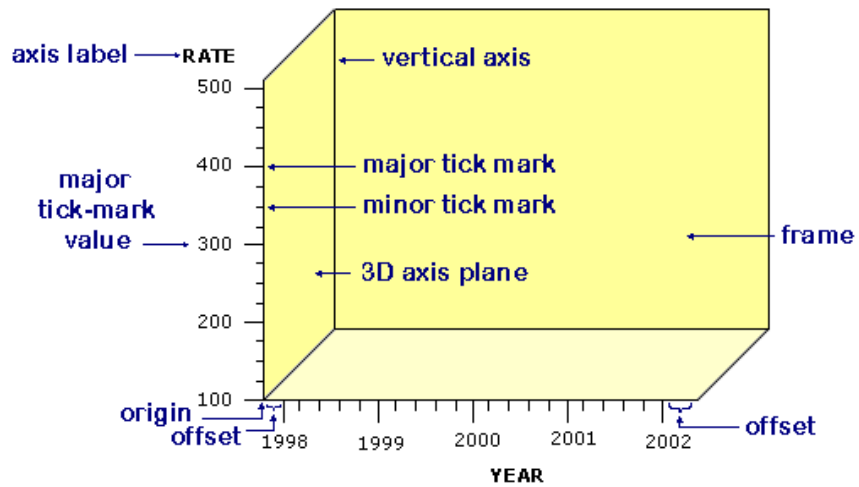
AXIS statements specify the characteristics of an axis, including:

- the way the axis is scaled
- how the data values are ordered
- the location and appearance of the axis line and the tick marks
- the text and appearance of the axis label and major tick mark values.

AXIS definitions are used only when they are explicitly assigned by an option in a procedure that produces graphs with axes.

Figure 7.1 on page 125 illustrates the terms associated with the various parts of axes.

Figure 7.1 Parts of Axes



Syntax

AXIS<1...99><options>;

option(s) can be one or more options from any or all of the following categories:

- axis scale options:

INTERVAL=EVEN | UNEVEN | PARTIAL

LOGBASE=*base* | E | PI

LOGSTYLE=EXPAND | POWER

ORDER=(*value-list*)

- appearance options:

COLOR=*axis-color*

LENGTH=*axis-length* <*units*>

NOBRACKETS

NOPLANE

OFFSET=(*<n1 ><n2 ><units >* | (*<n1<units>><n2<units >>*)

ORIGIN=(*<x><y ><units>* | (*<x<units >><y<units>>*)

STYLE=*line-type*

WIDTH=*thickness-factor*

- tick mark options:

MAJOR=(*tick-mark-suboption(s)*)| NONE

MINOR=(*tick-mark-suboption(s)*)| NONE

- text options:

LABEL=(*text-argument(s)*)| NONE

REFLABEL=(*text-argument(s)*)| NONE

SPLIT="*split-char*"

VALUE=(*text-argument(s)*)| NONE

Options

When the syntax of an option includes *units*, use one of these:

CELLS	character cells
CM	centimeters
IN	inches
PCT	percentage of the graphics output area
PT	points

If you omit *units*, a unit specification is searched for in this order:

- 1 GUNIT= in a GOPTIONS statement
- 2 the default unit, CELLS.

COLOR=*axis-color*

C=*axis-color*

specifies the color for all axis components (the axis line, all tick marks, and all text) unless you include a more explicit AXIS statement color specification. Any of these color specifications override COLOR= for the specified item:

Table 7.1

Option	Items Affected
AXIS statement:	axis label
LABEL=(COLOR= <i>color</i>)	reference-line labels
REFLABEL=(COLOR= <i>color</i>)	major tick mark values
VALUE=(COLOR= <i>color</i>)	
calling procedure:	all axis text (AXIS label and major tick mark value descriptions)
CTEXT=	
CAXIS=	axis line and major and minor tick marks

If you omit all color options, the AXIS statement looks for a color specification in this order:

- 1 the CTEXT= graphics option in a GOPTIONS statement.
- 2 If CTEXT= is not used, the color of all axis components is the first color in the colors list, except for PROC GCONTOUR, which uses the second color.

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226 .

INTERVAL=EVEN | UNEVEN | PARTIAL

The INTERVAL option affects the LOGBASE option on the AXIS statement. Specifying the option INTERVAL=UNEVEN and LOGBASE=10, permits non-base10 values to be specified for the ORDER option, while retaining a logarithmic scale for the axis.

LABEL=(*text-argument(s)*) | NONE

modifies an axis label. *Text-argument(s)* defines the appearance or the text of an axis label, or both. NONE suppresses the axis label. *Text-argument(s)* can be one or more of these:

'text-string'

provides up to 256 characters of label text. By default, the text of the axis label is either the variable name or a previously assigned variable label. Enclose each string in quotes. Separate multiple strings with blanks.

text-description-suboption

modifies a characteristic such as the font, color, or size of the text string(s) that follows it. *Text-description-suboption* can be

ANGLE=*degrees*

COLOR=*text-color*

FONT=*font* | NONE

HEIGHT=*text-height* <*units*>

JUSTIFY=LEFT | CENTER | RIGHT

ROTATE=*degrees*

See “Text Description Suboptions” on page 135 for a complete description.

Specify as many text strings and text description suboptions as you want, but enclose them all in one set of parentheses.

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226, “Example 2. Specifying Logarithmic Axes” on page 229 , and “Example 7. Using BY-group Processing to Generate a Series of Charts” on page 240

Not supported by: Java (partial), ActiveX (partial)

LENGTH=*axis length* <*units*>

specifies the length of the axis in number of units. If you request a length that cannot fit the display, an error message is issued and no graph is drawn.

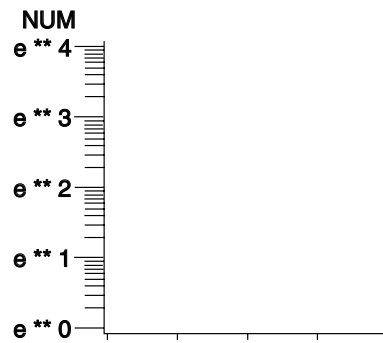
This option is not supported by the GRADAR Procedure.

Featured in: “Example 2. Specifying Logarithmic Axes” on page 229 and “Example 9. Combining Graphs and Reports in a Web Page” on page 248 .

Not supported by: Java, ActiveX

LOGBASE=*base* | E | PI

scales the axis values logarithmically according to the value specified. *Base* must be greater than 1. How the values are displayed on the axis depends on the LOGSTYLE= option. For example, LOGBASE=E with the default LOGSTYLE=EXPAND generates an axis like the one in Figure 7.2 on page 128.

Figure 7.2 Axis Generated with LOGBASE=E and LOGSTYLE=EXPAND

This option is not supported by the GRADAR Procedure.

Featured in: “Example 2. Specifying Logarithmic Axes” on page 229.

Not supported by: Java

LOGSTYLE=EXPAND | POWER

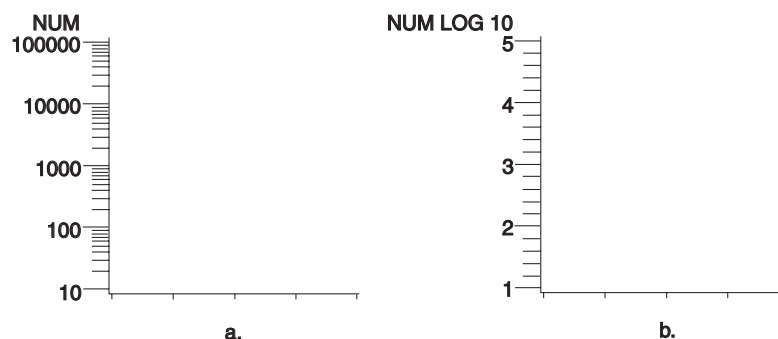
specifies whether the values displayed on the logarithmic axis are the values of the base or the values of the power. LOGSTYLE= is meaningful only when you use LOGBASE=.

LOGSTYLE=EXPAND specifies that the values displayed are the values of the base raised to successive powers and that the minor tick marks are logarithmically placed. For example, if the base is 10, the values displayed are 10, 100, 1000, 10000, and so on. The default is LOGSTYLE=EXPAND. This statement generates an axis like the one in part (a) of Figure 7.3 on page 128:

```
axis logbase=10 logstyle=expand;
```

LOGSTYLE=POWER specifies that the values displayed are the powers to which the base is raised (for example, 1, 2, 3, 4, 5, and so on). For example, this statement generates an axis like the one in part (b) of Figure 7.3 on page 128:

```
axis logbase=10 logstyle=power;
```

Figure 7.3 Axes Generated with the LOGSTYLE=option

If you use ORDER= with a logarithmic axis, the values specified by ORDER= must match the style specified by LOGSTYLE=. For example, if you specify a logarithmic axis with a base of 2 and you want to display the first five expanded values, use this statement:

```
axis logbase=2 logstyle=expand
    order=(2 4 8 16 32);
```

If you use LOGSTYLE=POWER, the values in ORDER= must represent the powers to which the base is raised, as in this example:

```
axis logbase=2 logstyle=power order=(1 2 3 4 5);
```

If the values that are specified by ORDER= do not match the type of values specified by LOGSTYLE=, the request for a logarithmic axis is ignored.

This option is not supported by the GRADAR Procedure.

Featured in: “Example 2. Specifying Logarithmic Axes” on page 229 .

Not supported by: Java

MAJOR=(*tick-mark-suboption(s)*)| NONE

modifies the major tick marks. *Tick-mark-suboption(s)* defines the color, size, and number of the major tick marks. NONE suppresses all major tick marks, although the values represented by those tick marks are still displayed.

Tick-mark-suboption can be

COLOR=*tick-color*

HEIGHT=*tick-height* <*units* >

NUMBER=*number-of-ticks*

WIDTH=*thickness-factor*

See “Tick Mark Description Suboptions” on page 139 for complete descriptions.

List all suboptions and their values within the parentheses.

AXIS definitions assigned to the group axis of a bar chart by the GAXIS= option ignore MAJOR= because the axis does not use tick marks.

Note: By default, tickmarks are now placed at three intervals on the spokes of a GRADAR chart. They are placed at the minimum value, maximum value, and at one value in between. The tick marks on the 12 o'clock spoke are also labeled by default.

HEIGHT is not supported by Java or ActiveX. WIDTH is not supported by Java. △

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226 , “Example 2. Specifying Logarithmic Axes” on page 229, and “Example 7. Using BY-group Processing to Generate a Series of Charts” on page 240.

Not supported by: Java (partial), ActiveX (partial).

MINOR=(*tick-mark-suboption(s)*)| NONE

modifies the minor tick marks that appear between major tick marks.

Tick-mark-suboption(s) defines the color, number, or size of the minor tick marks.

NONE suppresses all minor tick marks. *Tick-mark-suboption* can be

COLOR=*tick-color*

HEIGHT=*tick-height* <*units* >

NUMBER=*number-of-ticks*

WIDTH=*thickness-factor*

See “Tick Mark Description Suboptions” on page 139 for complete descriptions.

List all suboptions and their values within the parentheses.

AXIS definitions assigned to the group axis of a bar chart by the GAXIS= option ignore MINOR= because the axis does not use tick marks.

This option is not supported by the GRADAR Procedure.

HEIGHT is not supported by Java or ActiveX.

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226, “Example 2. Specifying Logarithmic Axes” on page 229, and “Example 7. Using BY-group Processing to Generate a Series of Charts” on page 240.

Not supported by: Java (partial), ActiveX (partial)

NOBRACKETS

suppresses the printing of group brackets drawn around the values on the group axis in a bar chart. NOBRACKETS applies only to the group axis of bar charts.

This option is not supported by the GRADAR Procedure.

See also: GROUP= on page 805 and GAXIS= on page 805.

Not supported by: Java, ActiveX

NOPLANE

removes either the horizontal or vertical 3D axis plane in bar charts produced by the HBAR3D and VBAR3D statements. NOPLANE affects only the axis to which the AXIS statement applies.

To remove selected axis elements such as lines, values or labels, use specific AXIS statement options. To remove all axis elements except the 3D planes use the NOAXIS option in the procedure. To remove the backplane, use the NOFRAME option in the procedure.

This option is not supported by the GRADAR Procedure.

Featured in: “Example 7. Using BY-group Processing to Generate a Series of Charts” on page 240.

OFFSET=(*<n1><n2><units > | (<n1<units>><n2<units>>*)

specifies the distance from the first and last major tick marks or bars to the ends of the axis line.

The value of (*n1*) is the distance from the beginning (origin) of the axis line to the first tick mark or middle of the first bar, and the value of (*n2*) is the distance from the end of the axis line to the last tick mark or middle of the last bar.

On a horizontal axis, the (*n1*) offset is measured from the left end of the axis line and the (*n2*) offset is measured from the right end. On a vertical axis, the (*n1*) offset is measured up from the bottom of the axis line and the (*n2*) offset is measured down from the top of the line.

To specify the same offset for both *n1* and *n2*, use one value, with or without a following comma. For example, either option sets both *n1* and *n2* to 4 centimeters:

```
offset=(4 cm)
offset=(4 cm,)
```

To specify different offsets, use two values, with or without a comma separating them. For example,

```
offset=(4 cm, 2 cm)
```

To specify only the second offset, use only one value preceded by a comma. This option offsets the last major tick mark or bar 3 centimeters from the right-hand end of the axis line:

```
offset=(, 3 cm)
```

You can specify *units* for the *n1,n2* pair or for the individual offset values.

This option is not by the GRADAR Procedure .

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226.

Not supported by: Java

ORDER=(*value-list*)

specifies the order in which data values appear on the axis. The values specified by ORDER= are the major tick mark values. You can modify the appearance of these values with the VALUE= option.

The way you specify *value-list* depends on the type of variable:

- For *numeric variables*, *value-list* is either an explicit list of values or a starting and an ending value with an interval increment, or a combination of both forms:

n <...*n*>

n TO *n* <BY *increment*>

n<...*n*> TO *n* <BY *increment* > <*n* <...*n* > >

If a numeric variable has an associated format, the specified values must be the *unformatted* values.

Values must be listed in either ascending or descending order. By default the increment value is 1. You can use a negative integer for *increment* to specify a value list in descending order. In all forms, multiple *n* values can be separated by blanks or commas. Here are some examples:

```
order=(2 4 6)
```

```
order=(6,4,2)
```

```
order=(2 to 10 by 2)
```

```
order=(50 to 10 by -5)
```

If the specified range is not evenly divisible by the increment value, the highest value displayed on the axis is the last incremental value below the ending value for the range. For example, this value list produces a maximum axis value of 9:

```
order=(0 to 10 by 3)
```

- For *character variables*, *value-list* is a list of unique character values enclosed in quotes and separated by blanks:

'*value-1*' <...'value-*n*'>

If a character variable has an associated format, the specified values must be the *formatted* values for PROC GCHART and the *unformatted* values for PROC GPLOT.

Character values can be specified in any order, but the character strings must match exactly the variable values in case and spelling. For example,

```
order=('Paris' 'London' 'Tokyo')
```

Observations can be inadvertently excluded if entries in the *value-list* are misspelled or if the case does not match exactly.

- For *date and time values*, *value-list* can have the following forms:

'SAS-*value*'*i* <...'SAS-*value*'*i*>

'SAS-*value*'*i* TO 'SAS-*value*'*i* <BY *interval*>

'SAS-*value*'*i*

is any SAS date, time, or datetime value described for the SAS functions INTCK and INTNX. Enclose the value in quotes and specify one of the following for *i*:

D date

T time

DT datetime

interval

is one of the valid arguments for the INTCK or INTNX functions. These are the default intervals:

DAY	default interval for date
SECOND	default interval for time
DTSECOND	default interval for datetime

These value lists use SAS date and time values:

```
order=('25MAY98'd '04JUL98'd '07SEP98'd)
order=('01JUL97'd to '01AUG97'd)
order=('01JUL97'd to '01JAN98'd by week)
order=('9:25't to '11:25't by minute)
order=('04JUN97:12:00:00'dt to
      '10JUN9712:00:00'dt by dtday)
```

With SAS date and time values, use a FORMAT statement so that the tick mark values have an understandable form. For more information on SAS date and time values, see the *SAS Language Reference: Dictionary*.

With any type of *value-list*, specifying values that are not distributed uniformly or are not in ascending or descending order, generates a warning message in the SAS log. The specified values are spaced evenly along the axis even if the values are not distributed uniformly.

Using ORDER= to restrict the values displayed on the axis may result in clipping. For example, if the data range is 1 to 10 and you specify ORDER=(3 TO 5), only the data values from 3 to 5 appear on the plot or chart. For charts, the omitted values are still included in the statistic calculation.

Note: Values out of range do not always produce a warning message in the SAS log. \triangle

CAUTION:

The ORDER= option does not calculate midpoint values; as a result it is not interchangeable with the MIDPOINTS= option in the GCHART procedure. \triangle

You can use ORDER= to specify the order in which the midpoints are displayed on a chart, but do not use it to calculate midpoint values. Be sure that the values you specify match the midpoint values that are calculated either by default by the GCHART procedure or by the MIDPOINTS= option. For details, see the description of MIDPOINTS= for the appropriate statement in Chapter 29, “The GCHART Procedure,” on page 773.

ORDER= overrides the suboption NUMBER= described in “Tick Mark Description Suboptions” on page 139.

ORDER= is not valid with the ASCENDING, DESCENDING and NOZEROS options used with the bar chart statements in the GCHART procedure.

This option is not supported by the GRADAR procedure.

Note: The Java applet supports ORDER= for numeric axes, but does not support ORDER= for categorical, character, midpoint, or group axes.

The ActiveX control supports only simple order lists. Non-uniform interval values, such as dates, are not supported. Only max and min values are supported with a default interval of one day. \triangle

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226, “Example 5. Filling the Area between Plot Lines” on page 236, and “Example 7. Using BY-group Processing to Generate a Series of Charts” on page 240

Not supported by: Java (partial), ActiveX (partial)

ORIGIN=(*<x><y><units>* | (*<x<units>><y<units>>*)

specifies the *x* coordinate and the *y* coordinate of the origin of the axis. The origin of the horizontal axis is the left end of the axis, and the origin of the vertical axis is the bottom of the axis. ORIGIN= explicitly positions the axis anywhere on the graphics output area.

If you specify only one value, with or without a comma following it, only the *x* coordinate is set to that value. For example, this specification sets *x* to 4 centimeters:

```
origin=(4 cm,)
```

If you specify two values, with or without a comma separating them, the first value sets the *x* coordinate and the second value sets the *y* coordinate, as in this example:

```
origin=(2 pct, 4 pct)
```

If you specify one value preceded by a comma, only the *y* coordinate is set to that value, as shown here:

```
origin=(,3 pct)
```

You can specify *units* for the *x,y* pair or for the individual coordinates. This option is not supported by the GRADAR Procedure.

Not supported by: Java, ActiveX

REFLABEL=(*text-argument(s)* | NONE

creates and defines the appearance of a reference-line label. *Text-argument(s)* defines the appearance or the text of the label, or both. NONE suppresses the reference-line label. *Text-argument(s)* can be one or more of these:

'text-string'

provides up to 256 characters of label text. By default, a reference line does not have a label. Enclose each string in quotes. Separate multiple strings with blank spaces; the strings are applied to the reference lines consecutively along the axis, from the plot origin to the end of the axis.

text-description-suboption

modifies a characteristic such as the font, color, or size of the text string(s) that follows it. *Text-description-suboption* can be

ANGLE=*degrees*

AUTOREF

COLOR=*text-color*

FONT=*font* | NONE

HEIGHT=*text-height* *<units >*

JUSTIFY=LEFT | CENTER | RIGHT

POSITION=TOP | MIDDLE | BOTTOM

ROTATE=*degrees*

T=*n*

See “Text Description Suboptions” on page 135 for a complete description.

Specify as many text strings and text description suboptions as you want, but enclose them all in one set of parentheses.

REFLABEL is not supported by the GRADAR Procedure.

Not supported by: Java, ActiveX.

SPLIT="split-char"

specifies the split character that the AXIS statement uses to break axis values into multiple lines. *Split-char* can be any character value that can be specified in a SAS character variable. The split character must be embedded in the variable values in the data set or in an associated format. When the AXIS statement encounters the split character, it automatically breaks the value at that point and continues on the next line. For example, suppose the data set contains the value **Berlin, Germany**, and you specify **SPLIT=","**. The value would appear on the axis as

```
Berlin
Germany
```

Note that the split character itself is not displayed.

Axis values specified with **VALUE=** do not use the split character. For example, suppose you specify this statement:

```
axis1 split="," value=(t1='December, 1999');
```

The value will appear on the axis on one line as **December, 1999**. However, any other axis values containing a comma would honor the split character.

This option is not supported by the GRADAR Procedure.

Featured in: Example 7 on page 856

Not supported by: Java, ActiveX

STYLE=line-type

specifies a line type for the axis line. Valid values for *line-type* are 0 through 46. If you specify **STYLE=0**, the axis line is not drawn. The default is 1, a solid line.

Note: In order for the axis *line* to be altered by the **STYLE=** option, the **NOFRAME** option must also be set. If only the **STYLE=** option is set, the axis *frame* will be modified.

Note: This option overrides the **LineStyle** attribute in graph styles. For more information on graph styles, see *SAS Output Delivery System: User's Guide* . \triangle

\triangle

Note: See also: Figure 7.22 on page 208 for examples of the available line types. \triangle

VALUE=(text-argument(s))| NONE

modifies the major tick mark values. That is, this option modifies the text that labels the major tick marks on the axis. *Text-argument(s)* defines the appearance or the text of a major tick mark value, or both. **NONE** suppresses the major tick mark values, although the major tick marks are still displayed. *Text-argument(s)* can be one or more of these:

'text-string'

provides up to 256 characters of text for the major tick mark value. By default, the value is either the variable value or an associated format value. Enclose each string in quotes and separate multiple strings with blanks.

Specified text strings are assigned to major tick marks in order. If you specify only one text string, only the first tick mark value changes, and all the other tick mark values display the default. If you specify multiple strings, the first string is the value of the first major tick mark, the second string is the value of the second major tick mark, and so forth. For example, to change default tick mark values 1, 2, and 3 to **First**, **Second**, and **Third**, use this option:

```
value=('First' 'Second' 'Third')
```

Note: Although VALUE= changes the text displayed at a major tick mark, it does not affect the actual value represented by the tick mark. To change the tick mark values, use ORDER=. Also note that with client-side rendering using Java or ActiveX, it is necessary to use ORDER= to ensure that the same number of tick marks are displayed as with server-rendered graphics. For example, specify ORDER=(1 to 12) to ensure that tick marks for all twelve months are displayed on the client.

To change the value of midpoints in bar charts produced with the GCHART procedure, use the MIDPOINTS= option in the procedure. △

text-description-suboption

modifies a characteristic such as the font, color, or size of the text string(s) that follows it. *Text-description-suboption* can be

ANGLE=*degrees*

COLOR=*text-color*

FONT=*font* | NONE

HEIGHT=*text-height* <*units*>

JUSTIFY=LEFT | CENTER | RIGHT

ROTATE=*degrees*

TICK=*n*.

For a complete description, see “Text Description Suboptions” on page 135.

Place text description suboptions before the text strings they modify.

Suboptions not followed by a text string affect the default values. To specify and describe the text for individual values or to produce multi-line text, use the TICK= suboption.

Specify as many text strings and text description suboptions as you want, but enclose them all in one set of parentheses.

Note: If an end-user viewing a graph in the Java applet or ActiveX control zooms in on a particular part of a graph for which VALUE= is specified, the values are not readjusted in coordination with the zooming. △

Featured in: “Example 2. Specifying Logarithmic Axes” on page 229, “Example 7. Using BY-group Processing to Generate a Series of Charts” on page 240, and “Example 9. Combining Graphs and Reports in a Web Page” on page 248.

Not supported by: Java (partial), ActiveX (partial)

WIDTH=*thickness-factor*

specifies the thickness of the axis line. Thickness increases directly with the value of *thickness-factor*. By default, WIDTH=1.

Note: In order for the axis *line* to be altered by the WIDTH= option, the NOFRAME option must also be set. If only the WIDTH=option is set, the axis *frame* will be modified.

Java does not support WIDTH. ACTIVEX ignores the WIDTH option for the vertical axis of an AXIS statement with GPLOT and GCONTOUR. △

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226.

Not supported by: Java, ActiveX (partial)

Text Description Suboptions

Text description suboptions are used by the LABEL=, REFLABEL=, and VALUE= options to change the color, height, justification, font, and angle of either default text or

specified text strings. See LABEL= on page 127, REFLABEL= on page 133, and VALUE= on page 134.

ANGLE=*degrees*

A=*degrees*

specifies the angle of the *baseline* with respect to the horizontal. A positive value for *degrees* moves the baseline counterclockwise; a negative value moves it clockwise. By default, ANGLE=0 (horizontal) unless the text is automatically angled or rotated to avoid overlapping. For an illustration of the effect of ANGLE=, see Figure 7.24 on page 213.

Note: Changing the angle of a vertical axis-label can result in the label being positioned above the graph when using client-side rendering with Java or ActiveX. △

See also: the ROTATE= suboption on page 138

Featured in: Example 7 on page 856 .

Not supported by: Java (partial)

AUTOREF

automatically labels each reference line on an axis with the response value at the reference line's position. AUTOREF is only used with the REFLABEL= option. The automatic labels are applied only to reference lines that do not have specific labels assigned to them. For example, the following option uses the response-axis value as the label for every reference line except the second reference line, which is assigned the label *two*:

```
reflabel=(autoref t=2 "two")
```

Note, however, that if at the same time you also request automatic labeling with a PLOT or BUBBLE statement (using the AUTOHREF or AUTOVREF option), then the automatic labeling of the PLOT or BUBBLE statement can write on top of your custom label specified with the AXIS statement. You must ensure that your custom labels specified with the AXIS statement are not at the same position as automatic labels requested with a different statement.

Not supported by: Java, ActiveX, GIF

COLOR=*text-color*

C=*text-color*

specifies the color for the text. If you omit the COLOR= suboption, a color specification is searched for in this order:

- 1 the CTEXT= option for the procedure
- 2 the CTEXT= option in a GOPTIONS statement
- 3 the default, the first color in the colors list.

FONT=*font* | NONE

F=*font* | NONE

specifies the font for the text. See Chapter 5, "SAS/GRAPH Fonts," on page 75 for details on specifying *font*. If you omit FONT=, a font specification is searched for in this order:

- 1 the FTEXT= option in a GOPTIONS statement
- 2 the default hardware font, NONE.

Not supported by: Java (partial)

HEIGHT=*text-height* <*units*>

H=*text-height* <*units*>

specifies the height of the text characters in number of units. By default, HEIGHT=1 CELL. If you omit HEIGHT=, a text height specification is searched for in this order:

- 1 the HTEXT= option in a GOPTIONS statement
- 2 the default value, 1.

JUSTIFY=LEFT | CENTER | RIGHT
J=L | C | R

specifies the alignment of the text. The default depends on the option with which it is used and the text it applies to.

- With the LABEL= option:
 - for a left vertical axis label, the default is JUSTIFY=RIGHT
 - for a right vertical axis label, the default is JUSTIFY=LEFT
 - for a horizontal axis label, the default is JUSTIFY=CENTER.

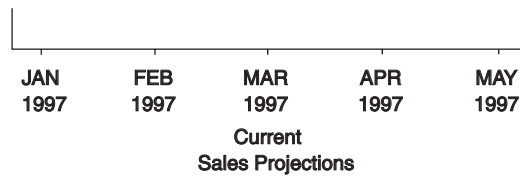
With the REFLABEL= option:

- for a vertical axis, the default is JUSTIFY=CENTER. RIGHT places the text string on the right end of the line, CENTER in the middle of the line, and LEFT to the left of the line.
- for a horizontal axis label, the default is JUSTIFY=RIGHT. RIGHT places the text string just to the right of the line, CENTER is centered on top of the line, and LEFT places the text string just to the left of the line.
- With the VALUE= option:
 - for numeric variables on a vertical axis, the default is JUSTIFY=RIGHT
 - for character variables on a vertical axis, the default is JUSTIFY=LEFT
 - for all variables on a horizontal axis, the default is JUSTIFY=CENTER.

Note: With client-side output using Java and ActiveX, text justification is relative to the text string, not the tick mark. For example, left justification means that the left end of the text string is justified with respect to the drawing location, as well as other strings in a multiline label—which may be to the right of a tick mark. Δ

You can use JUSTIFY= to print multiple lines of text by repeating JUSTIFY= before the text string for each line. You can also use JUSTIFY= to specify multi-line text at specified major tick marks. For example, this statement produces an axis label and major tick mark values like those shown in Figure 7.4 on page 138.

```
axis label=('Current' justify=c
           'Sales Projections')
value=(tick=1 'JAN' justify=c '1997'
       tick=2 'FEB' justify=c '1997'
       tick=3 'MAR' justify=c '1997'
       tick=4 'APR' justify=c '1997'
       tick=5 'MAY' justify=c '1997');
```

Figure 7.4 The JUSTIFY= suboption

Specify additional suboptions before any string.

See also: the suboption TICK= on page 138.

Not supported by: Java

POSITION=TOP | MIDDLE | BOTTOM

specifies the position of a reference-line label relative to the reference line. The default is TOP for both vertical and horizontal reference lines. POSITION= is only available on the REFLABEL= option.

- For horizontal reference lines, TOP places the label just above the reference line, MIDDLE places the label on the reference line, and BOTTOM places the label just under the reference line.
- For vertical reference lines, TOP places the label at the top end of the reference line, MIDDLE places the label in the middle of the line, and BOTTOM places the label at the bottom end of the line.

Not supported by: Java, ActiveX

ROTATE=*degrees*

R=*degrees*

specifies the angle at which *each character of text* is rotated with respect to the baseline of the text string. A positive value for *degree* rotates the character counterclockwise; a negative value moves it clockwise. By default, ROTATE=0 (parallel to the baseline) unless the text is automatically angled or rotated to avoid overlapping. For an illustration of the effect of ROTATE=, see Figure 7.31 on page 222.

See also: the suboption ANGLE= on page 136

Not supported by: Java (partial)

TICK=*n*

T=*n*

specifies the *n* reference line or tick mark value. Used only with REFLABEL= or with VALUE=. If neither one is specified, then T= is ignored.

- With REFLABEL=, T= specifies the *n*th reference line. It is used to limit modifications to individual reference lines when there are multiple reference lines on an axis. For example, the following option changes the color of only the third reference line's label and leaves all other reference-line labels unchanged:

```
reflabel=(autoref t=3 color=red)
```

Suboptions that *precede* T= affect all the reference-line labels on an axis.

Suboptions that *follow* T= affect only the specified line's label. For example, the following option assigns the color green to all the reference-line labels on an axis, but left-justifies only the third reference line's label:

```
reflabel(c=green "one" "two" t=3 j=left "three")
```


For the options to be applied to a text string, they must precede the quoted string. In the following option, the `j=left` is ignored because it follows the string:

```
reflabel(c=green "one" "two" t=3 "three" j=left)
```

□

Note: Client-side rendering with the Java applet or ActiveX control does not support the REFLABEL option. Δ

With `VALUE=`, `TICK=` specifies the *n*th major tick mark value. It is used to designate the tick mark value whose text and appearance you want to modify. For example, the following option changes the color of only the third tick mark value and leaves all others unchanged:

```
value=(tick=3 color=red)
```

Suboptions that *precede* `TICK=` affect all the major tick mark values. Suboptions that *follow* `TICK=` affect only the specified value. For example, the following option makes all the major tick mark values 4 units high and colors all of them blue except for the third one, which is red:

```
value=(height=4 color=blue tick=3 color=red)
```

Not supported by: Java, ActiveX

Using Text Description Suboptions

Text description suboptions affect all the strings that follow them unless the suboption is changed or turned off. If the value of a suboption is changed, the new value affects all the text strings that follow it. Consider this example:

```
label=(font=swiss height=4 'Weight'
        justify=right height=3 '(in tons)')
```

`FONT=SWISS` applies to both **Weight** and **(in tons)**. `HEIGHT=4` affects **Weight**, but is respecified as `HEIGHT=3` for **(in tons)**. `JUSTIFY=RIGHT` affects only **(in tons)**.

Tick Mark Description Suboptions

Tick mark description suboptions are used by `MAJOR=` and `MINOR=` to change the color, height, width, and number of the tick marks to which they apply. See `MAJOR=` and `MINOR=`.

`COLOR=tick-mark-color`

C=tick-mark color

colors the tick marks. If you omit the `COLOR=` suboption, a color specification is searched for in this order:

- 1 the `COLOR=` option in the `AXIS` statement
- 2 the `CAXIS=` option for the procedure
- 3 the default, the first color in the colors list.

`HEIGHT=tick-height <units>`

H=tick-height <units>

specifies the height of the tick mark. The defaults for the `HEIGHT=` suboption depend on the option with which it is used:

- With `MAJOR=` the default height .5 CELLS.
- With `MINOR=` the default height .25 CELLS.

If you specify a negative number, tick marks are drawn inside the axis.

Not supported by: Java, ActiveX

NUMBER=*number-of-ticks*

N=*number-of-ticks*

specifies the number of tick marks to be drawn. With MAJOR=, *number-of-ticks* must be greater than 1. With MINOR=, *number-of-ticks* must be greater than 0.

With MAJOR=, the NUMBER= suboption can be overridden by a major tick mark specification in the procedure, which in turn can be overridden by ORDER=.

With MINOR=, the NUMBER= suboption can be overridden by a minor tick mark specification in the procedure.

NUMBER= is not valid with logarithmic axes.

WIDTH=*thickness-factor*

W=*thickness-factor*

specifies the thickness of the tick mark, where *thickness-factor* is a number.

Thickness increases directly with *thickness-factor*. By default, WIDTH=1.

Not supported by: Java (partial)

Using the AXIS Statement

AXIS statements can be defined anywhere in your SAS program. They are global and remain in effect until redefined, canceled, or until the end of your SAS session. AXIS statements are not applied automatically, and must be explicitly assigned by an option in the procedure that uses them.

You can define up to 99 different AXIS statements. If you define two AXIS statements of the same number, the most recently defined one replaces the previously defined statement of the same number. An AXIS statement without a number is treated as an AXIS1 statement.

Cancel individual AXIS statements by defining an AXIS statement of the same number without options (a null statement):

```
axis4;
```

Canceling one AXIS statement does not affect any other AXIS definitions. To cancel all current AXIS statements, use the RESET= option in a GOPTIONS statement:

```
goptions reset=axis;
```

Specifying RESET=GLOBAL or RESET=ALL cancels all current AXIS definitions as well as other settings.

To display a list of current AXIS definitions in the LOG window, use the GOPTIONS procedure with the AXIS option:

```
proc goptions axis nolist;
run;
```

Assigning AXIS Definitions

AXIS definitions must always be explicitly assigned by the appropriate option in the statement that generates the graph. The following table lists the procedures and statements that generate axes, the type of axis, and the statement option that assigns an AXIS definitions to that axis:

Procedure	Statement that generates an axis	Type of axis	Option that assigns an AXIS definition
GCHART	HBAR VBAR	group axis midpoint axis response axis	GAXIS= MAXIS= RAXIS=
GCONTOUR	PLOT	horizontal axis vertical axis	HAXIS= VAXIS=
GPLOT	PLOT	horizontal axis vertical axis	HAXIS= VAXIS=
GRADAR	CHART	star axis	STARAXIS=

Some types of axes cannot use certain AXIS statement options:

- group and midpoint axes ignore LOGBASE=, MAJOR=, and MINOR=
- midpoint, horizontal and vertical axes ignore NOBRACKETS.

BY Statement

The BY statement processes data and orders output according to the BY group.

Used by:

GCHART, GCONTOUR, GMAP, GPLOT, GREDUCE, G3D, G3GRID procedures

Description

The BY statement divides the observations from an input data set into groups for processing. Each set of contiguous observations with the same value for a specified variable is called a *BY group*. A variable that defines BY groups is called a *BY variable* and is the variable that is specified in the BY statement. When you use a BY statement, the graphics procedure

- processes each group of observations independently
- generates a separate graph or output for each BY group
- automatically adds a heading called a *BY line* to each graph identifying the BY group represented in the graph
- adds BY statement information below the Description field of the catalog entry.

By default, the procedure expects the observations in the input data set to be sorted in ascending order of the BY variable values.

Note: The BY statement in SAS/GRAPH is essentially the same as the BY statement in base SAS, but the effect on the output is different when it is used with SAS/GRAPH procedures. △

Syntax

```
BY<DESCENDING>variable
  <...<DESCENDING>variable-n>
  <NOTSORTED>;
```

Required Arguments

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. By default, the procedure expects observations in the data set to be sorted in ascending order by all the variables that you specify or to be indexed appropriately.

Options

DESCENDING

indicates that the data set is sorted in descending order by the specified variable. The option affects only the variable that immediately follows the option name, and must be repeated before every variable that is not sorted in ascending order. For example, this BY statement indicates that observations in the input data set are arranged in descending order of VAR1 values and ascending order of VAR2 values:

```
by descending var1 var2;
```

This BY statement indicates that the input data set is sorted in descending order of both VAR1 and VAR2 values:

```
by descending var1 descending var2;
```

NOTSORTED

specifies that observations with the same BY value are grouped together, but are not necessarily sorted in alphabetical or numeric order. The observations can be grouped in another way, for example, in chronological order.

NOTSORTED can appear anywhere in the BY statement and affects all variables specified in the statement. NOTSORTED overrides DESCENDING if both appear in the same BY statement.

The requirement for ordering or indexing observations according to the values of BY variables is suspended when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. For NOTSORTED, the procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same value for the BY variables are not contiguous, the procedure treats each new value it encounters as the first observation in a new BY group and will create a graph for that value, even if it is only one observation.

Preparing Data for BY-Group Processing

Unless you specify the NOTSORTED option, observations in the input data set must be in ascending numeric or alphabetic order. To prepare the data set, either sort it with the SORT procedure using the same BY statement that you plan to use in the target SAS/GRAPH procedure or create an appropriate index on the BY variables.

If the procedure encounters an observation is out of the proper order, it issues an error message.

If you need to group data in some other order, such as chronological order, you can still use BY-group processing. To do so, process the data so that observations are arranged in contiguous groups that have the same BY-variable values and specify the NOTSORTED option in the BY statement.

For an example of sorting the input data set, see “Example 7. Using BY-group Processing to Generate a Series of Charts” on page 240 .

Controlling BY Lines

By default, the BY statement prints a BY line above each graph that contains the variable name followed by an equal sign and the variable value. For example, if you specify BY SITE in the procedure, the default heading when the value of SITE is **London** would be SITE=London.

Suppressing the BY line

To suppress the entire BY line, use the NOBYLINE option in an OPTION statement or specify HBYP=0 in the GOPTIONS statement. See “Example 7. Using BY-group Processing to Generate a Series of Charts” on page 240.

Suppressing the name of the BY variable

To suppress the variable name and the equal sign in the heading and leave only the BY value, use the LABEL statement to assign a null label ('00'X) to the BY variable. For example, this statement assigns a null label to the SITE variable:

```
label site='00'x;
```

See also GCHLEGNDEExample 12 on page 877.

Controlling the appearance of the BY line

To control the color, font, and height of the BY lines, use the following graphics options in a GOPTIONS statement:

CBY=BY-line-color
specifies the color for BY lines.

FBY=font
specifies the font for BY lines.

HBYP=n<units>
specifies the height for BY lines.

See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for a complete description of each option.

Naming the Catalog Entries

The catalog entries generated with BY-group processing always use incremental naming. This means that the first entry created by the procedure uses the base name and subsequent entries increment that name. The base name is either the default entry name for the procedure (for example, GPLOT) or the name specified with the NAME= option in the action statement. Incrementing the base name automatically appends a number to each subsequent entry (for example, GPLOT1, GPLOT2, and so forth). See also “Names and Descriptions of Catalog Entries” on page 55 and “Using the default output name” on page 63. For an example of incremented catalog names, see “Example 9. Combining Graphs and Reports in a Web Page” on page 248.

Using the BY Statement

This section describes the following:

- the effect of BY-group processing on the GCHART, GMAP, and GPLOT procedures
- the interaction between BY-group and RUN-group processing
- the requirements for using BY-group processing with the Annotate facility
- how to include BY information in titles, notes, and footnotes
- how patterns and symbols are assigned to BY-groups
- the effect of using BY-group processing with the ODS HTML statement.

For additional information on any of these topics, refer to the appropriate chapter.

With the GCHART Procedure

When you use BY-group processing with the GCHART procedure, you can do the following:

- With the BLOCK, HBAR, and VBAR statements, you can use the PATTERNID=BY option to assign patterns according to BY groups. With PATTERNID=BY, each BY group uses a different PATTERN definition, but all bars or blocks within a BY group use the same pattern.
- With the BLOCK statement, you can use the BLOCKMAX= option to produce the same block-height scaling in all block charts in a BY group.
- With the HBAR or VBAR statement, you can use the RAXIS= option to produce the same response axis scaling in all horizontal or vertical bar charts in a BY group.

With the PIE and STAR statements, the effect of a BY statement is similar to that of the GROUP= option, except that the GROUP= option allows you to put more than one graph on a single page while the BY statement does not. Do not use a BY variable as the group variable in STAR or PIE statements.

With the GMAP Procedure

By default, BY-group processing affects both the map data set and the response data set. This means that you get separate, individual output for each map area common to both data sets. For example, if the map data set REGION contains six states and the response data set contains the same six states, and you specify BY STATE in the GMAP procedure, you get six graphs with one state on each graph.

If you use the ALL option in the PROC GMAP statement and you also use the BY statement, you get one output for each map area in the response data set, but that output displays all the map areas in the map data set. Only one map area per output contains response data information; the others are empty. For example, if you create a block map using the data sets REGION and SALES, specify BY STATE, and include the ALL option in the PROC GMAP statement, you get six graphs with six states on each graph. One state per graph has a block; the remaining five are empty.

With the GPLOT Procedure

You can use the UNIFORM option in the PROC GPLOT statement to produce the same axis scaling for all graphs in a BY group. By default, the range of the axes may vary from graph to graph, but UNIFORM forces the scaling to be the same for all graphs generated by the procedure.

With the RUN Groups

If you use the BY statement with a procedure that processes data and supports RUN-group processing (the GCHART, GMAP, and GPLOT procedures), then each time you submit an action statement or a RUN statement you get a separate graph for each

value of the BY variable. For example, each of these two RUN-groups produces a separate plot for every value of the BY variable SITE:

```
/* first run group*/
proc gplot data=sales;
  title1 'Sales Summary';
  by site;
  plot sales*model_a;
run;

/* second run group */
plot sales*model_b;
run;
quit;
```

The BY statement stays in effect for every subsequent RUN group until you submit another BY statement or exit the procedure. Variables in subsequent BY statements replace any previous BY variables.

You can also turn off BY-group processing by submitting a null BY statement (BY;) in a RUN group, but when you do this, the null BY statement turns off BY-group processing *and* the RUN group generates a graph.

For more information, see “RUN-Group Processing” on page 33.

With the Annotate Facility

If a procedure that is using BY-group processing also specifies annotation with the ANNOTATE= option in the PROC statement, the same annotation is applied to every graph generated by the procedure.

If you specify annotation with the ANNOTATE= option in the action statements for a procedure, the BY-group processing is applied to the Annotate data set. In this way, you can customize the annotation for the output from each BY group by including the BY variable in the Annotate data set and by using each BY-variable value as a condition for the annotation to be applied to the output for that value.

With TITLE, FOOTNOTE, and NOTE Statements

TITLE, FOOTNOTE, and NOTE statements can automatically include the BY variable name, BY variable values, or BY lines in the text they produce. To insert BY variable information into the text strings used by these statements, use the #BYVAR, #BYVAL, and #BYLINE substitution options. For details, see the description of the text-string argument on page 222. For an example, see “Example 7. Using BY-group Processing to Generate a Series of Charts” on page 240.

With PATTERN and SYMBOL Definitions

Procedures that use SYMBOL or PATTERN definitions, assign the symbols or patterns in order to each BY group. For example, if the BY variable REGION has four values – **East**, **North**, **South**, and **West** – the patterns are assigned to the BY-groups in this order:

```
PATTERN1 is assigned to East,
PATTERN2 is assigned to North,
PATTERN3 is assigned to South,
PATTERN4 is assigned to West.
```

If you create sets of graphs from several data sets containing the variable REGION, and if you want the same pattern assigned to the same region each time, you must be

sure that REGION always has the same four values. Otherwise, the patterns may not be the same across graphs. For example, if the value **North** is missing from the data, the patterns are assigned as follows:

PATTERN1 is assigned to **East**,
 PATTERN2 is assigned to **South**,
 PATTERN3 is assigned to **West**.

In this case, **South** is assigned pattern 2 instead of pattern 3 and **West** is assigned pattern 3 instead of pattern 4. To avoid this, include the value **North** for the variable REGION, but assign it a missing value for all other variables.

FOOTNOTE Statement

See “TITLE, FOOTNOTE, and NOTE Statements” on page 210.

GOPTIONS Statement

The GOPTIONS statement temporarily sets default values for many graphics attributes and device parameters used by SAS/GRAPH procedures.

Used by:
 all statements and procedures in a SAS session

Description

The GOPTIONS statement specifies values for *graphics options*. Graphics options control characteristics of the graph, such as size, colors, type fonts, fill patterns, and symbols. In addition, they affect the settings of device parameters, which are defined in the device entry. Device parameters control such characteristics as the appearance of the display, the type of output produced, and the destination of the output.

The GOPTIONS statement allows you to change these settings temporarily, either for a single graph or for the duration of your SAS session. You can use the GOPTIONS statement to

- override default values for graphics options that control either graphics attributes or device parameters for a single graph or for an entire SAS session
- reset individual graphics options or all graphics options to their default values
- cancel definitions for AXIS, FOOTNOTE, PATTERN, SYMBOL, and TITLE statements.

To change device parameters permanently, you must use the GDEVICE procedure to modify the appropriate device entry or to create a new one. See Chapter 31, “The GDEVICE Procedure,” on page 915 for details.

To review the current settings of all graphics options, use the GOPTIONS procedure. See Chapter 36, “The GOPTIONS Procedure,” on page 1075 for details.

Syntax

GOPTIONS<options-list>;

options-list can be one or more options from any or all of the following categories:

- reset option

- RESET=ALL | GLOBAL | *statement-name* | (*statement-name(s)*)
- options that affect the appearance of the display area and the graphics output
 - ASPECT=*scaling-factor*
 - AUTOSIZE=ON | OFF | DEFAULT
 - BORDER | NOBORDER
 - CELL | NOCELL
 - GSIZE=*lines*
 - HORIGIN=*horizontal-offset* <IN | CM>
 - HPOS=*columns*
 - HSIZE=*horizontal-size* <IN | CM>
 - IBACK= *fileref* | '*external-file*'
 - IMAGESTYLE = TILE | FIT
 - IMAGEPRINT | NOIMAGEPRINT
 - ROTATE=LANDSCAPE | PORTRAIT
 - ROTATE | NOROTATE
 - SWAP | NOSWAP
 - TARGETDEVICE=*target-device-entry*
 - VORIGIN=*vertical-offset* <IN | CM>
 - VPOS=*rows*
 - VSIZE=*vertical-size* <IN | CM>
 - XMAX=*width* <IN | CM>
 - XPIXELS=*width-in-pixels*
 - YMAX=*height* <IN | CM>
 - YPIXELS=*height-in-pixels*
- options that affect color
 - CBACK=*background-color*
 - CBY=*BY-line-color*
 - COLORS=<(colors-list | NONE)>
 - CPATTERN=*pattern-color*
 - CSYMBOL=*symbol-color*
 - CTEXT=*text-color*
 - CTITLE=*title-color*
 - PENMOUNTS=*active-pen-mounts*
 - PENSORT | NOPENSORT
- options that control font selection or text appearance
 - CHARTYPE=*hardware-font-chartype*
 - FASTTEXT | NOFASTTEXT
 - FBY=*BY-line-font*
 - FCACHE=*number-fonts-open*
 - FONTRÉS=NORMAL | PRESENTATION
 - FTEXT=*text-font*
 - FTITLE=*title-font*
 - FTRACK=LOOSE | NONE | NORMAL | TIGHT | TOUCH | V5
 - HBY=*BY-line-height* <units>
 - HTEXT=*text-height* <units>

- HTITLE=*title-height* <units>
 RENDER=APPEND | DISK | MEMORY | NONE | READ
 RENDERLIB=*libref*
 SIMFONT=*software-font*
- options that set defaults for procedures and global statements
 - GUNIT=*units*
 - INTERPOL=*interpolation-method*
 - OFFSHADOW=(*x* <units>, *y* <units> | (*x,y*) <units>
 - V6COMP | NOV6COMP
 - image animation options
 - DELAY=*delay-time*
 - DISPOSAL=NONE | BACKGROUND | PREVIOUS | UNSPECIFIED
 - INTERLACED | NONINTERLACED
 - ITERATION=*iteration-count*
 - TRANSPARENCY | NOTRANSPARENCY
 - options that affect how your SAS/GRAPH program runs
 - DISPLAY | NODISPLAY
 - ERASE | NOERASE
 - GWAIT=*seconds*
 - GRAPHRC | NOGRAPHRC
 - IMAGEPRINT | NOIMAGEPRINT
 - PCLIP | NOPCLIP
 - POLYGONCLIP | NOPOLYGONCLIP
 - options that control how output is sent to devices or files
 - ADMGDF | NOADMGDF
 - DEVADDR=*device-address*
 - DEVICE=*device-entry*
 - DEVMAP=*device-map-name* | NONE
 - EXTENSION=*'file-type'*
 - FILECLOSE=DRIVERTERM | GRAPHEND
 - FILEONLY | NOFILEONLY
 - GACCESS=*output-format* | *'output-format > destination'*
 - GEND=*'string'* <...*'string-n'*>
 - GPEILOG=*'string'* <...*'string-n'*>
 - GOUTMODE=APPEND | REPLACE
 - GPROLOG=*'string'* <...*'string-n'*>
 - GPROTOCOL=*module-name*
 - GSFLEN=*record-length*
 - GSFMODE=APPEND | PORT | REPLACE
 - GSFNAME=*fileref*
 - GSFPROMPT | NOGSFPROMPT
 - GSTART=*'string'* <...*'string-n'*>
 - HANDSHAKE=HARDWARE | NONE | SOFTWARE | XONXOFF
 - KEYMAP=*map-name* | NONE
 - POSTGPEILOG=*'string'*

- POSTGPROLOG=*'string'*
 PREGPROLOG=*'string'*
 PREGPROLOG=*'string'*
 PROMPTCHARS=*'prompt-chars-hex-string'*X
- options that specify hardware capabilities of the device
 - CHARACTERS | NOCHARACTERS
 - CIRCLEARC | NOCIRCLEARC
 - DASH | NODASH
 - DASHSCALE=*scaling-factor*
 - FILL | NOFILL
 - FILLINC=0...9999
 - LFACTOR=*line-thickness-factor*
 - PIEFILL | NOPIEFILL
 - POLYGONFILL | NOPOLYGONFILL
 - SYMBOL | NOSYMBOL
 - options that control printer hardware features
 - AUTOCOPY | NOAUTOCOPY
 - AUTOFEED | NOAUTOFEED
 - BINDING=DEFAULTEDGE | LONGEDGE | SHORTEGE
 - COLLATE | NOCOLLATE
 - DUPLEX | NODUPLEX
 - GCOPIES=(*<current-copies>*,*<max-copies>*)
 - PAPERDEST=*bin*
 - PAPERFEED=*feed-increment* <IN | CM>
 - PAPERLIMIT=*width* <IN | CM>
 - PAPERSIZE=*'size-name'* | (*width,height*)
 - PAPERSOURCE=*tray*
 - PAPERTYPE=*'type-name'*
 - PPDFILE=*fileref* | *'external-file'*
 - REPAINT=*redraw-factor*
 - REVERSE | NOREVERSE
 - SPEED=*pen-speed*
 - UCC=*'control-characters-hex-string'*X
 - options that interact with the operating environment
 - DRVINIT=*'system-command(s)'*
 - DRVTERM=*'system-command(s)'*
 - PREGRAPH=*'system-command(s)'*
 - POSTGRAPH=*'system-command(s)'*
 - PROMPT | NOPROMPT
 - options for mainframe systems
 - GCLASS=SYSOUT-*class*
 - GDDMCOPY=FSCOPY | GSCOPY
 - GDDMNICKNAME=*nickname*
 - GDDMTOKEN=*token*
 - GDEST=*destination*

```
GFORMS='forms-code'
GWRITER='writer-name'
TRANTAB=table | user-defined-table
```

Options

See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for a complete description of all graphics options used by the GOPTIONS statement.

Using the GOPTIONS Statement

GOPTIONS statements are global and can be located anywhere in your SAS program. However, for the graphics options to affect the output from a procedure, the GOPTIONS statement must execute before the procedure.

With the exception of RESET=, graphics options can be listed in any order in a GOPTIONS statement. RESET= should be the first option in the GOPTIONS statement.

A graphics option remains in effect until you either specify the option in another GOPTIONS statement, or use RESET= to reset the values, or end the SAS session. When a session ends, the values of the graphics options return to their default values.

Graphics options are additive; that is, the value of a graphics option remains in effect until the graphics option is explicitly changed or reset or until you end your SAS session. Graphics options remain in effect even after you submit additional GOPTIONS statements specifying different options.

To reset an individual option to its default value, submit the option without a value (a null graphics option.) You can use a comma (but it is not required) to separate a null graphics option from the next one. For example, this GOPTIONS statement sets the values for background color, text height, and text font:

```
options cback=blue htext=6 pct ftext=zapf;
```

To reset only the background color specification to the default and keep the remaining values, use this GOPTIONS statement:

```
options cback=;
```

To reset all graphic options to their default values, specify RESET=GOPTIONS:

```
options reset=options;
```

Alternatively, you can use RESET=ALL, but it also cancels any global statement definitions in addition to resetting all graphics options to default values.

Graphics Option Processing

You can control many graphics attributes through statement options, graphics options, device parameters, or a combination of these. SAS/GRAPH searches these places to determine the value to use, stopping at the first place that gives it an explicit value:

- 1 statement options
- 2 the value of the corresponding graphics option
- 3 the value of a device parameter found in the catalog entry for your device driver.

Note: Not every graphics attribute can be set in all three places. See the statement and procedure chapters for the options that can be used with each. \triangle

Some graphics options are supported for specific devices or operating environments only. See the SAS Help facility for SAS/GRAPH or the SAS companion for your operating environment for more information.

LEGEND Statement

The LEGEND statement controls the location and appearance of legends on two-dimensional plots, contour plots, maps, and charts.

Used by:

GCHART, GCONTOUR, GMAP, GPLOT procedures

Global

Description

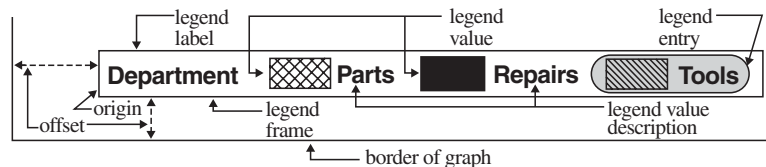
LEGEND statements specify the characteristics of a legend but do not create legends. These characteristics are

- the position and appearance of the legend box
- the text and appearance of the legend label
- the appearance of the legend entries, including the size and shape of the legend values
- the text of the labels for the legend values.

LEGEND definitions are not automatically applied when a procedure generates a legend. Instead, they must be explicitly assigned with a LEGEND= option in the appropriate procedure statement.

illustrates the terms associated with the various parts of a legend.

Figure 7.5 Parts of a Legend



Syntax

LEGEND<1...99><options>;

option(s) can be one or more options from any or all of the following categories:

- appearance options
 - ACROSS=*number-of-columns*
 - CBLOCK=*block-color*
 - CBORDER=*frame-color*
 - CFRAME=*background-color*
 - CSHADOW=*shadow-color*
 - DOWN=*number-of-rows*
 - FRAME

FWIDTH=thickness-factor

*SHAPE=BAR(width,height) <units> | LINE(length) <units> |
SYMBOL(width,height) <units>*

□ position-options

MODE=PROTECT | RESERVE | SHARE

OFFSET=(<x ><y ><units > | (<x <units >><y <units >>)

ORIGIN=(<x ><y ><units > | (<x <units >><y <units >>)

*POSITION=(<BOTTOM | MIDDLE | TOP> <LEFT | CENTER | RIGHT>
<INSIDE | OUTSIDE>)*

□ text-options

LABEL=(text-argument(s)) | NONE

ORDER=(value-list)

VALUE=(text-argument(s)) | NONE

Options

When the syntax of an option includes *units*, use one of these:

CELLS	character cells
CM	centimeters
IN	inches
PT	points
PCT	percentage of the graphics output area

Note: The Java applet does not support CM, IN, or PT. \triangle

If you omit *units*, a unit specification is searched for in this order:

- 1 GUNIT= in a GOPTIONS statement
- 2 the default unit, CELLS.

ACROSS=number-of-columns

specifies the number of columns to use for legend entries.

Featured in: “Example 8. Creating a Simple Web Page with the ODS HTML Statement” on page 245.

CBLOCK=block-color

generates and colors a three-dimensional block effect behind the legend. The size and position of the block are controlled by the graphics option OFFSHADOW=(*x,y*).

CBLOCK= and CSHADOW= are mutually exclusive. If both are present, SAS/GRAPH software uses the last one specified. CBLOCK= is usually used in conjunction with the FRAME, CFRAME=, or CBORDER= options.

The Java client treats the CBLOCK option like CSHADOW.

See also: the OFFSHADOW=“OFFSHADOW” on page 325 graphics option and “Creating Drop Shadows and Block Effects” on page 163.

Not supported by: Java

CBORDER=frame-color

draws a colored frame around the legend. This option overrides the FRAME option. CBORDER= can be used in conjunction with the CFRAME= option.

CFRAME=background-color

specifies the background color of the legend. This option overrides the FRAME option. If both CFRAME= and FRAME= are specified, only the solid background produced by CFRAME= is displayed. CFRAME= can be used in conjunction with the CBORDER= option.

CSHADOW=*shadow-color*

generates and colors a drop shadow behind the legend. The size and position of the shadow is controlled by the graphics option OFFSHADOW=(*x,y*).

CSHADOW= and CBLOCK= are mutually exclusive. If both are present, SAS/GRAPH uses the last one specified. CSHADOW= is usually specified in conjunction with the FRAME, CFRAME=, or CBORDER= options.

See also: the OFFSHADOW="OFFSHADOW" on page 325 graphics option and "Creating Drop Shadows and Block Effects" on page 163.

DOWN=*number-of-rows*

specifies the number of rows to use for legend entries.

FRAME

draws a frame around the legend. The color of the frame is the first color in the colors list.

FWIDTH=*thickness-factor*

specifies the thickness of the frame, where *thickness-factor* is a number. The thickness of the line increases directly with *thickness-factor*. By default, FWIDTH=1.

Not supported by: Java, ActiveX

LABEL=(*text-argument(s)*) | NONE

modifies a legend label. *Text-argument(s)* defines the appearance or the text of a legend label, or both. NONE suppresses the legend label. By default, the text of the legend label is either the variable name or a previously assigned variable label (except in the case of GPLOT with OVERLAY, in which case the default label is "PLOT"). *Text-argument(s)* can be one or more of these:

'text-string'

provides up to 256 characters of label text. Enclose each string in quotes. Separate multiple strings with blanks.

text-description-suboption

modifies a characteristic such as the font, color, or size of the text string(s) that follows it. *Text-description-suboption* can be

COLOR=*text-color*

FONT=*font* | NONE

HEIGHT=*text-height* <*units*>

JUSTIFY=LEFT | CENTER | RIGHT

POSITION=(<BOTTOM | MIDDLE | TOP> <LEFT | CENTER | RIGHT>)

Note: The Java applet does not support the POSITION suboption—it draws legend labels at the top-left of the legend. And, it does not support multiple values for the JUSTIFY suboption (only the first is honored). The ActiveX control supports POSITION= but does not support multiple values for the JUSTIFY suboption (only the first is honored). △

See "Text Description Suboptions" on page 158 for complete descriptions. Specify as many text strings and text description suboptions as you want, but enclose them all in one set of parentheses.

Featured in: “Example 3. Rotating Plot Symbols through the Colors List” on page 231 and “Example 8. Creating a Simple Web Page with the ODS HTML Statement” on page 245

Not supported by: Java (partial), ActiveX (partial)

MODE=PROTECT | RESERVE | SHARE

specifies whether or not the legend is drawn in the procedure output area or whether legend elements can overlay other graphics elements. MODE= can take one of these values:

PROTECT draws the legend in the procedure output area, but a *blanking area* surrounds the legend, preventing other graphics elements from being displayed in the legend. (A blanking area is a protected area in which no other graphics elements are displayed.)

RESERVE takes space for the legend from the procedure output area, thereby reducing the amount of space available for the graph. If MODE=RESERVE is specified in conjunction with OFFSET=, the legend may push the graph off the graphics output area. RESERVE is valid only when POSITION=OUTSIDE. If POSITION=INSIDE is specified, a warning is issued and MODE= is changed to PROTECT.

SHARE draws the legend in the procedure output area. If the legend is positioned over elements of the graph itself, both graphics elements and legend elements are displayed.

By default, MODE=RESERVE unless POSITION=INSIDE, in which case the default changes to MODE=PROTECT.

See also: “Positioning the Legend” on page 162

Featured in: “Example 8. Creating a Simple Web Page with the ODS HTML Statement” on page 245.

Not supported by: Java, ActiveX

OFFSET=($\langle x \rangle \langle y \rangle \langle units \rangle$ | ($\langle x \langle units \rangle \rangle \langle y \langle units \rangle \rangle$)

specifies the distance to move the entire legend; x is the number of units to move the legend right (positive numbers) or left (negative numbers), and y is the number of units to move the legend up (positive numbers) or down (negative numbers).

To set only the x offset, specify one value, with or without a following comma:

```
offset=(4 cm,)
```

To set both the x and y offset, specify two values, with or without a comma separating them:

```
offset=(2 pct, 4 pct)
```

To set only the y offset, specify one value preceded by a comma:

```
offset=(,-3 pct)
```

OFFSET= is usually used in conjunction with POSITION= to adjust the position of the legend. Moves are relative to the location specified by POSITION=, with OFFSET=(0,0) representing the initial position. You can also apply OFFSET= to the default legend position.

OFFSET= is unnecessary with ORIGIN= since ORIGIN= explicitly positions the legend and requires no further adjustment. However, if you specify both options, the values of OFFSET= are added to the values of ORIGIN=, and the LEGEND is positioned accordingly.

See also: “Positioning the Legend” on page 162 and the option POSITION= on page 155

Not supported by: Java, ActiveX

ORDER=(*value-list*)

selects or orders the legend values that appear in the legend. The way you specify *value-list* depends on the type of variable that generates the legend:

- For numeric variables, *value-list* is either an explicit list of values, or a starting and an ending value with an interval increment, or a combination of both forms:

n <...*n*>

n TO *n* <BY *increment*>

n <...*n*> TO *n* <BY *increment*> <*n* <...*n*>>

If a numeric variable has an associated format, the specified values must be the *unformatted* values.

- For character variables, *value-list* is a list of unique character values enclosed in quotes and separated by blanks:

'*value-1*' <...'value-*n*'>

If a character variable has an associated format, the specified values must be the *formatted* values.

For a complete description of *value-list*, see the option ORDER= on page 130 in the AXIS statement.

Even though ORDER= controls whether a legend value is displayed and where it appears, the VALUE= option controls the text that the legend value displays.

Not supported by: ActiveX, Java

ORIGIN=(<*x*><*y*><*units*> | (<*x* <*units*>><*y* <*units*>>)

specifies the *x* coordinate and the *y* coordinate of the lower-left corner of the legend box. ORIGIN= explicitly positions the legend anywhere on the graphics output area. It is possible to run a legend off the page or overlay the graph.

To set only the *x* coordinate, specify one value, with or without a following comma:

```
origin=(4 cm,)
```

To set both the *x* and *y* coordinates, specify two values, with or without a comma separating them:

```
origin=(2 pct, 4 pct)
```

To set only the *y* coordinate, specify one value preceded by a comma:

```
origin=(,3 pct)
```

ORIGIN= overrides the POSITION= option if both are used. Although using the OFFSET= option with the ORIGIN= option is unnecessary, if OFFSET= is also specified, it is applied after the ORIGIN= request has been processed.

See also: “Positioning the Legend” on page 162

Featured in: “Example 8. Creating a Simple Web Page with the ODS HTML Statement” on page 245 .

Not supported by: Java, ActiveX

POSITION=(<BOTTOM | MIDDLE | TOP> <LEFT | CENTER | RIGHT>
<OUTSIDE | INSIDE>)

positions the legend on the graph. Value for POSITION= are

OUTSIDE or INSIDE specifies the location of the legend in relation to the axis area.

BOTTOM or MIDDLE or TOP specifies the vertical position.

LEFT or CENTER or RIGHT specifies the horizontal position.

By default, POSITION=(BOTTOM CENTER OUTSIDE). You can change one or more settings. If you supply only one value the parentheses are not required. If you specify two or three values and omit the parentheses, SAS/GRAPH accepts the first value and ignores the others.

Once you assign the initial legend position, you can adjust it with the OFFSET= option.

The ORIGIN= options overrides POSITION=. The value of the MODE= option can affect the behavior of POSITION=.

Note: The Java applet defaults to BOTTOM-CENTER and supports all possible combinations of BOTTOM | MIDDLE | TOP with LEFT | CENTER | RIGHT except for MIDDLE-CENTER (which would overwrite the map.) The Java applet does not support INSIDE for positioning. \triangle

See also: information on positioning the Legend in the *SAS/GRAPH Reference, Volumes 1 and 2*, the options OFFSET= on page 154, and MODE= on page 154.

Not supported by: Java (partial)

SHAPE=BAR(*width*<units>,<height><units>) <units> | LINE(*length*) <units> | SYMBOL(*width*<units>,<height><units>) <units>

specifies the size and shape of the legend values displayed in each legend entry. The value you specify for SHAPE= depends on which procedure generates the legend.

BAR(*width,height*)<units>

is used with the GCHART and GMAP procedures, the GPLOT procedure if you use the AREAS= option, and the GCONTOUR procedure if you use the PATTERN option. Each legend value is a bar of the specified width and height. By default, *width* is 5, *height* is 0.8, and *units* are CELLS. You can specify *units* for the *width,height* pair or for the individual coordinates.

Featured in: “Example 3. Rotating Plot Symbols through the Colors List” on page 231 and “Example 8. Creating a Simple Web Page with the ODS HTML Statement” on page 245.

LINE(*length*) <units>

is used with the GPLOT and GCONTOUR procedures. Each legend value is a line of the length you specify. Plotting symbols are omitted from the legend values. By default, *length* is 5 and *units* are CELLS. You can specify *units* for *length*.

SYMBOL(*width*<units>,<height><units>) <units>

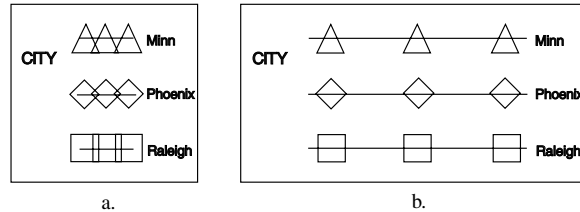
is used with the GPLOT procedure. Each legend value (*not* each symbol) is the width and height you specify. For example, this specification produces legend values like the ones in Figure 7.6 on page 157(a):

```
shape=symbol(.5,.5)
```

This specification produces legend values like the ones in Figure 7.6 on page 157(b):

```
shape=symbol(2,.5)
```

Figure 7.6 Legend Values Produced with SHAPE= SYMBOL



By default, *width* is 5, *height* is 1, and *units* are CELLS. You can specify *units* for the *width,height* pair or for the individual coordinates.

Featured in: “Example 3. Rotating Plot Symbols through the Colors List” on page 231.

Not supported by: Java, ActiveX

VALUE=(*text-argument(s)*) | NONE

modifies the legend value descriptions. *Text-argument(s)* defines the appearance or the text of the value descriptions. By default, value descriptions are the values of the variable that generates the legend or an associated format value. Numeric values are right justified and character values are left justified.

NONE suppresses the value descriptions although the legend values (bars, lines, and so on) are still displayed. (NONE is not supported by Java or ActiveX). *Text-argument(s)* can be one or more of these:

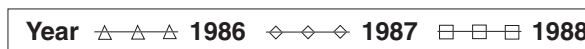
'text-string'

provides up to 256 characters of text for the value description. Enclose each string in quotes. Separate multiple strings with blanks.

Specified text strings are assigned to the legend values in order. If you submit only one string, only the first legend entry uses the value of that string. If you specify multiple strings, the first string is the text for the first entry; the second string is the text for the second entry; and so forth. For example, this specification produces legend entries like those shown in Figure 7.7 on page 157:

```
value=('1986' '1987' '1988')
```

Figure 7.7 Specifying Value Descriptions with the VALUE= Option



text-description-suboption

modifies a characteristic such as the font, color, or size of the text string(s) that follows it. *Text-description-suboption* can be

COLOR=*text-color*

FONT=*font* | NONE

HEIGHT=*text-height* <*units*>

JUSTIFY=LEFT | CENTER | RIGHT

TICK=*n* (not supported by Java or ActiveX)

See “Text Description Suboptions” on page 158 for complete descriptions.

Place text description suboptions before the text strings they modify.

Suboptions not followed by a text string affect the default values. To specify and describe the text for individual values or to produce multi-line text, use the TICK= suboption.

Specify as many text strings and text description suboptions as you want, but enclose them all in one set of parentheses.

To order or select legend entries, use the ORDER= option.

See also: “Text Description Suboptions” on page 158 and the option ORDER= on page 155

Not supported by: Java (partial), ActiveX (partial)

Text Description Suboptions

Text description suboptions are used by the LABEL= and VALUE= options to change the color, height, justification, font, and angle of either default text or specified text strings. See LABEL= on page 153 and VALUE= on page 157.

COLOR=*text-color*

C=*text-color*

specifies the color of the text. If you omit COLOR=, a color specification is searched for in this order:

- 1 the CTEXT= option for the procedure
- 2 the CTEXT= option in a GOPTIONS statement
- 3 the default, the first color in the colors list.

FONT=*font* | NONE

F=*font* | NONE

specifies the font for the text. See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for information on specifying fonts. If you omit FONT=, a font specification is searched for in this order:

- 1 the FTEXT= option in a GOPTIONS statement
- 2 the default hardware font, NONE.

HEIGHT=*text-height* <*units*>

H=*text-height* <*units*>

specifies the height of the text characters in the number of units. By default, HEIGHT=1 CELL. If you omit HEIGHT=, a text height specification is searched for in this order:

- 1 the HTEXT= option in a GOPTIONS statement
- 2 the default value, 1.

JUSTIFY=LEFT | CENTER | RIGHT

J=L | C | R

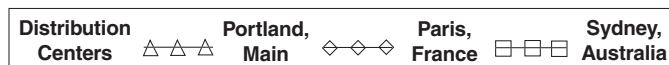
specifies the alignment of the text. The default for character variables is JUSTIFY=LEFT. The default for numeric variables is JUSTIFY=RIGHT.

Associating a character format with a numeric variable does not change the default justification of the variable.

You can use JUSTIFY= to print multiple lines of text by repeating JUSTIFY= before the text string for each line. For example, this statement produces a legend label and value descriptions like those shown in Figure 7.8 on page 159:

```
legend label=(justify=c 'Distribution'
              justify=c 'Centers')
value=(tick=1 justify=c 'Portland,'
       justify=c 'Maine'
       tick=2 justify=c 'Paris,'
       justify=c 'France'
       tick=3 justify=c 'Sydney,'
       justify=c 'Australia');
```

Figure 7.8 Specifying Multiple Lines of Text with the JUSTIFY= Suboption



Specify additional suboptions before any string.

See also: the suboption TICK= on page 160.

POSITION=(<BOTTOM | MIDDLE | TOP> <LEFT | CENTER | RIGHT>)

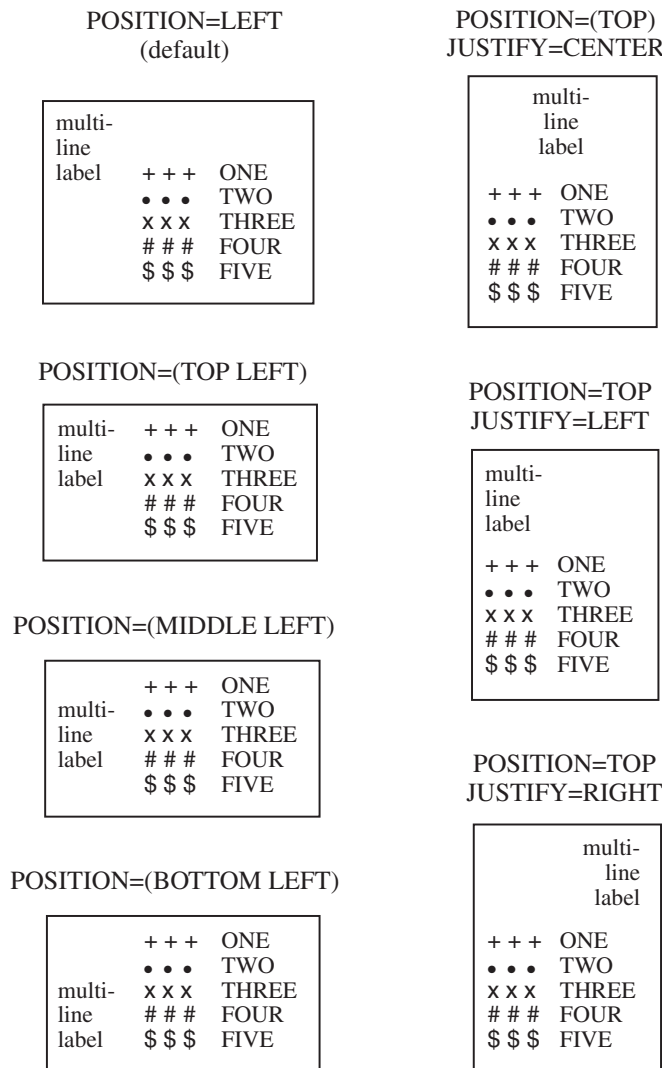
places the legend label in relation to the legend entries. The POSITION= suboption is used only with the LABEL= option. By default, POSITION=LEFT.

The parentheses are not required if only one value is supplied. If you specify two or three values and omit the parentheses, SAS/GRAPH accepts the first value and ignores the others.

Figure 7.9 on page 160 shows some of the ways POSITION= affects a multi-line legend label in which the entries are stacked in a column (ACROSS=1). This figure uses a label specification such as

```
label=('multi-'
      justify=left 'line'
      justify=left 'label'
      position=left)
```

In this specification, POSITION= specifies the default value, LEFT, which is represented by the first legend in the figure. The value of POSITION= is indicated above each legend. The default justification is used unless you also use the JUSTIFY= suboption.

Figure 7.9 Using the POSITION= Suboption with Multi-line Legend Labels

In addition, specifying POSITION=RIGHT mirrors the effect of POSITION=LEFT, and specifying POSITION=BOTTOM mirrors the effect of POSITION=TOP.

Not supported by: Java

TICK=*n*

T=*n*

specifies the *n*th legend entry. The TICK= suboption is used only with the VALUE= option to designate the legend entry whose text and appearance you want to modify. For example, to change the text of the third legend entry to **Minneapolis**, specify

```
value=(tick=3 'Minneapolis')
```

The characteristics of all other value descriptions remain unchanged.

If you use TICK= when you designate text for one legend entry, you must also use it when you designate text for any additional legend entries. For example, this option changes the text of both the second and third legend entries:

```
value=(tick=2 'Paris' tick=3 'Sydney')
```

If you omitted TICK=3, the text of the second legend entry would be **ParisSydney**.

Text description suboptions that *precede* TICK= affect all the value descriptions for the legend unless the same suboption (with a different value) follows a TICK= specification. Text description suboptions that *follow* TICK= affect only the specified legend entry. For example, suppose you specify this option for a legend with three entries:

```
value=(color=red font=swiss tick=2 color=blue)
```

The text of all three entries would use the Swiss font; the first and third entries would be red and only the second entry would be blue.

Using Text Description Suboptions

Text description suboptions affect all the strings that follow them unless the suboption is changed or turned off. If the value of a suboption is changed, the new value affects all the text strings that follow it. Consider this example:

```
label=(font=swiss height=4 'Weight'
       justify=right height=3 '(in tons)')
```

FONT=SWISS applies to both **Weight** and **(in tons)**. HEIGHT=4 affects **Weight**, but is respecified as HEIGHT=3 for **(in tons)**. JUSTIFY=RIGHT affects only **(in tons)**.

Using the LEGEND Statement

LEGEND statements can be located anywhere in your SAS program. They are global and remain in effect until canceled or until you end your SAS session. LEGEND statements are not applied automatically, and must be explicitly assigned by an option in the procedure that uses them.

You can define up to 99 different LEGEND statements. If you define two LEGEND statements of the same number, the most recently defined one replaces the previously defined statement of the same number. A LEGEND statement without a number is treated as a LEGEND1 statement.

Cancel individual LEGEND statements by defining a LEGEND statement of the same number without options (a null statement):

```
legend4;
```

Canceling one LEGEND statement does not affect any other LEGEND definitions. To cancel all current LEGEND statements, use RESET= in a GOPTIONS statement:

```
goptions reset=legend;
```

Specifying RESET=GLOBAL or RESET=ALL cancels all current LEGEND definitions as well as other settings.

To display a list of current LEGEND definitions in the LOG window, use the GOPTIONS procedure with the LEGEND option:

```
proc goptions legend nolist;
run;
```

Positioning the Legend

By default, the legend shares the procedure output area with the procedure output, such as a map or bar chart. (See “Placement of Graphic Elements in the Graphics Output Area” on page 39.) However, several LEGEND statement options allow you to position a legend anywhere on the graphics output area and even to overlay the procedure output. This section describes these options and their effect on each other.

Positioning the Legend on the Graphics Output Area

There are two ways you can position the legend on the graphics output area:

- Describe the general location of the legend with the POSITION= option. If necessary, fine-tune the position with the OFFSET= option.
- Position the legend explicitly with the ORIGIN=option.

Using POSITION= and OFFSET=

The values of the POSITION= option affect the legend in two ways:

- OUTSIDE and INSIDE determine whether the legend is located outside or inside the axis area.
- BOTTOM or MIDDLE or TOP (vertical position) and LEFT or CENTER or RIGHT (horizontal position) determine where the legend is located in relation to its OUTSIDE or INSIDE position.

Figure 7.10 on page 162 shows the legend positions inside the axis area.

Figure 7.10 Legend Positions Inside the Axis Area

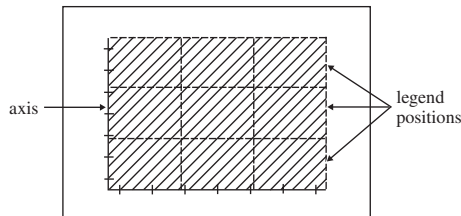
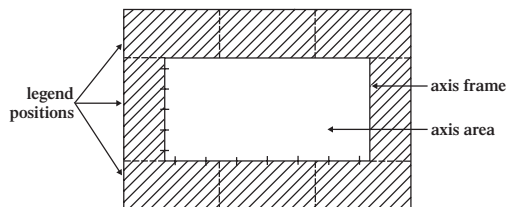


Figure 7.11 on page 162 shows legend positions outside the axis area.

Figure 7.11 Legend Positions Outside the Axis Area



The default combination is POSITION=(BOTTOM CENTER OUTSIDE). The combination (OUTSIDE MIDDLE CENTER) is not valid.

Use `OFFSET=(x,y)` to adjust the position of the legend specified by `POSITION=`. The x value shifts the legend either left or right and the y value shifts the legend either up or down.

The offset values are always applied *after* the `POSITION=` request. For example, if `POSITION=(TOP RIGHT OUTSIDE)`, the legend is located in the upper right corner of the graphics output area. If `OFFSET=(0,0)` is specified, the legend does not move. If `OFFSET=(-5,-8)CM`, the legend moves 5 centimeters to the left and 8 centimeters down.

Using `ORIGIN=`

Use `ORIGIN=(x,y)` to specify the coordinates of the exact location of the lower left corner of the legend box. Because `ORIGIN=(0,0)` is the lower left corner of the graphics output area, the values of x and y must be positive. If you specify negative values, a warning is issued and the default value is used.

Relating Legends to Other Graphic Elements

By default, the legend is inside the procedure output area and the space it occupies reduces the size of the graph itself. To control the way the legend relates to the other elements of the graph, use the `MODE=` option. These are values for `MODE=`:

- `RESERVE` reserve space for the legend outside the axis area and move the graph to make room for the legend. This is the default setting and is valid only when `POSITION=OUTSIDE`.
- `PROTECT` prevents the legend from being overwritten by the procedure output. `PROTECT` blanks out graphics elements, allowing only legend elements to be displayed in the legend's space.
- `SHARE` displays both graphics elements and legend elements in the same space. This setting is usually used when the legend is positioned inside the axis area. `SHARE` is useful when the graph has a space that the legend can fit into. For an example, see “Example 8. Creating a Simple Web Page with the ODS HTML Statement” on page 245.

Interactions Between `POSITION=` and `MODE=`

You cannot specify both `POSITION=INSIDE` and `MODE=RESERVE` because `MODE=RESERVE` assumes the legend is *outside* the axis area, and `POSITION=INSIDE` positions the legend *inside* the axis area. Therefore, when you specify `POSITION=INSIDE`, change the value of `MODE=` to `SHARE` or `PROTECT`. Otherwise, SAS/GRAPH issues a warning and automatically changes the value of `MODE=` to `PROTECT`.

Creating Drop Shadows and Block Effects

To produce a drop shadow or a three-dimensional block effect behind the legend use the `CSHADOW=` or `CBLOCK=` option in the `LEGEND` statement in conjunction with the graphics option `OFFSHADOW=(x,y)`.

The value of x determines how far the shadow or block extends to the right (positive numbers) or to the left (negative numbers) of the legend. The value of y determines how far the shadow or block extends above (positive numbers) or below (negative numbers) the legend. If `OFFSHADOW=(0,0)` is specified, the shadow or block is not visible.

By default, `OFFSHADOW=(0.0625, -0.0625) IN`; that is, the shadow or block extends 1/16th of an inch to the right and 1/16th of an inch below the legend.

NOTE Statement

See “TITLE, FOOTNOTE, and NOTE Statements” on page 210.

ODS HTML Statement

The ODS HTML statement opens or closes the HTML destination.

Used by:

GANNO, GAREABAR, GBARLINE, GCHART, GCONTOUR, GFONT, GIMPORT, GMAP, GPLOT, GPRINT, GRADAR, GREPLAY, GSLIDE, GTESTIT, G3D, G3GRID procedures

Requirements:

If the HTML destination is open, the BODY= argument is required.

Operating Environment Information: On mainframes, either GPATH= or PATH= is also required. Δ

Description

This section describes the ODS HTML statement as it relates to SAS/GRAPH procedures.

The ODS HTML statement opens or closes the HTML destination. If the destination is open, it produces output that is written in Hyper Text Markup Language (HTML). If DEVICE=GIF, graphics output is produced as GIF files, and the HTML files display the GIF files that are created by the SAS/GRAPH procedures. If DEVICE=JAVAMETA, graphics output is produced as *metagraphics* data, which provides graphs that are exact replicas of their corresponding GRSEG graphs but that are interactive in a Web page. Procedures GCHART, GCONTOUR, GMAP, GPLOT, and G3D can also be used with the JAVA and ACTIVEEX drivers, both of which generate JavaScript in the output HTML file. If DEVICE=JAVA, then a Java-enabled browser can use the script to render graphs as a SAS/GRAPH Applet for Java. If DEVICE=ACTIVEEX, then an ActiveX-enabled browser can use the script to render graphs as a SAS/GRAPH Control for ActiveX.

Syntax

```
ODS HTML HTML-file-specification(s) | action
  <ANCHOR=string>
  <ARCHIVE=string>
  <ATTRIBUTES=(attribute-name1'=attribute-value-1' ...
  attribute-name-n'=attribute-value-n')>
  <CODEBASE=file-location <(URL=Uniform-Resource-Locator)>>
  <BASE=base-text>
  <GFOOTNOTE | NOGFOOTNOTE>
  <GPATH=graphics-location <(URL=Uniform-Resource-Locator | NONE )>>
  <GTITLE | NOGTITLE>
  <HEADTEXT=HTML-for-document-head>
  <METATEXT=HTML-for-document-head>
  <NEWFILE=starting-point>
  <PARAMETERS=(parameter-name1'=parameter-value-1' ...
  parameter-name-n'=parameter-value-n')>
  <PATH=file-location <(URL=Uniform-Resource-Locator | NONE )>>
```

```
<RECORD_SEPARATOR='string' | NONE>
<STYLE=style-definition>
<TRANTAB='translation-table'>;
```

- *action* can be one of

```
CLOSE
EXCLUDE
SELECT
SHOW
```

Note: For information on EXCLUDE, SELECT, and SHOW, see *SAS Output Delivery System: User's Guide*. △

- *HTML-file-specification(s)* can be one or more of

```
BODY=file-specification
CONTENTS=file-specification
FRAME=file-specification
PAGE=file-specification
```

Note: BODY= is required. If you use FRAME=, you must also use CONTENTS= or PAGE=. △

Required Arguments

One of these arguments is required.

CLOSE

closes the HTML destination and closes any HTML files that are currently open.

Featured in: “Example 8. Creating a Simple Web Page with the ODS HTML Statement” on page 245.

EXCLUDE

excludes output objects from the HTML destination.

SELECT

selects output objects to send to the HTML destination.

SHOW

writes to the SAS log the current selection or exclusion list for the HTML destination.

HTML-file-specification

opens the HTML destination and specifies the HTML file or files to write to. You can open up to four HTML files; the file designated by BODY= is required.

Whenever you open one of these files, it remains open until you either

- close the HTML destination with ODS HTML CLOSE
- open a different file for the same HTML file specification.

HTML-file-specification can be one or more of the following arguments. Values for *file-specification* follow the arguments.

```
BODY=file-specification
FILE=file-specification
```

identifies the file that contains the HTML version of the procedure output. With SAS/GRAPH, the body file contains references to the graphs. If

DEVICE=GIF, the graphs are stored in separate GIF files. When you view the body file on a browser, the graphs are automatically displayed.

Featured in: “Example 8. Creating a Simple Web Page with the ODS HTML Statement” on page 245 and “Example 10. Creating a Bar Chart with Drill-down for the Web” on page 255.

CONTENTS=*file-specification*

identifies the file that contains a table of contents to the ODS output that is produced while the HTML destination is open. The contents file contains links to the body file(s).

The text of links to graphics output is taken from the description field of the GRSEG catalog entry. Use the DESCRIPTION= option in the procedure to change the link text.

You can display a contents file alone or in conjunction with a frame file. If you display a contents file directly (without using a frame file), selecting a link opens the associated body file, and the contents file is no longer displayed. If you display a contents file with a frame file, the contents file always remains available in the left frame, and selecting a link opens the associated body file in the right frame.

FRAME=*file-specification*

identifies a file that points to the body file and to either the table of contents file or the page file, or both. If you specify FRAME=, you must also specify either CONTENTS= or PAGE= or both.

When you open the frame file in the browser, it displays the Table of Contents or the Table of Pages or both in the left frame, and the body file in the right frame.

PAGE=*file-specification*

identifies the file that contains a table of pages to the ODS output that is produced while the HTML destination is open. The pages file contains links to the body file(s). ODS produces a new page of output whenever a procedure explicitly specifies for a new page. The SAS system option PAGESIZE= has no effect on pages in HTML output.

File-specification identifies the file or SAS catalog to write to and can be one of the following:

- fileref* (<URL='Uniform-Resource-Locator'> <NO_BOTTOM_MATTER>
<NO_TOP_MATTER> <DYNAMIC>)
- external-file* (<URL='Uniform-Resource-Locator'> <NO_BOTTOM_MATTER>
<NO_TOP_MATTER> <DYNAMIC>)
- entry.HTML* (<URL='Uniform-Resource-Locator'> <NO_BOTTOM_MATTER>
<NO_TOP_MATTER> <DYNAMIC>)

where

external-file

is the physical name of an external file to write to. For information on specifying external files, see the SAS companion for your operating environment.

fileref

is a fileref that has been assigned to an external file. The fileref must refer to a single file; it cannot point to an aggregate file storage location. Use a FILENAME statement to assign a fileref. See also “FILENAME Statement” on page 28.

entry.HTML

specifies an entry in a SAS catalog to write to. You must also specify a library and catalog. See the discussion on PATH=“ODS HTML Statement” on page 164.

URL=’Uniform-Resource-Locator’

provides a URL for *file-specification*. ODS uses this URL instead of the file name in all the links and references that it creates that point to the file.

This option is useful for building HTML files that may be moved from one location to another. If the links from the contents and page files are constructed with a simple URL (one name), they work as long as the contents, page, and body files are all in the same location.

NO_BOTTOM_MATTER**NOBOT**

omits the bottom matter for the file. By default, when you close a file that was open for HTML output of any kind, ODS writes some HTML to the bottom of the file. This HTML ends the file so that it can be viewed cleanly in a browser.

If you wish to leave a file in a state that you can append to, use NO_BOTTOM_MATTER on the BODY= option on the ODS HTML statement that opens the file. This option, in conjunction with NO_TOP_MATTER makes it possible for you to add output to a file that already exists and to put your own HTML code in the file between pieces of output.

To use NO_BOTTOM_MATTER, you must use a fileref for *file-specification*. The FILENAME statement that defines the fileref must include the host-specific option that opens the file for appending.

When you are opening a file that ODS has previously written to, you must use ANCHOR= to specify a new base name for the anchors to avoid duplicating anchors that already exist in the file. See the discussion “About Anchors” on page 168.

NO_TOP_MATTER**NOTOP**

omits the opening matter for the file. By default, when you open a file for HTML output of any kind, ODS writes some HTML to the top of the file.

If you wish to append ODS output to an existing file, you must open the file with NO_TOP_MATTER on the BODY= option on the ODS HTML statement that opens the file. This option, in conjunction with NO_BOTTOM_MATTER makes it possible for you to add output to a file that already exists and to put your own HTML code in the file between pieces of output.

To use NO_TOP_MATTER, you must use a fileref for *file-specification*. The FILENAME statement that defines the fileref must include the host-specific option that opens the file for appending.

When you are opening a file that ODS has previously written to, you must use ANCHOR= to specify a new base name for the anchors to avoid duplicating anchors that already exist in the file. See the discussion “About Anchors” on page 168.

DYNAMIC

enables you to send HTML output directly to a web server instead of writing it to a file. This option sets the value of the HTMLContentType= attribute.

By default, if you do not specify DYNAMIC, ODS sets the value of HTMLContentType= for writing to a file.

Note: If you specify the DYNAMIC suboption with any file specification in the ODS HTML statement, you must specify it for all the file specifications in the statement. △

Note: Regardless of how you specify the file, you may need to include the extension .HTML or .HTM on the file name. Some browsers require one of these extensions in order to read the file. \triangle

Note: For additional information, refer to the *Output Delivery System*. \triangle

Using the ODS HTML Statement

While the ODS HTML destination is open, you can submit as many ODS HTML statements as you like, and you can place them anywhere in your SAS/GRAPH program. This enables you to open new files, change anchor names, or specify a new location for graphics output whenever you like. At the end of your ODS HTML processing step, submit ODS HTML CLOSE to close the destination and all open files.

Specifying a Destination for ODS HTML Output

When you use the ODS HTML statement for SAS/GRAPH, you must do the following:

- assign a body file with the ODS HTML BODY= option
- specify DEVICE=GIF (or TARGET=GIF) in a GOPTIONS statement to create the GIF files. By default with ODS processing, the GIF files are stored in the current directory. To specify a destination for all the HTML and GIF files, use the PATH= option. To store the GIF files in a different location than the HTML files, use the GPATH= option to specify a location for the GIF files, and PATH= to specify the location of the HTML files. In both cases, the destination must be an aggregate storage location. With procedures GCHART, GCONTOUR, GMAP, GPLOT, and G3D, you can also use DEVICE=ACTIVE X to create graphs as ActiveX controls, or DEVICE=JAVA to create graphs as Java applets. The controls or applets are defined within the body file.

Note: For more information, see *SAS Output Delivery System: User's Guide*. \triangle

For more information about the output files generated for use with the Web, see “Types of Web Presentations Available” on page 370.

About Anchors

ODS HTML automatically creates an *anchor* for every piece of output generated by the SAS procedures. An anchor specifies a particular location within an HTML file. In SAS/GRAPH, an anchor usually defines a link target such as a graph whose location is defined in an IMG element.

Note: For additional information, refer to the *Output Delivery System*. \triangle

In order for the links from the contents, page, or frame file to work, each piece of output in the body files must have a unique anchor to link to. The anchor for the first piece of output in a body file acts as the anchor for that file. These anchors are used by the frame and contents files, if they are created, to identify the targets for the links that ODS HTML automatically generates.

By default, the first anchor is named **IDX** and all subsequent anchors generated while the HTML destinations remain open increment that name. Anchor values increment while the ODS HTML destinations remain open unless you use ANCHOR= to assign a new value. Anchor values continue to increment when you open new body files, start new procedures, or produce different types of output.

Controlling the anchor name is useful when you create a graph with drill-down capability. In this case, you must create a variable that contains the names of the anchors that are the targets for the different areas of the graph that the user may click on.

PATTERN Statement

The PATTERN statement defines the characteristics of patterns used in graphs.

Used by:

GCHART, GCONTOUR, GMAP, GPLOT procedures; SYMBOL statement;
Annotate facility

Global

Assigned by default

Description

PATTERN statements create PATTERN definitions that define the color and type of area fill for patterns used in graphs. These are the procedures and the graphics areas that they create that use PATTERN definitions:

GCHART	color, fill pattern, or image for the bars in 2D bar charts; color and fill pattern for the segments of 3D bar charts, pie charts, and star charts.
GCONTOUR	contour levels in contour plots
GMAP	map areas in choropleth, block, and prism maps; blocks in block maps
GPLOT	areas beneath or between plotted lines.

In addition, the SYMBOL statement and certain Annotate facility functions and macros can use pattern specifications. For details see the “SYMBOL Statement” on page 183 and Chapter 24, “Using Annotate Data Sets,” on page 587.

You can use the PATTERN statement to control the fill and color of a pattern, and whether the pattern is repeated. There are three types of patterns:

- bar and block patterns
- map and plot patterns
- pie and star patterns

Pattern fills can be solid or empty, or composed of parallel or crosshatched lines. For two-dimensional bar charts, the PATTERN statement can specify images to fill horizontal or vertical bars. In addition, you can specify device-dependent hardware patterns for rectangle, polygon, and pie fills on devices that support hardware patterns.

If you do not create PATTERN definitions, SAS/GRAPH software generates them as needed and assigns them to your graphs by default. Generally, the default behavior is to rotate a solid pattern through the current colors list. For details, see “About Default Patterns” on page 177.

Syntax

```
PATTERN<1...255>
    <COLOR=pattern-color>
    <REPEAT=number-of-times>
```

```
<VALUE=bar/block-pattern
| map/plot-pattern
| pie/star-pattern
| hardware-pattern>;
```

- *bar/block-pattern* can be one of these:
 - EMPTY
 - SOLID
 - style <density>*
- *map/plot-pattern* can be one of these:
 - MEMPTY
 - MSOLID
 - Mdensity <style <angle>>*
- *pie/star-pattern* can be one of these:
 - PEMPTY
 - PSOLID
 - Pdensity <style <angle>>*
- *hardware-pattern* has this form:
 - HWxxxnnn

Options

COLOR=*pattern-color*

C=*pattern-color*

specifies the color of the fill. *Pattern-color* is any SAS/GRAPH color name. See Chapter 6, “SAS/GRAPH Colors and Images,” on page 91 .

Note: ActiveX assigns colors in a different order from Java, so the same data can appear differently with those two drivers. Δ

Using COLOR= with a null value cancels the color specified in a previous PATTERN statement of the same number without affecting the values of other options.

COLOR= overrides the CPATTERN= graphics option.

The CFILL= option in the PIE and STAR statements overrides COLOR=. For details, see “Controlling Slice Patterns and Colors” on page 831.

CAUTION:

Omitting COLOR= in a PATTERN statement may cause the PATTERN statement to generate multiple PATTERN definitions. Δ

If no color is specified for a PATTERN statement, that is, if neither COLOR= nor CPATTERN= is used, the PATTERN statement rotates the specified fill through each color in the colors list before the next PATTERN statement is used. For details, see “Understanding Pattern Sequences” on page 182.

See also: “Working with PATTERN Statements” on page 179.

Featured in: “Example 7. Using BY-group Processing to Generate a Series of Charts” on page 240.

Not supported by: Java (partial), ActiveX (partial)

IMAGE= *fileref* | '*external-file*'

specifies an image file that will be used to fill one or more bars of a two-dimensional bar chart, as generated by the HBAR and VBAR statements of the GCHART procedure. The format of the external file specification varies across operating environments. See also the IMAGESTYLE= option.

Note: When you specify an image file to fill a bar, the bar is not outlined. Δ

Note: If an image is specified on a PATTERN statement that is used with another type of chart, then the PATTERN statement is ignored and default pattern rotation is affected. For example, if you submit a PIE statement when an image has been specified in a PATTERN statement, the default fill pattern is used for the pie slices, with each slice in the pie displaying the fill pattern in the same color.

In the Java applet, IMAGE= only works on 2-dimensional rectangular bars. The Java applet does not support images on arbitrary polygons. Δ

See also: For related information, see “Placing Images on the Bars of Two-Dimensional Bar Charts” on page 116.

Not supported by: Java (partial), ActiveX (partial)

IMAGESTYLE = TILE | FIT

specifies how the image specified in the IMAGE= option is to be applied to fill a bar in a two-dimensional bar chart. The TILE value, which is the default, copies the image as needed to fill the bar. The FIT value stretches a single instance of the image to fill the bar.

See also: For related information, see “Placing Images on the Bars of Two-Dimensional Bar Charts” on page 116.

Not supported by: Java (partial), ActiveX (partial)

REPEAT=*number-of-times*

R=*number-of-times*

specifies the number of times that a PATTERN definition is applied before the next PATTERN definition is used. By default, REPEAT=1.

The behavior of REPEAT= depends on the color specification:

- If you use both COLOR= and REPEAT= in a PATTERN statement, the pattern is repeated the specified number of times in the specified color. The fill can be either the default solid or a fill specified with VALUE=.
- If you use CPATTERN= in a GOPTIONS statement to specify a single pattern color, and use REPEAT= either alone or with VALUE= in a PATTERN statement, the resulting hatch pattern is repeated the specified number of times.
- If you omit both COLOR= and CPATTERN=, and use REPEAT= either alone (generates default solids) or with VALUE= in a PATTERN statement, the resulting pattern is rotated through each color in the colors list, and then the entire group generated by this cycle is repeated the number of times specified in REPEAT=. Thus, the total number of patterns produced depends on the number of colors in the current colors list.

Using REPEAT= with a null value cancels the repetition specified in a previous PATTERN statement of the same number without affecting the values of other options.

See also: “Understanding Pattern Sequences” on page 182.

Not supported by: Java (partial), ActiveX (partial)

VALUE=*bar / block-pattern*

V=*bar / block-pattern*

specifies patterns for:

- bar charts produced by the HBAR, HBAR3D, VBAR, and VBAR3D statements in the GCHART procedure including 2D and 3D bar shapes.
- the front surface of blocks in block charts produced by the BLOCK statement in the GCHART procedure.
- the blocks in block maps produced by the BLOCK statement in the GMAP procedure. (The map area from which the block rises takes a map pattern as described on the option VALUE= on page 173). See also “About Block Maps and Patterns” on page 1016.

Values for *bar/block-pattern* are

EMPTY E	an empty pattern. Neither the Java applet nor the ActiveX control supports EMPTY.
SOLID S	a solid pattern (the only valid value for 3D charts).
<i>style</i> < <i>density</i> >	a shaded pattern.

Note: *style*<*density*> is not supported by the Java or ActiveX clients. Δ

Style specifies the direction of the lines:

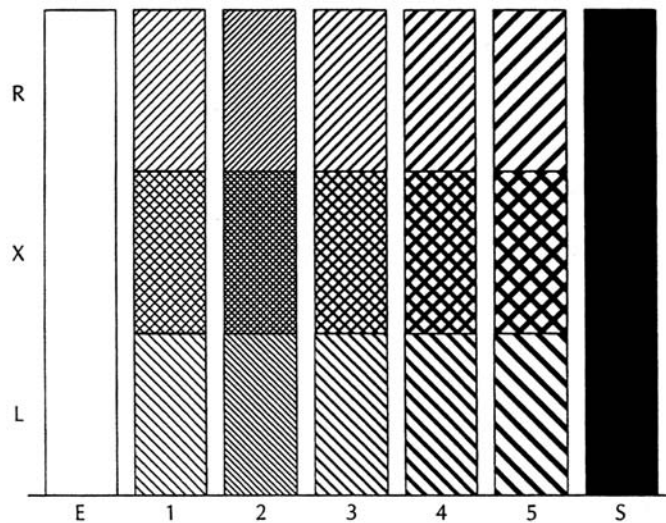
L	left-slanting lines.
R	right-slanting lines.
X	crosshatched lines.

Density specifies the density of the pattern's shading:

1...5	1 produces the lightest shading and 5 produces the heaviest shading.
-------	--

Figure 7.12 on page 172 shows all of the patterns available for bars and blocks.

Figure 7.12 Bar and Block Patterns



If no valid patterns are available, default bar and block fill patterns are selected in this order:

- 1 SOLID

2 X1– X5

3 L1– L5

4 R1– R5

Each fill is used once with every color in the colors list unless a pattern color is specified. The entire sequence is repeated as many times as required to provide the necessary number of patterns.

Note: If the V6COMP graphics option is in effect, or if color is limited to a single color with the CPATTERN= or COLORS= graphics options, the order is X1–X5, L1– L5, R1– R5, S, and E. △

Not supported by: Java (partial), ActiveX (partial)

VALUE=*map/plot-pattern*

V=*map/plot-pattern*

specifies patterns for:

- contour levels in contour plots produced by the GCONTOUR procedure
- map area surfaces in block, choropleth, and prism maps produced by the BLOCK, CHORO, AND PRISM statements in the GMAP procedure.
- areas under curves in plots produced by the AREAS= option in the PLOT statement in the GPLOT procedure.

Values for *map/plot-pattern* are

MEMPTY an empty pattern. EMPTY or E are also valid aliases, except
ME when used with the map areas in block maps created by the
 GMAP procedure.

MSOLID a solid pattern. SOLID or S are also valid aliases, except when
MS used with the map areas in block maps created by the GMAP
 procedure.

Mdensity<*style*<*angle*>> a shaded pattern.

Note: Mdensity<*style*<*angle*>> is not supported by the Java or ActiveX clients. △

Density specifies the density of the pattern's shading:

1...5 1 produces the lightest shading and 5
 produces the heaviest shading.

Style specifies the type of the pattern lines:

N parallel lines (the default).

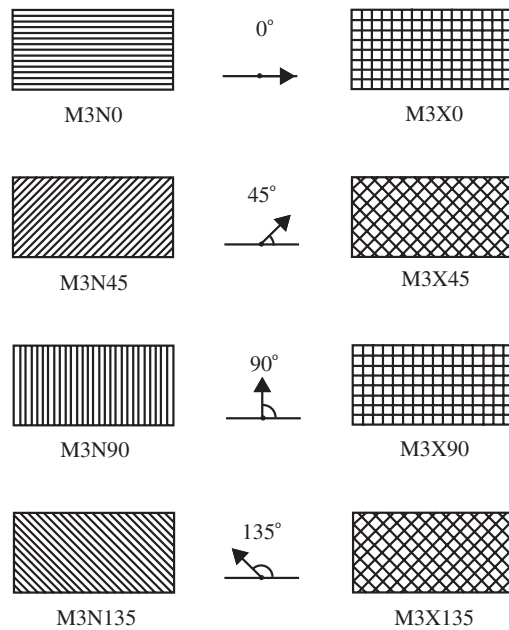
X crosshatched lines.

Angle specifies the angle of the pattern lines:

0...360 the degrees at which the parallel lines are
 drawn, measured from the horizontal. By
 default, *angle* is 0 (lines are horizontal).

Figure 7.13 on page 174 shows some typical map and plot patterns.

Figure 7.13 Map and Plot Patterns



If no valid patterns are available, default map and plot fill patterns are selected in this order:

- 1 MSOLID
- 2 M2N0
- 3 M2N90
- 4 M2X45
- 5 M4N0
- 6 M4N90
- 7 M4X90

Each fill is used once with every color in the colors list unless a pattern color is specified. The entire sequence is repeated as many times as required to provide the necessary number of patterns.

Note: If the V6COMP graphics option is in effect, or if color is limited to a single color with the CPATTERN= or COLORS= graphics options, MSOLID is not used and the default fill list starts with M2N0. Δ

Not supported by: Java (partial), ActiveX (partial)

VALUE=*pie / star-pattern*

V=*pie / star-pattern*

specifies patterns for pie and star charts produced by the PIE and STAR statements in the GCHART procedure. Values for *pie / star-pattern* are

PEMPTY an empty pattern. EMPTY or E are also valid aliases.
PE

PSOLID a solid pattern. SOLID or S are also valid aliases.
PS

Pdensity<*style*<*angle*>> a shaded pattern.

Note: $Pdensity<style<angle>>$ is not supported by the Java or ActiveX clients. Δ

Density specifies the density of the pattern's shading:

1...5 1 produces the lightest shading and 5 produces the heaviest shading.

Style specifies the type of the pattern lines:

N parallel lines (the default).
X crosshatched lines.

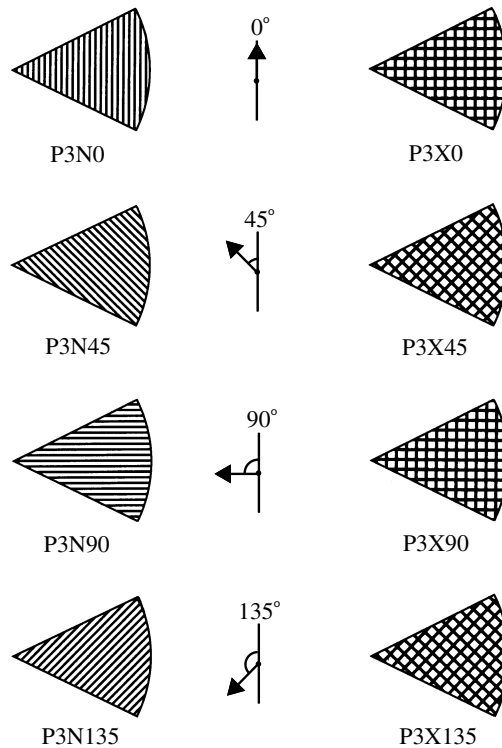
Angle specifies the angle of the pattern lines:

0...360 the angle of the lines, measured in degrees from perpendicular to the radius of the slice. By default, *angle* is 0.

The FILL= option in the PIE and STAR statements in the GCHART procedure overrides VALUE=.

Figure 7.14 on page 175 shows some typical pie and star patterns.

Figure 7.14 Pie and Star Patterns



If no valid patterns are available, default pie and star fill patterns are selected in this order:

- 1 PSOLID
- 2 P2N0
- 3 P2N90

- 4 P2X45
- 5 P4N0
- 6 P4N90
- 7 P4X90

Each fill is used once with every color in the colors list unless a pattern color is specified. The entire sequence is repeated as many times as required to provide the necessary number of patterns.

Note: If the V6COMP graphics option is in effect, or if color is limited to a single color with the CPATTERN= or COLORS= graphic options, PSOLID is not used and the default fill list starts with P2N0. Δ

Note: If you use hatch patterns and request a legend instead of slice labels, the patterns in the slices are oriented to be visually equivalent to the legend. Δ

Not supported by: Java (partial), ActiveX (partial)

VALUE=HWxxxnnn

specifies a hardware pattern where

HW	identifies the pattern as a hardware pattern. The pattern name must begin with the characters HW.
xxx	the last two or three characters of the module name in the <i>Module</i> field in the Detail window of your device entry. If the module name has eight characters (SASGDPSL, for example), use the last three characters (PSL). If the module name has only seven characters (SASGDVT, for example), use the last two characters (VT).
nnn	the number the driver uses to identify the device-dependent pattern. Patterns and associated pattern numbers vary from device to device. See the documentation for your device for valid pattern numbers. For a brief description of some device specific pattern values, see “Specifying Device-Dependent Hardware Patterns” on page 181.

If you specify a hardware pattern for a device that does not support hardware patterns, or if you specify an invalid pattern number, a solid rectangle, polygon, or pie fill is substituted. A solid fill will also be used in place of a hardware pattern in certain types of clipped polygons. See the PCLIP and POLYGONCLIP options in Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for more information on using hardware patterns with clipped polygons.

See also: “Specifying Device-Dependent Hardware Patterns” on page 181.

Not supported by: Java (partial), ActiveX (partial)

Using the PATTERN Statement

PATTERN statements can be located anywhere in your SAS program. They are global and remain in effect until redefined, canceled, or until the end of your SAS session.

You can define up to 255 different PATTERN statements. A PATTERN statement without a number is treated as a PATTERN1 statement.

PATTERN statements generate one or more PATTERN definitions, depending on how the COLOR=, VALUE=, and IMAGE= options are used. For information on PATTERN definitions, see “Working with PATTERN Statements” on page 179, as well as the

description of COLOR= on page 170, VALUE= on page 173, and IMAGE= on page 171 options.

PATTERN definitions are generated in the order in which the statements are numbered, regardless of gaps in the numbering or the statement's position in the program. Although it is common practice, you do not have to start with PATTERN1, and you do not have to use sequential statement numbers.

PATTERN definitions are applied automatically to all areas of the graphics output that require patterns. When assigning PATTERN definitions, SAS/GRAPH starts with the lowest-numbered definition with an appropriate fill specification or with no fill specification. It continues to use the specified patterns until all valid PATTERN definitions have been used. Then, if more patterns are required, SAS/GRAPH returns to the default pattern rotation, but continues to outline the areas in the same color as the fill.

Altering or Canceling PATTERN Statements

PATTERN statements are additive. If you define a PATTERN statement and later submit another PATTERN statement with the same number, the new PATTERN statement redefines or cancels only the options that are included in the new statement. Options not included in the new statement are not changed and remain in effect. For example, assume you define PATTERN4 as

```
pattern4 value=x3 color=red repeat=2;
```

This statement cancels only REPEAT= without affecting the rest of the definition:

```
pattern4 repeat=;
```

Add or change options in the same way. This statement changes the color of the pattern from red to blue:

```
pattern4 color=blue;
```

After all these modifications, PATTERN4 has these characteristics:

```
pattern4 value=x3 color=blue;
```

Cancel individual PATTERN statements by defining a PATTERN statement of the same number without options (a null statement):

```
pattern4;
```

Canceling one PATTERN statement does not affect any other PATTERN definitions. To cancel all current PATTERN statements, use the RESET= option in a GOPTIONS statement:

```
goptions reset=pattern;
```

Specifying RESET=GLOBAL or RESET=ALL cancels all current PATTERN definitions as well as other settings.

To display a list of current PATTERN definitions in the LOG window, use the GOPTIONS procedure with the PATTERN option:

```
proc goptions pattern nolist;
run;
```

About Default Patterns

When a procedure produces a graph that needs one or more patterns, SAS/GRAPH either

- automatically generates the appropriate default patterns and outlines to fill the areas, or
- uses patterns, colors, and outlines that are defined by PATTERN statements, graphics options, and procedure options.

In order to understand how SAS/GRAPH generates and assigns patterns defined with PATTERN statements it is helpful to understand how it generates and assigns default patterns. The following sections describe the default pattern behavior for all procedures. See “Working with PATTERN Statements” on page 179 for details about defining patterns.

How Default Patterns and Outlines Are Generated

In general, SAS/GRAPH uses default patterns when no PATTERN statements are defined. Typically, the default pattern that SAS/GRAPH uses is a solid fill that it rotates once through the colors list, skipping the foreground color. By default, SAS/GRAPH also outlines all areas in the foreground color. (Typically, the foreground color is the first color in the device’s colors list.)

Specifically, SAS/GRAPH uses default patterns and outlines when you

- do not specify *any* PATTERN statements, and
- do not use the CPATTERN= graphics option, and
- do not use the COLORS= graphics options (that is, you use the device’s default colors list and it has more than one color), and
- do not use the COUTLINE= option in the action statement.

If all of these conditions are true, then SAS/GRAPH

- selects the first default fill for the appropriate pattern, which is always solid, and rotates it once through the colors list, generating one solid pattern for each color. If the first color in the device’s colors list is black (or white), SAS/GRAPH skips that color and begins generating patterns with the next color.

Note: The one exception to the default solid pattern is the map area pattern in a block map produced by the GMAP procedure, which uses a hatch fill by default. By default the map areas and their outlines use the first color in the colors list, regardless of whether the list is the default device list or one specified with COLORS= in the GOPTIONS statement. \triangle

- uses the foreground color to outline every patterned area.

For example, the default colors list for the PSCOLOR device contains BLACK, RED, GREEN, BLUE, CYAN, MAGENTA, YELLOW, and GRAY. Therefore, for this device, the first five default patterns are solid red, solid green, solid blue, solid cyan, and solid magenta. These patterns are all outlined in black, the first color in the colors list.

If a procedure needs additional patterns, SAS/GRAPH selects the next default pattern fill appropriate to the graph and rotates it through the colors list, skipping the foreground color as before. SAS/GRAPH continues in this fashion until it has generated enough patterns for the chart.

Things That Affect Default Patterns

Changing any of these conditions may change or override the default behavior:

- If you specify a colors list with the COLORS= option in a GOPTIONS statement and the list contains more than one color, SAS/GRAPH rotates the default fills, beginning with SOLID, through that list. In this case, it uses every color, even if the foreground color is black (or white). The default outline color remains the foreground color.

- If you specify either `COLORS=(one-color)` or the `CPATTERN= graphics` option, the default fill changes from `SOLID` to the appropriate list of hatch patterns. SAS/GRAPH uses the specified color to generate one pattern definition for each hatch pattern in the list. The default outline color remains the foreground color.
- Whenever you specify `PATTERN` statements, whether or not the procedure can use them, the default outline color for all patterns changes from foreground to `SAME`. Therefore, when a procedure runs out of specified patterns and returns to the default pattern rotation, the outline color is `SAME`, not foreground.

For a description of these graphics options, see Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261.

Working with PATTERN Statements

With `PATTERN` statements, you can specify

- the type of fill (`VALUE=`)
- the color of the fill (`COLOR=`)
- the images used to fill the bars in a 2D chart (`IMAGE=`)
- how many times to apply the statement before using the next one (`REPEAT=`).

See “Placing Images on the Bars of Two-Dimensional Bar Charts” on page 116 for information on filling the bars of two-dimensional bar charts with images using the `PATTERN` statement.

You can also use procedure options to specify the pattern outline color and the `CPATTERN= graphics` option to specify a default color for all patterns.

Whether you use `PATTERN` statement options alone or with each other affects the number and kind of patterns your `PATTERN` statements generate. Depending on the options you use, you can explicitly specify every pattern used by your graphs or you can let the `PATTERN` statement generate a series of pattern definitions using either the colors list or the list of default fills.

Explicitly Specifying Patterns

To explicitly specify all the patterns in your graph, you need to do one of the following for every pattern your graph requires:

- Provide a `PATTERN` statement that uses the `COLOR=` option to specify the pattern color, for example

```
pattern1 color=red;
```

By default, the fill type `SOLID`.

- Provide a `PATTERN` statement that uses both the `COLOR=` option and the `VALUE=` option to specify the fill, for example

```
pattern1 color=blue value=r3;
```

Including `COLOR=` in the `PATTERN` statement is the simplest way to assure that you get exactly the patterns you want. When you use the `COLOR=` option, the `PATTERN` statement generates exactly one `PATTERN` definition for that statement. If you also use the `REPEAT=` option, the `PATTERN` definition is repeated the specified number of times.

Generating Multiple Pattern Definitions

You can also use `PATTERN` statements to generate multiple `PATTERN` definitions. To do this use the `VALUE=` option to specify the type of fill you want but omit the `COLOR=` option – for example

```
pattern1 value=r3;
```

In this case, the PATTERN statement rotates the R3 fill through all the colors in the colors list. For more information on pattern rotation, see “Understanding Pattern Sequences” on page 182.

Selecting an Appropriate Pattern

The type of fill you specify depends on the type of graph you are producing:

With...	Use...
bar and block charts (PROC GCHART), block maps (PROC GMAP)	VALUE= bar/block-pattern on page 171
contour plots (PROC GCONTOUR), map area surfaces (PROC GMAP)	VALUE=map/plot-pattern on page 173
pie and star charts (PROC GCHART)	VALUE=pie/star-pattern on page 174

Note: If you specify a fill that is inappropriate for the type of graph you are generating (for example, if you specify VALUE=L1 in a PATTERN statement for a choropleth map), SAS/GRAPH ignores the PATTERN statement and continues searching for a valid pattern. If it does not find a definition with a valid fill specification, it uses default patterns instead. \triangle

Controlling Outline Colors

Whenever you use PATTERN statements, the default outline color is the same as the fill color, for example, a blue bar has a blue outline. The effect is the same as specifying COUTLINE=SAME. Even when the procedure runs out of user-defined patterns and generates default patterns, the outlines continue to match the interior fill color.

To change the outline color of any pattern, whether default or user-defined, use the COUTLINE= option in the action statement that generates the chart.

The Effect of the CPATTERN= Graphics Option

Although the CPATTERN= graphics option is used most often with default patterns, it does affect the PATTERN statement. With default patterns (no PATTERN statements specified) it

- specifies the color for all patterns
- causes default patterns to use hatched fills instead of the default SOLID.

In conjunction with the PATTERN statement it does the following:

- With a PATTERN statement that only specifies a fill (VALUE=), CPATTERN= determines the color of that fill. For example, these statements produce two green, hatched patterns:

```
goptions cpattern=green;
pattern1 value=x3;
pattern2 value=x1;
```

- With a PATTERN statement that only specifies a color (COLOR=), the COLOR= option overrides the CPATTERN= color, but CPATTERN= causes the fill to be

hatched, not the default SOLID. For example, these statements produce one red, hatched pattern:

```
goptions cpattern=green;
pattern1 color=red;
```

See also the description of CPATTERN="CPATTERN" on page 274.

Specifying Version 6 Patterns

If you specify the V6COMP graphics option, SAS/GRAPH generates patterns by rotating the appropriate Version 6 default patterns through all the colors in the colors list. With V6COMP, all patterns are outlined in the same color as the fill.

Specifying Device-Dependent Hardware Patterns

You can specify device-dependent hardware patterns with the types of device drivers described in this section.

GDDM Drivers

GDDM drivers include several sets of hardware patterns. These patterns include both predefined and user-defined (device-dependent) fill patterns. When you use a hardware pattern with a GDDM driver, specify the name of the device-dependent pattern set you want the driver to use. This name will be stored in the GPROLOG string in the device entry for the driver. Specify the name of the pattern set in either of these ways:

- Use the GPROLOG= graphics option to assign the pattern set name to the GPROLOG string.
- Enter the pattern set name in the Gprolog window of the device entry for the GDDM device driver.

If you do not specify a pattern set name, the device uses a predefined pattern.

Values for *nnn* for predefined patterns are 1 through 16. Values for *nnn* for device-dependent patterns are 65 through 128.

Information regarding both types of fill patterns can be found in *GDDM Application Programming Guide*. For additional information on specifying hardware patterns with GDDM drivers, see also the *GDDM Base Programming Reference*.

TEK42xx Series Terminal Drivers

TEK42xx series terminal drivers support the predefined fill patterns found in the Technical Reference Guide for each terminal. These drivers can also support user-defined fill patterns. Values for *nnn* for these drivers are numbers less than 175.

HPLJxxxx Drivers

HPLJxxxx drivers for the HP LaserJet support the predefined shading levels and predefined fill patterns for rectangle fill only. These patterns are documented in the appropriate HP LaserJet technical manual. Values for *nnn* for shading levels are 001 through 008. Values for fill patterns are 009 through 014.

Metagraphics Drivers

Metagraphics drivers can use the hardware patterns supported by the device for which they are written. When you specify hardware patterns for a metagraphics driver, values of *nnn* can range from 0 through 999.

Understanding Pattern Sequences

Pattern sequences are sets of PATTERN definitions that SAS/GRAPH automatically generates when a PATTERN statement specifies a fill but not a color. In this case, the specified fill is used once with every color in the colors list. If REPEAT= is also used, the resulting PATTERN definitions are repeated the specified number of times.

Generating Pattern Sequences

SAS/GRAPH generates pattern sequences when a PATTERN statement uses VALUE= to specify a fill and all of the following conditions are also true:

- the COLOR= option is not used in the PATTERN statement
- the CPATTERN= graphics option is not used
- the colors list, either default or user-specified, contains more than one color.

In this case, the PATTERN statement rotates the fill specified by VALUE= through every color in the colors list, generating one PATTERN definition for every color in the list. After every color has been used once, SAS/GRAPH goes to the next PATTERN statement. For example, suppose you specified the following colors list and PATTERN statements for bar/block patterns:

```
goptions colors=(blue red green) ctext=black;
pattern1 color=red value=x3;
pattern2 value=r3;
pattern3 color=blue value=l3;
```

Here, **PATTERN1** generates the first PATTERN definition. **PATTERN2** omits COLOR=, so the specified fill is rotated through all three colors in the colors list before the PATTERN3 statement is used. This table shows the color and fill of the PATTERN definitions that would be generated if nine patterns were required:

Definition Number	Source	Characteristics:	
		Color	Fill
1	PATTERN1	red	x3
2	PATTERN2	blue	r3
3	PATTERN2	red	r3
4	PATTERN2	green	r3
5	PATTERN3	blue	l3
6	first default	blue	solid
7	first default	red	solid
8	first default	green	solid
9	second default	blue	x1

Notice that after all the PATTERN statements are exhausted, the procedure begins using the default bar and block patterns, beginning with SOLID. Each fill from the default list is rotated through all three colors in the colors list before the next default fill is used.

Repeating Pattern Sequences

If you use REPEAT= but not COLOR=, the sequence generated by cycling the definition through the colors list is repeated the number of times specified by REPEAT=. For example, these statements illustrate the effect of REPEAT= on PATTERN statements both with and without explicit color specifications:

```
options colors=(red blue green);
pattern1 color=gold repeat=2;
pattern2 value=x1 repeat=2;
```

Here, **PATTERN1** is used twice and **PATTERN2** cycles through the list of three colors and then repeats this cycle a second time:

Sequence Number	Source	Characteristics: Color	Fill
1	PATTERN1	gold	solid (first default)
2	PATTERN1	gold	solid (first default)
3	PATTERN2	red	x1
4	PATTERN2	blue	x1
5	PATTERN2	green	x1
6	PATTERN2	red	x1
7	PATTERN2	blue	x1
8	PATTERN2	green	x1

SYMBOL Statement

The SYMBOL statement defines the characteristics of symbols that display the data plotted by a PLOT statement used by PROC GBARLINE, PROC GCONTOUR, and PROC GPLOT.

Used by:

GBARLINE, GCONTOUR, GPLOT procedures

Global

Assigned by default

Description

SYMBOL statements create SYMBOL definitions, which are used by the GPLOT and GCONTOUR procedures. For the GPLOT procedure, SYMBOL definitions control

- the appearance of plot symbols and plot lines, including bars, boxes, confidence limit lines, and area fills
- interpolation methods
- how plots handle data out of range.

For the GCONTOUR procedure, SYMBOL definitions control

- the appearance and text of contour labels
- the appearance of contour lines.

If you create SYMBOL definitions, they are automatically applied to a graph by the procedure. If you do not create SYMBOL definitions, these procedures generate default definitions and apply them as needed to your plots.

Syntax

```
SYMBOL<1...255>
  <COLOR=symbol-color>
  <MODE=EXCLUDE | INCLUDE>
  <REPEAT=number-of-times>
  <STEP=distance<units>>
  <appearance-option(s)>
  <interpolation-option>
  <SINGULAR=n>;
```

appearance-options can be one or more of these:

```
BWIDTH=box-width
CI=line-color
CO=color
CV=value-color
FONT=font
HEIGHT=symbol-height<units>
LINE=line-type
POINTLABEL<=(label-description(s) | NONE>
VALUE=special-symbol | text-string | NONE
WIDTH=thickness-factor
```

interpolation-option can be one of these:

- general methods
 - INTERPOL=JOIN
 - INTERPOL=map/plot-pattern
 - INTERPOL=NEEDLE
 - INTERPOL=NONE
 - INTERPOL=STEP<placement><J><S>
- high-low interpolation methods
 - INTERPOL=BOX<option(s)><00...25>
 - INTERPOL=HILO<C><option(s)>
 - INTERPOL=STD<1 | 2 | 3><variance><option(s)>
- regression interpolation methods
 - INTERPOL=R<type><0><CLM | CLI<50...99>>
- spline interpolation methods
 - INTERPOL=L<degree><P><S>
 - INTERPOL=SM<nn><P><S>
 - INTERPOL=SPLINE<P><S>

Options

When the syntax of an option includes *units*, use one of these:

```
CELLS          character cells
```

CM	centimeters
IN	inches
PCT	percentage of the graphics output area
PT	points.

If you omit *units*, a unit specification is searched for in this order:

- 1 the GUNIT= option in a GOPTIONS statement
- 2 the default unit, CELLS.

BWIDTH=*box-width*

specifies the width of the box generated by either the INTERPOL=BOX or INTERPOL=HILOB option. *Box-width* can be any number greater than 0. By default, the value of *box-width* is the same as the value of the WIDTH= option, whose default value is 1. Therefore, if you specify a value for WIDTH= and omit BWDITH=, the width of the box changes accordingly.

Featured in: “Example 4. Creating and Modifying Box Plots” on page 233.

CI=*line-color*

specifies a color for an interpolation line (GPLOT) or a contour line (GCONTOUR). If you omit CI= but specify CV=, CI= assumes the value of CV=. In this case, CI= and CV= specify the same color, which is the same as specifying COLOR= alone.

If you omit CI=, the color specification is searched for in this order:

- 1 the CV= option
- 2 the COLOR= option
- 3 the CSYMBOL= option in a GOPTIONS statement
- 4 each color in the colors list sequentially before the next SYMBOL definition is used.

See also: “Using Color” on page 206

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226

CO=*color*

specifies a color for

- outlines of filled areas generated by the INTERPOL=*map/plot-pattern* option
- confidence limit lines generated by the INTERPOL=R *series* option
- staffs, boxes, and bars generated by the high-low interpolation methods: INTERPOL=HILO, INTERPOL=BOX, and INTERPOL=STD.

If you omit the CO= option, the search order for a color specification depends on the interpolation method being used.

See also: “Using Color” on page 206

Featured in: “Example 5. Filling the Area between Plot Lines” on page 236 and “Example 4. Creating and Modifying Box Plots” on page 233.

COLOR=*symbol-color*

C=*symbol-color*

specifies a color for the entire definition, unless it is followed by a more explicit specification. For the GPLOT procedure, this includes plot symbols, the plot line, confidence limit lines, and outlines. For the GCONTOUR procedure, this includes contour lines and labels.

Using the COLOR= option is exactly the same as specifying the same color for both the CI= and CV= options.

If COLOR= precedes CI= or CV= in the same statement, CI= or CV= is used instead.

If you do not use COLOR= or CI=, CV=, and CO=, the color specification is searched for in this order:

- 1 the CSYMBOL= option in a GOPTIONS statement
- 2 each color in the colors list sequentially before the next SYMBOL definition is used.

If you do not use a SYMBOL statement to specify a color for each symbol, but you do specify a colors list in a GOPTIONS statement, then Java and ActiveX assign colors to symbols differently than does the SAS server. To ensure consistency on all devices, you should specify the desired color of each symbol. The SAS server restarts at the first color in the colors list and rotates through all of the colors in the color list for the first default symbol before going to the next symbol in the default symbol list where it again rotates through all of the colors in the color list before picking up the next symbol.

Note: Neither the Java applet nor the ActiveX control supports using COLOR= with PROC GCONTOUR. Δ

See also: “Using Color” on page 206

Not supported by: Java (partial), ActiveX (partial)

CV=*value-color*

specifies a color for

- plot symbols in the GPLOT procedure
- the filled areas generated by the INTERPOL=*map/plot-pattern* option
- contour labels in the GCONTOUR procedure.

If you omit CV= but specify CI=, CV= assumes the value of CI=. In this case, CV= and CI= specify the same color, which is the same as specifying COLOR= alone.

If you omit CV=, the color specification is searched for in this order:

- 1 the CI= option
- 2 the COLOR= option
- 3 the CSYMBOL= option in a GOPTIONS statement
- 4 each color in the colors list sequentially before the next SYMBOL definition is used.

Note: Neither the Java applet nor the ActiveX control supports using CV= with PROC GCONTOUR. Δ

See also: “Using Color” on page 206

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226, “Example 5. Filling the Area between Plot Lines” on page 236, and “Example 4. Creating and Modifying Box Plots” on page 233.

Not supported by: Java (partial), ActiveX (partial)

FONT=*font*

F=*font*

specifies the font for the plot symbol (GPLOT) or contour-label text (GCONTOUR) specified by VALUE=. The *font* specification can be

- the name of a software font. For example, FONT=MARKER specifies a software font that is stored in the catalog SASHELP.FONTS.
- a hardware font specification of the form HW*xxxxnnn* or *hardware-font-name:*

HW*xxxxnnn*

HW identifies the font as a hardware font, *xxx* are the last two or three characters of the module name as listed in the Module field in the device entry's Detail window, and *nnn* is the Chartype number of the hardware font as listed in the device entry's Chartype window (for example, FONT=HWDMX001).

hardware-font-name

specifies the name of a hardware font as shown in the device entry's Chartype window (for example, FONT="Palatino-Italic"). The name must be enclosed in double quotation marks.

By default, no font is specified. The symbol specified by VALUE= is taken from the special symbol table shown in Figure 7.21 on page 202. To use symbols from the special symbol table, omit FONT=.

You can use FONT= to specify a symbol font, such as Marker, that contains a symbol that you want to use in your plot. In this case, the string specified by VALUE= is the character code for the symbol. For example, this definition specifies a heart:

```
symbol font=marker value=N;
```

You can also use FONT= to specify a text font, such as Swiss. In this case, the string specified by VALUE= appears in the plot:

```
symbol font=swiss value=star;
```

Here, the word "star" is displayed in the plot.

To cancel a font specification and return to the default special symbol table, enter a null value:

```
symbol font=, value=star;
```

See also: the VALUE= on page 199 option, "Specifying Plot Symbols" on page 205, and Chapter 5, "SAS/GRAPH Fonts," on page 75.

Featured in: Example 2 on page 906

Not supported by: Java, ActiveX

HEIGHT=*symbol-height*<units>

H=*symbol-height*<units>

specifies the height in number of units of plot symbols (GPLOT) or contour labels (GCONTOUR).

Note: HEIGHT= affects only the height of the symbols and labels on the plot; it does not affect the height of any symbols that may appear in a legend.

The HEIGHT option overrides the MarkerSize attribute in graph styles. For more information on graph styles, see *SAS Output Delivery System: User's Guide*. △

Note: With client-side rendering using Java, the minimum height is 2 pixels; with ActiveX a symbol may be so small as to be invisible.

Neither the Java applet nor the ActiveX control supports HEIGHT= with PROC GCONTOUR. △

See also: the option SHAPE= on page 156 in the LEGEND statement

Featured in: "Example 4. Creating and Modifying Box Plots" on page 233, "Example 3. Rotating Plot Symbols through the Colors List" on page 231, and Example 2 on page 906.

Not supported by: Java (partial), ActiveX (partial)

INTERPOL=BOX<*option(s)*><00...25>

I=BOX<*option(s)*><00...25>

produces box and whisker plots. The bottom and top edges of the box are located at the sample 25th and 75th percentiles. The center horizontal line is drawn at the 50th percentile (median). By default, INTERPOL=BOX, in which case the vertical lines, or whiskers, are drawn from the box to the most extreme point within 1.5 interquartile ranges. (An interquartile range is the distance between the 25th and the 75th sample percentiles.) Any value more extreme than this is marked with a plot symbol.

Values for *option(s)* are one or more of these:

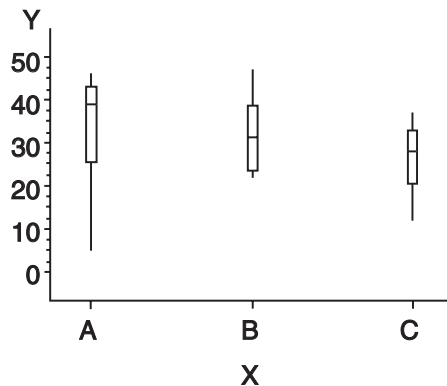
- F fills the box with the color specified by CV= and outlines the box with the color specified by CO=
- J joins the median points of the boxes with a line
- T draws tops and bottoms on the whiskers.

In addition, you can specify a percentile to control the length of the whiskers within the range 00 through 25. These are examples of percentile specifications and their effect:

- 00 high/low extremes. INTERPOL=BOX00 is *not* the same as the default, INTERPOL=BOX.
- 01 1st percentile low, 99th high
- 05 5th percentile low, 95th high
- 10 10th percentile low, 90th high
- 25 25th percentile low, 75th high; since the box extends from the 25th to the 75th percentile, no whiskers are produced.

Figure 7.15 on page 188 shows the type of plot INTERPOL=BOX produces.

Figure 7.15 Box Plot



Note: If you use HAXIS= or VAXIS= in the PLOT statement or ORDER= in an AXIS definition to restrict the range of axis values, by default any observations that fall outside the axis range are excluded from the interpolation calculation. See the MODE= option on page 197 Δ

You cannot use the GPLOT procedure PLOT statement option AREAS= with INTERPOL=BOX.

To increase the thickness of all box plot lines, including the box, whiskers, join line, and top and bottom ticks, use the WIDTH= option.

To increase the width of the box itself, use the BWIDTH= option. By default the value of BWIDTH= is the same as the value of WIDTH=. Therefore, if you specify a value for WIDTH= and omit BWIDTH=, the width of the box changes.

For a scatter effect with the box, use a multiple plot request, as in this example:

```
symbol1 i=none v=star color=green;
symbol2 i=box v=none color=blue;
proc gplot data=test;
  plot (y y)*x / overlay;
```

This option cannot be used in a symbol definition that is named in the GPLOT procedure, when that procedure is generating output for the Web using a Java device driver. This applies only when the PLOT statement is used with the OVERLAY option, or when the PLOT2 statement is used, with or without the OVERLAY option.

Featured in: “Example 4. Creating and Modifying Box Plots” on page 233.

Not supported by: Java (partial)

INTERPOL=HILO<C><option>

I=HILO<C><option>

specifies that a solid vertical line connect the minimum and maximum Y values for each X value. The data should have at least two values of Y for every value of X; otherwise, the single value is displayed without the vertical line.

By default, for each X value, the mean Y value is marked with a tick. This is shown in Figure 7.16 on page 190.

To specify high, low, close stock market data, include this option:

C draws tick marks at the close value instead of at the mean value. Specifying C assumes that there are three values of Y (HIGH, LOW, and CLOSE) for every value of X. If more or fewer than three Y values are specified, the mean is ticked. The Y values can be in any order in the input data set.

In addition, you can specify one of these values for *option*:

B connects the minimum and maximum Y values with bars instead of lines. Use the BWIDTH= option to increase the width of the bars.

J joins the mean values or the close values (if HILOC is specified) with a line. This point is not marked with a tick mark. You cannot use the PLOT statement option AREAS= with INTERPOL=HILOJ.

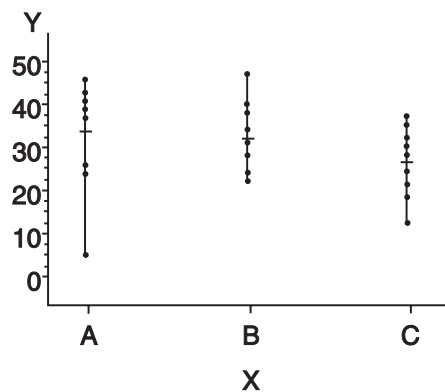
T adds tops and bottoms to each line.

BJ connects maximum and minimum values with a bar and joins the mean or close values.

TJ adds tops and bottoms to the lines and joins the mean or close values.

Figure 7.16 on page 190 shows the type of plot INTERPOL=HILO produces. Plot symbols in the form of dots have been added to this figure.

Figure 7.16 High-Low Plot



To increase the thickness of all lines generated by the INTERPOL=HILO option, use the WIDTH= option.

Note: If you use HAXIS= or VAXIS= in the PLOT statement or ORDER= in an AXIS definition to restrict the range of axis values, by default any observations that fall outside the axis range are excluded from the interpolation calculation. See the option MODE= on page 197. Δ

This option cannot be used in a symbol definition that is named in the GPLOT procedure, when that procedure is generating output for the Web using a Java device driver. This applies only when the PLOT statement is used with the OVERLAY option, or when the PLOT2 statement is used, with or without the OVERLAY option.

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226.

Not supported by: Java (partial)

INTERPOL=JOIN

I=JOIN

connects data points with straight lines. Points are connected in the order they occur in the input data set. Therefore, the data should be sorted by the independent (horizontal axis) variable.

If the data contain missing values, the observations are omitted. However, the plot line is not broken at missing values unless the SKIPMISS option is used.

See also: the SKIPMISS on page 1112 option and “Missing Values” on page 1087.

INTERPOL=L<degree><P><S>

I=L<degree><P><S>

specifies a Lagrange interpolation to smooth the plot line. Specify one of these values for *degree*:

1 | 3 | 5 specifies the degree of the Lagrange interpolation polynomial.
By default, *degree* is 1.

In addition, you can specify one or both of these:

P specifies a parametric interpolation

S sorts a data set by the independent variable before plotting its data.

The Lagrange methods are useful chiefly when data consist of tabulated, precise values. A polynomial of the specified degree (1, 3, or 5) is fitted through the nearest 2, 4, or 6 points. In general, the first derivative is not continuous. If the

values of the horizontal variable are not strictly increasing, the corresponding parametric method (L1P, L3P, or L5P) is used.

Specifying INTERPOL=L1P, INTERPOL=L3P, or INTERPOL=L5P results in a parametric Lagrange interpolation of degree 1, 3, or 5, respectively. Both the horizontal and vertical variables are processed with the Lagrange method and a parametric interpolation of degree 1, 3, or 5, using the distance between points as a parameter.

INTERPOL=*map/plot-pattern*

I=*map/plot-pattern*

specifies that a pattern fill the polygon that has been defined by the data points. Values for *map/plot-pattern* are

MEMPTY

ME

an empty pattern. EMPTY and E are valid aliases.

The Java applet does not support this option.

MSOLID

MS

a solid pattern. SOLID and S are valid aliases

Mdensity<*style*<*angle*>>

a shaded pattern. (The Java applet does not support this option.)

Density specifies the density of the pattern's shading:

1...5 1 produces the lightest shading and 5 produces the heaviest.

Style specifies the direction of pattern lines:

N parallel lines (the default)

X crosshatched lines.

Angle specifies the starting angle for parallel or crosshatched lines:

0...360 the degree at which the parallel lines are drawn. By default, *angle* is 0 (lines are parallel to the horizontal axis).

The INTERPOL=*map/plot-pattern* option only works if the data are structured so that the data points and, consequently, the plot lines form an enclosed area. The plot lines should not cross each other.

See also: the "PATTERN Statement" on page 169

Featured in: "Example 5. Filling the Area between Plot Lines" on page 236

Not supported by: Java (partial)

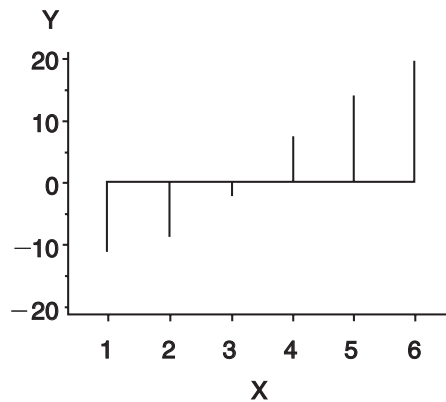
INTERPOL=NEEDLE

I=NEEDLE

draws a vertical line from each data point to a horizontal line at the 0 value on the vertical axis or the minimum value on the vertical axis if it is greater than 0. The horizontal line is drawn automatically.

Figure 7.17 on page 192 shows the type of plot INTERPOL=NEEDLE produces. Plot symbols are not displayed in this figure.

Figure 7.17 Needle Plot



You cannot use the PLOT statement option AREAS= with INTERPOL=NEEDLE.

INTERPOL=NONE

I=NONE

suppresses any interpolation and, if VALUE= is not specified, also suppresses plot points. If no interpolation method is specified in a SYMBOL statement and if the graphics option INTERPOL= is not used, INTERPOL=NONE is the default.

You cannot use the PLOT statement option AREAS= with INTERPOL=NONE.

INTERPOL=R<type><0><CLM | CLI<50...99>>

I=R<type><0><CLM | CLI<50...99>>

specifies that a plot is a regression analysis. By default, regression lines are not forced through plot origins and confidence limits are not displayed.

Type specifies the type of regression. Specify one of these values for *type*:

L requests linear regression representing the regression equation

$$Y = \beta_0 + \beta_1 X$$

Q requests quadratic regression representing the regression equation

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2$$

C requests cubic regression representing the regression equation

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$$

Note: When least-square solutions for the parameters are not unique, the SAS/GRAPH server defaults to a quadratic equation for the interpolation whereas the Java client and ActiveX client might pick a cubic solution to use. Δ

By default, *type* is L. The regression line is drawn in the line type specified in the LINE= option. By default, the type of the regression line is 1.

Note: You must specify *type* if you use either 0, or CLI, or CLM. Δ

To force the regression line through a (0,0) origin, specify:

0 eliminates the β_0 parameter, or intercept, from the regression equation. If the origin is at (0,0), also forces the regression line through the origin. For example, if you specify 0 for a linear regression, the plot line represents the equation

$$Y = \beta_1 X$$

Note: To force the regression line through the origin (0,0) when the data ranges do not place the origin at (0,0), use the GPLOT procedure options HZERO and VZERO (ignored if the data contain negative values), or use HAXIS and VAXIS to specify axes ranges from 0 to maximum data value. If the data ranges contain negative values and HAXIS and VAXIS specify ranges starting at 0, only values within the displayed range are used in the interpolation calculations. Δ

To display confidence limits, specify one of these:

CLM displays confidence limits for mean predicted values

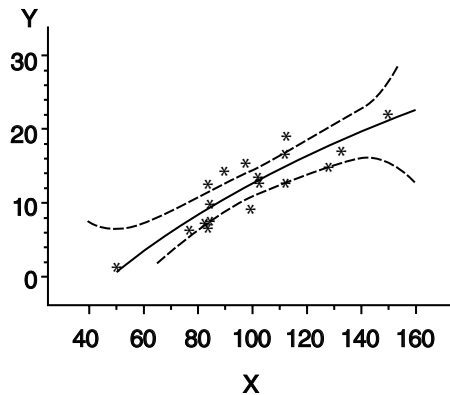
CLI displays confidence limits for individual predicted values.

You can specify confidence levels from 50% to 99%. By default, the confidence level is 95%. Include a confidence level specification only if you use CLM or CLI.

The line type used for the confidence limit lines is determined by adding 1 to the values of LINE=. By default, the line type of confidence limit lines is 2.

Figure 7.18 on page 193 shows the type of plot INTERPOL=RCCLM95 produces (cubic regression analysis with 95% confidence limits).

Figure 7.18 Plot of Regression Analysis and Confidence Limits



Featured in: Example 4 on page 1126.

Not supported by: Java (partial)

INTERPOL=SM<nn><P><S>

I=SM<nn><P><S>

specifies that a smooth line is fit to data using a spline routine. INTERPOL=SM is a method for smoothing noisy data. The points on the plot do not necessarily fall on the line.

The relative importance of plot values versus smoothness is controlled by *nn*. Values for *nn* are

0...99 produces a cubic spline that minimizes a linear combination of the sum of squares of the residuals of fit and the integral of the square of the second derivative (Reinsch 1967)*. The greater the *nn* value, the smoother the fitted curve. By default, the value of *nn* is 0.

* Reinsch, C.H. (1967), "Smoothing by Spline Functions," *Numerische Mathematik*, 10, 177–183.

In addition, specify one or both of these:

- P specifies a parametric cubic spline
 S sorts data by the independent variable before plotting.

Not supported by: Java

INTERPOL=SPLINE<P><S>

I=SPLINE<P><S>

specifies that the interpolation for the plot line use a spline routine.

INTERPOL=SPLINE produces the smoothest line and is the most efficient of the nontrivial spline interpolation methods.

Spline interpolation smoothes a plot line using a cubic spline method with continuous second derivatives (Pizer 1975)**This method uses a piecewise third-degree polynomial for each set of two adjacent points. The polynomial passes through the plotted points and matches the first and second derivatives of neighboring segments at the points.

Specify one or both of these:

- P specifies a parametric spline interpolation method. This interpolation uses a parametric spline method with continuous second derivatives. Using the method described earlier for the spline interpolation, a parametric spline is fitted to both the horizontal and vertical values. The parameter used is the distance between points

$$t = \sqrt{(x^2 + y^2)}$$

If two points are so close together that the computations overflow, the second point is not used.

- S sorts a data set by the independent variable before plotting its data.

Note: When points on the graph are out of range of the axis values, the curve is clipped. If an end point is out of range, no curve is drawn. Out-of-range conditions may be caused by restricting the range of axis values with the HAXIS= or VAXIS= option in the PLOT statement or the ORDER= option in an AXIS definition.

Note: When points on the graph are close together and a spline interpolation is used, the Java applet is unable to draw some line types correctly. \triangle

\triangle

INTERPOL=STD<1 | 2 | 3><variance><option(s)>

I=STD<1 | 2 | 3><variance><option(s)>

specifies that a solid line connect the mean Y value with $\pm 1, 2,$ or 3 standard deviations for each X.

Note: By default, 2 standard deviations are used. \triangle

** Pizer, Stephen M. (1975), *Numerical Computing and Mathematical Analysis*, Chicago: Science Research Associates, Inc., Chapter 4.

The sample variance is computed about each mean, and from it, the standard deviation s_y is computed. *Variance* can be one or both of these:

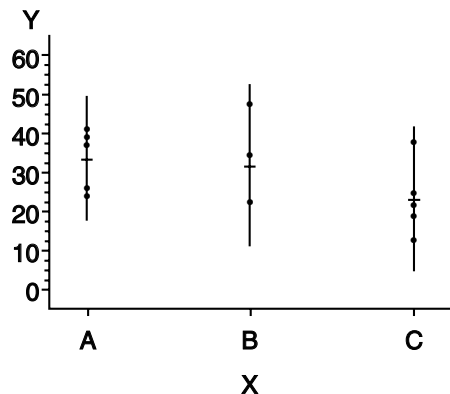
- M computes $s_{\bar{y}}$,
 P computes sample variances using a pooled estimate, as in a one-way ANOVA model.

In addition, specify one of these values for *option(s)*:

- B connects the minimum and maximum Y values with bars instead of lines.
 J connects the means from bar to bar with a line.
 T adds tops and bottoms to each line.
 BJ connects maximum and minimum values with a bar and joins the mean values.
 TJ adds tops and bottoms to the lines and joins the mean values.

Figure 7.19 on page 195 shows the type of plot INTERPOL=STD produces. A horizontal tick is drawn at the mean. Plot symbols in the form of dots have been added to this figure.

Figure 7.19 Plot of Standard Deviations



Note: By default, the vertical axis ranges from the minimum to the maximum Y value in the data. If the requested number of standard deviations from the mean covers a range of values that exceeds the maximum or is less than the minimum, the STD lines are cut off at the minimum and maximum Y values. When this cutoff occurs, rescale the axis using VAXIS= in the PLOT statement or ORDER= in an AXIS definition so that the STD lines are shown. Δ

If you restrict the range of axis values by using HAXIS= or VAXIS= in a PLOT statement or ORDER= in an AXIS definition, by default any observations that fall outside the axis range are excluded from the interpolation calculation. See the MODE= on page 197 option.

To increase the thickness of all lines generated by the INTERPOL=STD option, use the WIDTH= option.

You cannot use the PLOT statement option AREAS= with INTERPOL=STD.

This option cannot be used in a symbol definition that is named in the GPLOT procedure, when that procedure is generating output for the Web using a Java device driver. This applies only when the PLOT statement is used with the

OVERLAY option, or when the PLOT2 statement is used, with or without the OVERLAY option.

Not supported by: Java (partial)

INTERPOL=STEP<placement><J><S>

I=STEP<placement><J><S>

specifies that the data are plotted with a step function. By default, the data point is on the left of the step, the steps are not joined with a vertical line, and the data are not sorted before processing.

Specify one of these values for *placement*:

- | | |
|---|--|
| L | displays the data point on the left of the step. |
| R | displays the data point on the right of the step. |
| C | displays the data point in the center of the step. |

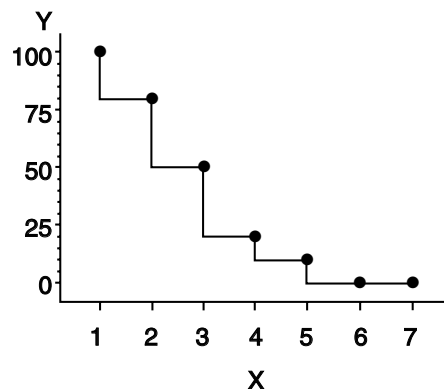
Note: When a step is retraced in order to locate its center point, both the server and Java treat this as effectively not drawing that part of the step at all. ActiveX, however, draws each part of the step—resulting in a somewhat differently appearing graph. Δ

In addition, specify one or both of these:

- | | |
|---|---|
| J | produces steps joined with a vertical line. |
| S | sorts unordered data by the independent variable before plotting. |

Figure 7.20 on page 196 shows the type of plot INTERPOL=STEPJR produces. Plot symbols in the form of dots have been added to this figure.

Figure 7.20 Step Plot



LINE=*line-type*

L=*line-type*

specifies the line type of the plot line in the GPLOT procedure, or the contour line in the GCONTOUR procedure:

- | | |
|-------|----------------|
| 1 | a solid line. |
| 2..46 | a dashed line. |

Line types are shown in Figure 7.22 on page 208. By default, LINE=1.

Note: This option overrides the LineStyle attribute in graph styles. For more information on graph styles, see *SAS Output Delivery System: User's Guide*.

Neither the Java applet nor ActiveX control supports client-side rendering for GCONTOUR. △

Not supported by: Java (partial), ActiveX (partial)

MODE=EXCLUDE | INCLUDE

specifies that interpolation calculations exclude or include data values that are outside the range of plot axes. By default, MODE=EXCLUDE, which excludes values outside the axis range from any calculations.

If you control the range of values displayed on an axis by using HAXIS= and VAXIS= in the GPLOT procedure, or ORDER= in an AXIS definition, any data points that lie outside of the range of the axes are discarded before the calculations are done for interpolation lines. This has a particularly noticeable effect on the high-low interpolation methods, which include INTERPOL=HILO, INTERPOL=BOX, and INTERPOL=STD. Regression analysis also represents only part of the original data.

See also: “Values Out of Range” on page 1087.

POINTLABEL<=(label-description(s)) | NONE>

labels plot points. The labels always use the format that is assigned to the variable(s) whose values are used for the labels. POINTLABEL without any specified descriptions labels points with the Y value. NONE suppresses the point labels. *Label-description(s)* can be used to change the variable whose values are used to label points, and/or to change features of the label text, such as the color, font, or size of the text.

Note: If you do not specify a color on a SYMBOL statement, the symbol definition is rotated through the colors list before the next SYMBOL statement is used. Thus, if your plot contains multiple plot lines and you want to limit your POINTLABEL specification to a single line, you must specify a color on the SYMBOL statement that contains the POINTLABEL description. △

Label-description(s) can be one or more of these:

COLOR=*text-color*

C=*text-color*

specifies the color of the label text. The default is the first color from the colors list.

FONT=*font* | NONE

F=*font* | NONE

specifies the font for the text. See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for details on specifying *font*. If you omit FONT=, a font specification is searched for in this order:

- 1 the FTEXT= option in a GOPTIONS statement
- 2 the default hardware font, NONE.

HEIGHT=*text-height* <units >

H=*text-height* <units >

specifies the height of the text characters in number of units. By default, HEIGHT=1 CELL. If you omit HEIGHT=, a text height specification is searched for in this order:

- 1 the HTEXT= option in a GOPTIONS statement
- 2 the default value, 1.

JUSTIFY=CENTER | LEFT | RIGHT

J=C | L | R

specifies the horizontal alignment of the label text. The default is CENTER. The location of the point label is relative to the location of the corresponding data point.

POSITION=TOP | MIDDLE | BOTTOM

specifies the vertical placement of the label text. The default is TOP. The location of the point label is relative to the location of the corresponding data point.

"#var" | "#x:#y <\$char>" | "#y:#x <\$char>"

specifies the variable(s) whose values will label the plot points. The variable specification must be enclosed in either single or double quotation marks. The first specified variable must be prefixed with a pound sign (#). If a second variable is specified, it must be prefixed with a colon and a pound sign (:#). Optionally, when you specify both the X and Y variables, you can specify the character to display as the delimiter between variable values in the plot label.

By default if POINTLABEL is specified without naming a label variable, the Y values label the plot points. You can change the default by using "#var" to specify a different variable whose values should label the points. For example, you might specify the name of the X variable. The following option specifies the variable SALES as the variable whose values will label plot points:

```
POINTLABEL=("#sales")
```

Alternatively, you can label the plot points with the values of the X and Y variables, in either order. The order that you specify X and Y in the variable specification determines the order that the values are displayed in the label. The following option specifies variables HEIGHT and WEIGHT; in the label, the value for HEIGHT will be displayed, followed by the value for WEIGHT:

```
POINTLABEL=("#height:#weight")
```

The variables that you specify must be the plot's X and Y variables. Specifying any other variables will cause unexpected labeling.

By default when you specify both the X and Y variables, a colon (:) displays in the label to separate the values in each label. To change the character that displays as the delimiter, use the \$ syntax to specify an alternative character. The following option specifies a vertical bar (|) as the delimiter in the label:

```
POINTLABEL=("#height:#weight $|")
```

The \$ syntax must be within the same quotation marks as the variable specification. The \$ specification can precede or follow the variable specification, but it must be separated from the variable specification by at least one space.

Note: Specifying a delimiting character with the \$ only changes the character that displays in the label. It does not change the syntax of the variable specification, which requires a colon and pound sign (:#) to precede the second variable. \triangle

Note: There is a sixteen character length limit for each variable. A maximum character length limit of thirty-three characters is possible. This can be composed of X and Y variables, any other valid data set variable, and a separator as required. \triangle

Specify as many label-description suboptions as you want. Enclose them all within a single set of parentheses, and separate each suboption from the others by at least one space.

Not supported by: Java (partial), ActiveX (partial)

REPEAT=*number-of-times*

R=*number-of-times*

specifies the number of times that a SYMBOL definition is applied before the next SYMBOL definition is used. By default, REPEAT=1.

The behavior of REPEAT= depends on whether any of the SYMBOL color options (CI=, CV=, CO=, and COLOR=) or the CSYMBOL= graphics option also is used:

- If any SYMBOL color option also is used in the SYMBOL definition, that SYMBOL definition is repeated the specified number of times in the specified color.
- If no SYMBOL color option is used but the CSYMBOL= graphics option is currently in effect, the SYMBOL definition is repeated the specified number of times in the specified color.
- If no SYMBOL statement color options are used and the CSYMBOL= graphics option is not used, the SYMBOL definition is cycled through each color in the colors list, and then the entire group generated by this cycle repeats the number of times specified by REPEAT=. Thus, the total number of iterations of the SYMBOL definition depends on the number of colors in the current colors list.

Neither the Java applet nor ActiveX control supports client-side rendering for GCONTOUR.

See also: “Using the SYMBOL Statement” on page 202.

Not supported by: Java (partial), ActiveX (partial)

SINGULAR=*n*

tunes the algorithm used to check for singularities. The default value is machine dependent but is approximately 1E-7 on most machines. This option is rarely needed.

STEP=*distance*<*units*>

specifies the minimum distance between labels on contour lines. The value of *distance* must be greater than zero. By default, STEP=65PCT.

Note: If you specify units of PCT or CELLS, STEP= calculates the distance between the labels based on the width of the graphics output area, not the height. For example, if you specify STEP=50PCT and if the graphics output area is 9 inches wide, the distance specified is 4.5 inches. A value less than 10 percent is ignored and 10 percent is used instead. Δ

When you use STEP=, specify the minimum distance that you want between labels. The option then calculates how many labels it can fit on the contour line, taking into account the length of the labels and the minimum distance you specified. Once it has calculated how many labels it can fit while retaining the minimum distance between them, it places the labels, evenly spaced, along the line. Consequently, the space between labels may be greater than what you specify, although it will never be less.

In general, to increase the number of labels from the default, reduce the value of *distance*.

If the procedure cannot write the label at a particular location on the contour, for example because the contour line makes a sharp turn, the label may be placed farther along the line or omitted. If labels are omitted, a note appears in the log. Specifying a low value for the GCONTOUR procedure’s TOLANGLE= option may also cause labels to be omitted, since this forces the procedure to select smoother labeling locations, which may not be available on some contours.

Featured in: Example 2 on page 906.

Not supported by: Java, ActiveX

VALUE=*special-symbol* | *text-string* | NONE

V=*special-symbol* | *text-string* | NONE

- specifies a plot symbol for the data points (GPLOT and GBARLINE). If you omit the SYMBOL statement, plot points are generated using the default plot symbol. The default symbol is a square if you use the ActiveX or Java devices and a PLUS sign for other devices. If you specify a SYMBOL statement, but do not specify the VALUE= option, plot symbols are suppressed.

Note: For ActiveX output, the VALUE= option is not supported when INTERPOL=HILO or INTERPOL=STD. You can use the OVERLAY option with GPLOT to get symbols to appear on the data points. Δ

- specifies contour-label text in a contour plot (GCONTOUR). By default with the AUTOLABEL option, GCONTOUR labels contour lines with the contour variable's value at that contour level.
- VALUE=NONE suppresses plot symbols at the data points, or labels on the contour lines. You can set the VALUE=NONE option independent of the INTERPOL= option.

Values for *special-symbol* are the names and characters shown in Figure 7.21 on page 202. The special symbol table can be used only if the FONT= option is not used or a null value is specified:

```
font=,
```

Note: To specify a single quotation mark, you must enclose it in double quotation marks: Δ

```
value="'"
```

To specify a double quotation mark, you must enclose it in single quotation marks:

```
value='"'
```

In some operating environments, punctuation characters may require single quotes.

If you use VALUE=*text-string* to specify a plot symbol, you must also use the FONT= option to specify a symbol font or a text font. If you specify a symbol font, the characters in the string are character codes for the symbols in the font. If you specify a text font, the characters in the string are displayed. If you specify a text string containing quotes or blanks, enclose the string in single quotes.

For example, if you specify this statement, the plot symbol is the word "plus" instead of the symbol +:

```
symbol font=swiss value=plus;
```

Java and ActiveX support the following characters from the marker font for *special-symbol*:

Character	Aliases
Marker	Cone, Pyramid, Default
Square	Cube
Star	
Circle	Sphere, Dot, Balloon
Plus	Cross
Flag	Y

Character	Aliases
X	
Prism	Z
Spade	“
Heart	#
Diamond	\$
Club	%
Hexagon	Paw
Cylinder	Hash

Note: If you do not use a SYMBOL statement to specify a color for each symbol, but you do specify a colors list in a GOPTIONS statement, then Java and ActiveX assign colors to symbols differently than does the SAS server. To ensure consistency on all devices, you should specify the desired color of each symbol. The SAS server restarts at the first color in the colors list and rotates through all of the colors in the color list for the first default symbol before going to the next symbol in the default symbol list where it again rotates through all of the colors in the color list before picking up the next symbol. \triangle

Note: The VALUE option overrides the MarkerSymbol attribute in graph styles. For more information on graph styles, see *SAS Output Delivery System: User's Guide*. \triangle

See also: the option FONT= on page 186 and “Specifying Plot Symbols” on page 205.

Featured in: “Example 3. Rotating Plot Symbols through the Colors List” on page 231, “Example 4. Creating and Modifying Box Plots” on page 233, and Example 2 on page 906.

Not supported by: Java (partial), ActiveX (partial)

WIDTH=*thickness-factor*

W=*thickness-factor*

specifies the thickness of interpolated lines (GPLOT) or contour lines (GCONTOUR), where *thickness-factor* is a number. The thickness of the line increases directly with *thickness-factor*. By default, WIDTH=1.

WIDTH= also affects all the lines in box plots (INTERPOL=BOX), high-low plots with bars (INTERPOL=HILOB), and standard deviation plots (INTERPOL=STD). It also affects the outlines of the area generated by the AREAS= option in the PLOT statement of the GPLOT procedure.

Note: By default, the value specified by WIDTH= is used as the default value for the BWIDTH= option. For example, specifying WIDTH=6 also sets BWIDTH= to 6 unless you explicitly assign a value to BWIDTH=.

Java and ActiveX do not provide the same measure of control for width as SAS/GRAPH on the server. Measurements are translated to pixels rather than a percentage. \triangle

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226 and “Example 4. Creating and Modifying Box Plots” on page 233.

Not supported by: Java (partial) and ActiveX (partial)

Figure 7.21 Special Symbols for Plotting Data Points

VALUE=	Plot Symbol	VALUE=	Plot Symbol
PLUS	+	% (percent)	♁
X	×	& (ampersand)	♁
STAR	*	' (single quote)	♁
SQUARE	□	= (equals)	☆
DIAMOND	◇	- (hyphen)	⊙
TRIANGLE	△	@ (at)	♀
HASH	#	* (asterisk)	♀
Y	Y	+ (plus)	⊕
Z	Z	> (greater than)	♂
PAW	⋯	. (period)	℥
POINT	.	< (less than)	℥
DOT	●	, (comma)	⊕
CIRCLE	○	/ (slash)	♁
_ (underscore)	◻	? (question mark)	℥
" (double quote)	♠	((left parenthesis)	♁
# (pound sign)	♥) (right parenthesis)	♁
\$ (dollar sign)	◇	: (colon)	♁

Note: The words or special characters in the VALUE= column are entered exactly as shown. Δ

Using the SYMBOL Statement

A SYMBOL statement specifies one or more options that indicate the color and other attributes used by the GPLOT procedure or the GCONTOUR procedure. For GPLOT, the main attributes include the plot symbol, interpolation method, and type of plot line. For GCONTOUR, the main attributes include the type of contour lines used and the text used to label those lines.

Note: SYMBOL statements can only be applied to contour plots when the AUTOLABEL option is specified on GCONTOUR. Δ

You can define up to 99 different SYMBOL statements. A SYMBOL statement without a number is treated as a SYMBOL1 statement.

SYMBOL definitions can be defined anywhere in your SAS program. They are global and remain in effect until canceled or until you end your SAS session. Once defined, SYMBOL definitions can be

- assigned by default by GPLOT or explicitly selected with the plot request
- used by GCONTOUR to control the labels and attributes of contour lines.

SYMBOL statements generate one or more symbol definitions, depending on how color is used and whether a plot symbol or type of contour line is specified. For more

information, see “Controlling Consecutive SYMBOL Statements” on page 203 and “Using Generated Symbol Sequences” on page 208.

Although it is common practice, you do not have to start with SYMBOL1, and you do not have to use sequential statement numbers. When assigning SYMBOL definitions, SAS/GRAPH software starts with the lowest-numbered definition and works upward, ignoring gaps in the numbering.

Altering or Canceling SYMBOL Statements

SYMBOL statements are additive. If you define a SYMBOL statement and later submit another SYMBOL statement with the same number, the new SYMBOL statement defines or cancels only the options that are included in the new statement. Options that are not included in the new statement are not changed and remain in effect.

Assume you define SYMBOL4 as:

```
symbol4 value=star cv=red height=4;
```

The following statement cancels only HEIGHT= without affecting the rest of the definition:

```
symbol4 height=;
```

Add or change options in the same way. This statement adds an interpolation method to SYMBOL4:

```
symbol4 interpol=join;
```

This statement changes the color of the plot symbol from red to blue:

```
symbol4 cv=blue;
```

After all these modifications, SYMBOL4 has these characteristics:

```
symbol4 value=star cv=blue interpol=join;
```

Cancel individual SYMBOL statements by defining a SYMBOL statement of the same number without options (a null statement):

```
symbol4;
```

Canceling one SYMBOL statement does not affect any other SYMBOL definitions. To cancel all current SYMBOL statements, use RESET= in a GOPTIONS statement:

```
goptions reset=symbol;
```

Specifying RESET=GLOBAL or RESET=ALL cancels all current SYMBOL definitions as well as other settings.

To display current SYMBOL definitions in the LOG window, use the GOPTIONS procedure with the SYMBOL option:

```
proc goptions symbol nolist;
run;
```

Controlling Consecutive SYMBOL Statements

If you specify consecutively numbered SYMBOL statements and you want SAS/GRAPH to use each definition only once, use color specifications to ensure each SYMBOL statement generates only one symbol definition. You can

- specify colors on each SYMBOL statement, using the COLOR=, CI=, CV=, or CO= options. This method lets you explicitly assign colors for each definition. For example, these statements generate two definitions:

```
symbol1 value=star color=green;
symbol2 value=square color=yellow;
```

- specify a default color for all SYMBOL statements using the CSYMBOL= option on the GOPTIONS statement. This method makes it easy to specify the same color for each definition when you do not need more explicit color specifications.
- limit the colors list to a single color using the COLORS= option on the GOPTIONS statement. This method makes it easy to specify the same color for each definition when you want the color to apply to other definitions also, such as PATTERN definitions.

For more information on specifying colors for symbol definitions, see “Using Color” on page 206.

If you do not use color to limit a SYMBOL statement to a single symbol definition, SAS/GRAPH generates multiple symbol definitions from that statement by rotating the current definition through the colors list (for more details, see “Using Generated Symbol Sequences” on page 208). Because SAS/GRAPH uses symbol definitions in the order they are generated, this means that the n th symbol definition applied to a graph does not necessarily correspond to the SYMBOL n statement.

For example, assuming no color is specified on the CSYMBOL= graphics option, these statements generate four definitions:

```
goptions colors=(red blue green);
symbol1 value=star;
symbol2 value=square color=yellow;
```

Because no color is specified on SYMBOL1, SAS/GRAPH rotates the symbol definition through the colors list, which has three colors. Thus, SYMBOL1 defines the first three applied symbol definitions, and SYMBOL2 defines the 4th:

Sequence Number	Source	Characteristics: Color	Symbol
1	SYMBOL1	red	star
2	SYMBOL1	blue	star
3	SYMBOL1	green	star
4	SYMBOL2	yellow	square

In this case, if a graph needs only three symbols, the SYMBOL2 definition is not used.

To make the n th applied symbol definition correspond to the SYMBOL n statement, limit each SYMBOL statement to a single color, using one of the techniques listed at the beginning of this section.

Setting Definitions for PROC GPLOT

The following topics apply only for SYMBOL statements used with PROC GPLOT:

- specifying plot symbols
- specifying default interpolation methods
- sorting data with spline interpolation.

Specifying Plot Symbols

The VALUE= option specifies the plot symbols that PROC Gplot uses to mark the data points on a plot. Plot symbols can be

- special symbols from Figure 7.21 on page 202
- characters from symbol fonts
- text strings.





By default, the plot symbol is the + symbol. To specify a special symbol, use VALUE= to specify a name or a character from Figure 7.21 on page 202:

```
symbol1 value=hash color=green;
symbol2 value=) color=blue;
```

This example uses color to ensure each SYMBOL statement generates only one definition. You can omit color specifications to let SAS/GRAPH rotate symbol definitions through the colors list. For details, see “Using Generated Symbol Sequences” on page 208.

To use plot symbols other than those in Figure 7.21 on page 202, use the FONT= option to specify a font for the plot symbol. If the font is a symbol font, such as Marker, the string specified with the VALUE= option is the character code for the symbol to be displayed. If the font is a text font, the string specified with VALUE= is displayed as the plot symbol. (See VALUE= on page 199 and FONT= on page 186.)

This table illustrates some of the ways you can define a plot symbol:

Definition	Plot Symbol
symbol1 value=plus;	
symbol2 value=+;	
symbol3 font=swiss value=plus;	plus
symbol4 font=marker value=U;	
symbol5 value="";	

Specifying a Default Interpolation Method

The INTERPOL= option in a GOPTIONS statement specifies a default interpolation method to be used with all SYMBOL definitions. This default interpolation method is in effect unless you specify a different interpolation in a SYMBOL statement. If the GOPTIONS statement does not specify an interpolation method, the default for each SYMBOL statement is NONE.

Sorting Data with Spline Interpolation

If you want the Gplot procedure to sort by the horizontal axis variable before plotting, add the letter S to the end of any of the spline interpolation methods (INTERPOL=L, INTERPOL=SM, and INTERPOL=SPLINE). For example, suppose you want to overlay three plots (Y1*X1, Y2*X2, and Y3*X3) and for each plot, you want the X variable sorted in ascending order. Use these statements:

```
symbol1 i=splines c=red;
symbol2 i=splines c=blue;
```

```

symbol3 i=splines c=green;

proc gplot;
  plot y1*x1 y2*x2 y3*x3 / overlay;
run;

```

Using Color

Generally, there are two ways to explicitly specify color for SYMBOL statements:

- specify colors on the SYMBOL statements
- specify a color on the CSYMBOL= graphics option.

You can also let SAS/GRAPH rotate symbol definitions through the colors list. For details, see “Using Generated Symbol Sequences” on page 208.

Specifying Colors with SYMBOL Statements

The SYMBOL statement has these options for specifying color:

- The CV= option specifies color for plot symbols in GPLOT, or for contour labels in GCONTOUR.
- The CO= option specifies color for confidence limit lines and area outlines in GPLOT.
- The CI= option specifies color for plot lines in GPLOT, or contour lines in GCONTOUR.
- The COLOR= option specifies color for the entire symbol. For GPLOT, this includes plot symbols, plot lines, and outlines. For GCONTOUR, this includes contour lines and labels.

CV= and CI= have the same effect as using COLOR= when they are used in these ways:

- Only CV= or CI= option is used. (The option that is not used is assigned the value of the option used.)
- Both CV= and CI= specify the same color.

In general, CI=, CV=, and CO= color specific areas of the symbol. Use these options to produce symbols and plot lines of different colors without having to overlay multiple plot pairs. For example, if you request regression analysis with confidence limits, use this statement to assign red to the plot symbol, blue to the regression lines, and green to the confidence limit lines:

```

symbol cv=red ci=blue co=green;

```

The COLOR= option colors the entire symbol or those portions of it not colored by one of the other color options. If COLOR= precedes CI= or CV=, the CI= or CV= specification is used instead. If none of the SYMBOL color options is used, color specifications are searched for in this order:

- 1 the CSYMBOL= option in a GOPTIONS statement
- 2 each color in the colors list sequentially before the next SYMBOL definition is used.

CAUTION:

If no color options are used, the SYMBOL definition cycles through each color in the colors list. \triangle

If the SYMBOL color options and the CSYMBOL= graphics option are not used, the SYMBOL definition cycles through each color in the colors list before the next definition is used. For details, see “Using Generated Symbol Sequences” on page 208.

Specifying Color with CSYMBOL=

The CSYMBOL= option on the GOPTIONS statement specifies the default color to be used by all SYMBOL definitions:

```
goptions csymbol=green;  
symbol1 value=star;  
symbol2 value=square;
```

In this example, both SYMBOL statements use green.

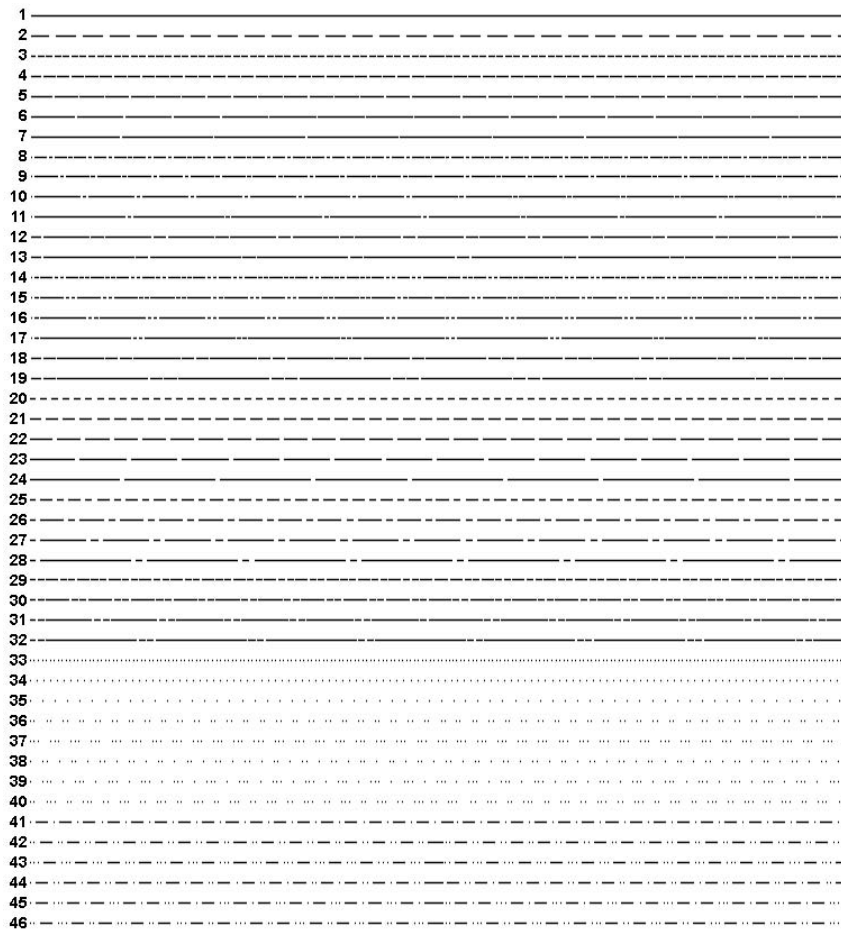
CSYMBOL= is overridden by any of the SYMBOL statement color options. See “Using Color” on page 206 for details.

If more SYMBOL definitions are needed, SAS/GRAPH returns to generating default symbol sequences.

Specifying Line Types

To specify the type of line for plot or contour lines, use the LINE= option to specify a number from 1 through 46. Figure 7.22 on page 208 shows the line types represented by these numbers. By default, the line type is 1 for plot and contour lines, and 2 for confidence limit lines.

Figure 7.22 Line Types



Note: These line types are also used by other statements and procedures. Some options accept a line type of 0, which produces no line. Δ

Using Generated Symbol Sequences

Symbol sequences are sets of SYMBOL definitions that are automatically generated by SAS/GRAPH software if any of these conditions is true:

- no valid SYMBOL definition is available. In this case, default symbol sequences are generated by rotating symbol definitions through the color specified on the GOPTIONS statement's CSYMBOL= option. If a CSYMBOL= color is not in effect, the definitions are rotated through the colors list.
- a SYMBOL statement specifies color but not a plot symbol for the GPLOT procedure, or a line type for the GCONTOUR procedure (assuming GCONTOUR does not specify the needed line types). In this case, a default plot symbol or line type is used with the specified color and only one definition is generated.
- a SYMBOL statement specifies a plot symbol for GPLOT or a line type for GCONTOUR, but no color options. In this case, the specified plot symbol or line type is used once with the color specified by the CSYMBOL= graphics option. If a

CSYMBOL= color is not in effect, the specified plot symbol or line type is rotated through the colors list.

If REPEAT= is also used, the resulting SYMBOL definition is repeated the specified number of times.

Default Symbol Sequences

Default symbol sequences are generated by rotating symbol definitions through the current colors list.

- Definitions used for GPLOT rotate plot symbols through the colors list; the first default plot symbol is a plus sign (+).
- Definitions used for GCONTOUR rotate line types; the first default line type is a solid line (line type 1).

Each time a default definition is required, SAS/GRAPH takes the first default plot symbol or line type and uses it with the first color in the colors list. If more than one definition is required, it uses the same plot symbol or line type with the next color in the colors list and continues until all the colors have been used once. If more definitions are needed, SAS/GRAPH selects the second default plot symbol or line type and rotates it through the colors list. It continues in this fashion, selecting default plot symbols or line types and cycling them through the colors list until all the required definitions are generated.

If a color has been specified with the CSYMBOL= option on the GOPTIONS statement, each default plot symbol or line type is used once with the specified color, and the colors in the colors list are ignored.

Symbol Sequences Generated from SYMBOL Statements

If a SYMBOL statement does not specify color, and if the CSYMBOL= graphics option is not used, the symbol definition is rotated through every color in the colors list before the next SYMBOL definition is used:

```
goptions colors=(blue red green);
symbol1 cv=red i=join;
symbol2 i=spline v=dot;
symbol3 cv=green v=star;
```

Here, the SYMBOL1 statement generates the first SYMBOL definition. The SYMBOL2 statement does not include color, so the first default plot symbol is rotated through all colors in the colors list before the SYMBOL3 statement is used. This table shows the colors and symbols that would be used if nine symbol definitions were required for PROC GPLOT:

Sequence		Characteristics:		
Number	Source	Color	Symbol	Interpolation
1	SYMBOL1	cv=red	first default	join
2	SYMBOL2	color=blue	dot	spline
3	SYMBOL2	color=red	dot	spline
4	SYMBOL2	color=green	dot	spline
5	SYMBOL3	cv=green	star	NONE
6	first default	color=blue	first default	default
7	first default	color=red	first default	default

Sequence		Characteristics:		
Number	Source	Color	Symbol	Interpolation
8	first default	color=green	first default	default
9	second default	color=blue	second default	default

Notice that after the SYMBOL statements are exhausted, the procedure begins using the default definitions (sequences 6 through 9). Each plot symbol from the default list is rotated through all colors in the colors list before the next plot symbol is used. Also, SYMBOL1 does not specify a plot symbol, so the default sequencing provides the first default symbol (a + sign). When sequencing resumes in sequence number 6, it starts at the beginning again, selecting the first default plot symbol and rotating it through the colors list.

If you use REPEAT= but no color, the sequence generated by cycling the definition through the colors list is repeated the number of times specified by REPEAT=. For example, these statements define a colors list and illustrate the effect of REPEAT= on SYMBOL statements both with and without explicit color specifications:

```
goptions colors=(blue red green);
symbol1 color=gold repeat=2;
symbol2 value=star color=cyan;
symbol3 value=square repeat=2;
```

Here, SYMBOL1 is used twice, SYMBOL2 is used once, and SYMBOL3 rotates through the list of three colors and then repeats this cycle a second time:

Sequence		Characteristics:		
Number	Source	Color	Symbol	Interpolation
1	SYMBOL1	gold	first default	default
2	SYMBOL1	gold	first default	default
3	SYMBOL2	cyan	star	default
4	SYMBOL3	blue	square	default
5	SYMBOL3	red	square	default
6	SYMBOL3	green	square	default
7	SYMBOL3	blue	square	default
8	SYMBOL3	red	square	default
9	SYMBOL3	green	square	default

TITLE, FOOTNOTE, and NOTE Statements

The TITLE, FOOTNOTE, and NOTE statements control the content, appearance, and placement of text.

Used by:

GBARLINE, GCHART, GCONTOUR, GFONT, GMAP, GPLOT, GPRINT,
GRADAR, GSLIDE, G3D procedures

Global: TITLE and FOOTNOTE

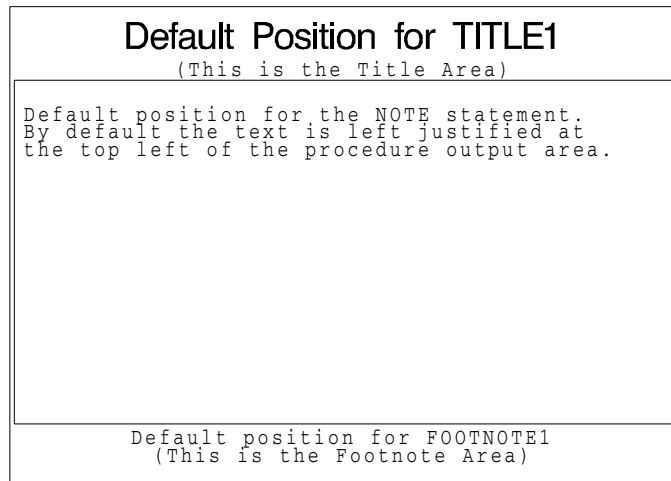
Description

TITLE, FOOTNOTE, and NOTE statements add text to maps, plots, charts, and text slides. With these statements you can

- control the content, appearance, and placement of the text, including color, size, font, and alignment
- underline or draw boxes around the text
- draw straight lines on the output.

Figure 7.23 on page 211 shows the default appearance and placement of titles, footnotes, and notes on the graphics output area.

Figure 7.23 Default Placement of Titles, Footnotes, and Notes



Titles are centered at the top of the graphics output in the *title area*. They are positioned in numeric order with the lowest-numbered TITLE at the top of the title area and the highest-numbered TITLE at the bottom of the title area.

TITLE statements have these default characteristics:

- TITLE1 is twice the height of all other titles and uses the SWISS font.
- All other TITLE statements are one unit high and use the default hardware font.

Footnotes are positioned similarly in the *footnote area* at the bottom of the graphics output area, with the lowest numbered FOOTNOTE at the top of the footnote area. Unless otherwise specified, they use the default hardware font and are one unit high.

Space for the title area and the footnote area is taken from the procedure output area. The more titles and footnotes you specify and the bigger they are, the smaller the procedure output area will be.

Notes are positioned at the top of the procedure output area and are left justified. The statements appear one below another in the order they appear in the program. Unless otherwise specified, they use the default hardware font and are one unit high.

For more information on titles, footnotes, and notes in the graphics output area, see “Placement of Graphic Elements in the Graphics Output Area” on page 39.

Syntax

TITLE<1...10><*text-argument(s)*>;

FOOTNOTE<1...10><*text-argument(s)*>;

NOTE<*text-arguments(s)*>;

text-argument(s) can be one or more of these:

'text-string'

text-options (text options must precede text-string.)

text-options can be one or more of the following, in any order:

- appearance options
 - COLOR=*color*
 - FONT=*font*
 - HEIGHT=*text-height*<*units*>
- placement and spacing options
 - JUSTIFY=LEFT | CENTER | RIGHT
 - LSPACE=*line-space*<*units*>
 - MOVE=(*x,y*)<*units*>
- baseline angling and character rotation options
 - ANGLE=*degrees*
 - LANGLE=*degrees*
 - ROTATE=*degrees*
- boxing, underlining, and line drawing options
 - BCOLOR=*background-color*
 - BLANK=YES
 - BOX=1...4
 - BSPACE=*box-space*<*units*>
 - DRAW=(*x,y*...,*x-n,y-n*)<*units*>
 - UNDERLIN=0...3
- linking option
 - LINK= '*url*'

Options

When the syntax of an option includes *units*, use one of these:

CELLS	character cells
CM	centimeters
IN	inches
PT	points
PCT	percentage of the graphics output area

If you omit *units*, a unit specification is searched for in this order:

- 1 the GUNIT= option in a GOPTIONS statement
- 2 the default unit, CELLS.

ANGLE=*degrees*

A=*degrees*

specifies the angle of the *baseline* of the entire text string with respect to the horizontal. A positive *degrees* value will angle the baseline counterclockwise; a negative value will angle it clockwise. By default, ANGLE=0 (horizontal).

Angled titles or footnotes may require more vertical space and, consequently, may increase the size of the title area or the footnote area, thereby reducing the vertical space in the procedure output area.

Using the BOX= option with angled text does not produce angled boxes; the box is sized to accommodate the angled note.

Using the ANGLE= option after one text string and before another can reset some options to their default values. See “Using Options That Can Reset Other Options” on page 225.

ANGLE= has the same effect on the text as LANGLE=, except when you specify an angle of 90 degrees or -90 degrees. In these angle specifications, the procedure output area is shrunk from the left or right to accommodate the angled title or footnote. The result depends on the statement in which you use the option:

- *With the TITLE statement:*

Figure 7.24 on page 213 shows how ANGLE=90 degrees or ANGLE=-90 degrees positions and rotates title text.

ANGLE=90

positions the title at the left edge of the graphics output area, angled 90 degrees (counterclockwise) and centered vertically.

ANGLE=-90

positions the title at the right edge of the graphics output area, angled -90 degrees (clockwise) and centered vertically.

Figure 7.24 Positioning Titles with the ANGLE= Option



- *With the FOOTNOTE statement:*

Figure 7.25 on page 214 shows how ANGLE=90 degrees or ANGLE=-90 degrees positions and rotates footnote text.

ANGLE=90

positions the footnote at the right edge of the graphics output area, angled 90 degrees (counterclockwise) and centered vertically.

ANGLE=-90

positions the footnote at the left edge of the graphics output area, angled -90 (clockwise) and centered vertically.

Figure 7.25 Positioning Footnotes with the ANGLE=Option



- *With the NOTE statement:*

Figure 7.26 on page 214 shows how ANGLE= 90 degrees or -90 degrees positions and rotates note text.

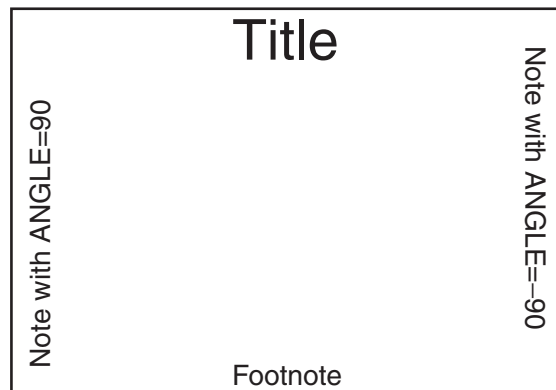
ANGLE=90

positions the note at the bottom of the left edge of the graphics output area, angled 90 degrees (counterclockwise) and reading from bottom to top.

ANGLE=-90

positions the note at the top of the right edge of the graphics output area, angled -90 (clockwise) and reading from top to bottom.

Figure 7.26 Positioning Notes with the ANGLE= Option



See also: the options LANGLE= on page 219 and ROTATE= on page 222

Featured in: “Example 6. Enhancing Titles” on page 238

Not supported by: Java, ActiveX

BCOLOR=*background-color*

BC=*background-color*

specifies the background color of a box produced by the BOX= option. If you omit BOX=, BCOLOR= is ignored. By default, the background color of the box is the same as the background color for the entire graph. The color of the frame of the box is determined by the color specification used in BOX=.

Note: BCOLOR= may be reset by ANGLE= or JUSTIFY=, or by MOVE= with absolute coordinates. See “Using Options That Can Reset Other Options” on page 225 for details. △

See also: the option BOX= on page 215.

Featured in: “Example 6. Enhancing Titles” on page 238.

BLANK=YES

BL=YES

protects the box and its contents from being overwritten by any subsequent graphics elements by blanking out the area where the box is displayed. BLANK= enables you to overlay graphics elements with boxed text. It is ignored if you omit BOX=. Because titles and footnotes are written from the highest numbered to the lowest numbered, the BLANK= option only blanks out titles and footnotes of a lower number.

Note: BLANK= may be reset by ANGLE= or JUSTIFY=, or by MOVE= with absolute coordinates. See “Using Options That Can Reset Other Options” on page 225 for details. △

See also: the option BOX= on page 215.

Featured in: “Example 6. Enhancing Titles” on page 238.

Not supported by: Java, ActiveX

BOX=1...4

BO=1...4

draws a box around one line of text. A value of 1 produces the thinnest box lines; 4 produces the thickest. Boxing angled text does not produce an angled box; the box is sized to include the angled text.

The color of the box is either:

- the color specified by the COLOR= option in the statement
- the default text color.

COLOR= affects only the frame of the box. To color the background of the box, use BCOLOR=.

You can include more than one text string in the box as long as no text break occurs between the strings; that is, you cannot use JUSTIFY= to create multiple lines of text within a box.

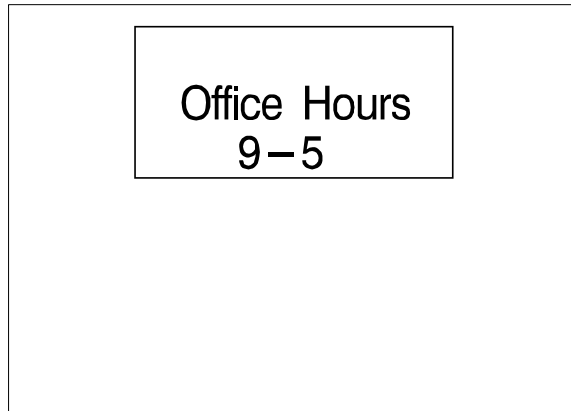
To draw a box around multiple lines of text, you can either

- Use MOVE= with relative coordinates to position the lines of text where you want them and enclose them with BOX=. For example, this statement produces the boxed note shown in Figure 7.27 on page 216:

```
note font=swiss justify=center box=3
      'Office Hours'      move=(40pct,-12pct) '9-5';
```

- Use the DRAW= option to draw the box and do not use BOX=.

Figure 7.27 Using the BOX= Option and the MOVE= Option to Box Multiple Lines of Text



Note: BOX= may be reset by ANGLE= or JUSTIFY=, or by MOVE= with absolute coordinates. See “Using Options That Can Reset Other Options” on page 225 for details. \triangle

See also: the options BCOLOR= on page 215, BLANK= on page 215, and BSPACE= on page 216.

Featured in: “Example 6. Enhancing Titles” on page 238

Not supported by: Java, ActiveX

BSPACE=*box-space*<units>

BS=*box-space*<units>

specifies the amount of space between the boxed text and the box. The space above the text is measured from the font maximum, and the space below the text is measured from the font minimum. By default, BSPACE=1. If BOX= is not used, BSPACE= is ignored.

The spacing is uniform around the box. For example, BSPACE=.5IN leaves one-half inch of space between the text and the top, bottom, and sides of the box.

Note: BSPACE= may be reset by ANGLE= or JUSTIFY=, or by MOVE= with absolute coordinates. See “Using Options That Can Reset Other Options” on page 225 for details. \triangle

See also: the option BOX= on page 215.

Not supported by: Java, ActiveX

COLOR=*color*

C=*color*

specifies the color for the following text, box, or line. COLOR= affects all text, lines, and boxes that follow it and stays in effect until another COLOR= specification is encountered.

Change colors as often as you like. For example, this statement produces a title with red text in a box with a blue frame and a cream background:

```
title color=red 'Total Sales' color=blue
  box=3 bcolor=cream;
```

Although BCOLOR= controls the background color of the box, the frame color is controlled with the COLOR= that precedes BOX=.

If you omit COLOR=, a color specification is searched for in this order:

- 1 the CTITLE= option in a GOPTIONS statement
- 2 the CTEXT= option in a GOPTIONS statement
- 3 the default, the first color in the colors list.

See also: the option BCOLOR= on page 215.

“Controlling Titles and Footnotes with ODS Output” on page 492

DRAW=(*x,y...x-n,y-n*)<*units*>

D=(*x,y...x-n, y-n*)<*units*>

draws lines anywhere on the graphics output area using *x* and *y* as absolute or relative coordinates. The following table shows the specifications for absolute and relative coordinates:

Absolute Coordinates	Relative Coordinates
<i>x</i> < <i>units</i> >	± <i>x</i> < <i>units</i> >
<i>y</i> < <i>units</i> >	± <i>y</i> < <i>units</i> >

The coordinate position (0,0) is the lower-left corner of the graphics output area. Specify at least two coordinate pairs. Commas between coordinates are optional; blanks can be used instead. DRAW= does not affect the positioning of text.

The starting point for lines specified with relative coordinates begins at the end of the most recently drawn text or line in the current statement. If no text or line has been drawn in the current statement, a warning is issued and the relative draw is measured from where a zero-length text string would have ended, given the normal placement for the statement.

You can mix relative and absolute coordinates. For example, DRAW=(+0,+0,+0,1IN) draws a vertical line from the end of the text to one inch from the bottom of the graphics output area.

Not supported by: Java, ActiveX

FONT=*font*

F=*font*

specifies the font for the subsequent text. See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for details on specifying SAS/GRAPH fonts. If you omit this option, a font specification is searched for in this order:

- for a TITLE1 statement
 - 1 the FTITLE= option in a GOPTIONS statement
 - 2 the FTEXT= option in a GOPTIONS statement
 - 3 the default font, SWISS (COMPLEX in Release 6.06 and earlier).
- for all other TITLE statements and the FOOTNOTE and NOTE statements:
 - 1 the FTEXT= option in a GOPTIONS statement
 - 2 the default hardware font, NONE.

Note: Font names greater than eight characters in length must be enclosed in quotation marks. △

Note: If the TITLE or FOOTNOTE is being output through an ODS markup destination and the corresponding NOGTITLE or NOGFOOTNOTE option is specified, then the *bold* and *italic* FONT attributes are on by default. However, if you specify different attributes with the FONT= option, the *bold* and *italic* attributes are turned off. △

See also: “Controlling Titles and Footnotes with ODS Output” on page 492

Featured in: “Example 6. Enhancing Titles” on page 238.

`HEIGHT=text-height<units>`

`H=text-height<units>`

specifies the height of text characters in number of units. By default, `HEIGHT=1`. Height is measured from the font minimum to the capline. Ascenders may extend above the capline, depending on the font.

If your text line is too long to be displayed in the height specified in `HEIGHT=`, the height specification is reduced so that the text can be displayed. A note in the SAS log tells you what percentage of the specified size was used.

If you omit `HEIGHT=`, a text height specification is searched for in this order:

□ *for a TITLE1 statement:*

- 1 the `HTITLE=` option in a `GOPTIONS` statement
- 2 the `HTEXT=` option in a `GOPTIONS` statement
- 3 the default value, 2.

By default, a `TITLE1` title is twice the height of all other titles.

□ *for all other TITLE statements and the FOOTNOTE and NOTE statements:*

- 1 the `HTEXT=` option in a `GOPTIONS` statement
- 2 the default value, 1.

Note: The Java applet and ActiveX control allow you to control the relative height of text with the `HEIGHT=` option, but not the absolute height in terms of specific units. \triangle

See also: “Controlling Titles and Footnotes with ODS Output” on page 492

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226 and “Example 6. Enhancing Titles” on page 238.

Not supported by: Java (partial), ActiveX (partial)

`JUSTIFY=LEFT | CENTER | RIGHT`

`J=L | C | R`

specifies the alignment of the text string. The default depends on the statement with which you use `JUSTIFY=`:

- *for a FOOTNOTE statement* the default is `CENTER`
- *for a NOTE statement* the default is `LEFT`
- *for a TITLE statement* the default is `CENTER`.

All the text strings following `JUSTIFY=` are treated as a single string and are displayed as one line that is left-, right-, or center-aligned.

You can change the justification within a single line of text. For example, this `NOTE` statement displays a date on the left side of the output and the page number on the same line on the right:

```
note 'June 28, 1997' justify=right 'Page 3';
```

In addition, you can use `JUSTIFY=` to produce multiple lines of text by repeating `JUSTIFY=` with the same value before the text string for each line. Multiple lines of text with the same justification are blocked together. For example, this `TITLE` statement produces a three-line title with each line right-justified:

```
title justify=right 'First Line'
      justify=right 'Second Line'
      justify=right 'Third Line';
```

You can get the same effect with three `TITLE` statements, each specifying `JUSTIFY=RIGHT`. If you produce a block of text by specifying the same

justification for multiple text strings, and then change the justification for an additional text string, that text is placed on the same line as the first string specified in the statement.

Note: Using JUSTIFY= after one text string and before another can reset some options to their default values. See “Using Options That Can Reset Other Options” on page 225 for details. \triangle

Featured in: “Example 3. Rotating Plot Symbols through the Colors List” on page 231.

LANGLE=*degrees*

LA=*degrees*

specifies the angle of the *baseline* of the entire text string(s) with respect to the horizontal. A positive value for *degrees* moves the baseline counterclockwise; a negative value moves it clockwise. By default, LANGLE=0 (horizontal).

Angled titles or footnotes may require more vertical space and consequently may increase the size of the title area or the footnote area, thereby reducing the vertical space in the procedure output area.

Using BOX= with angled text does not produce an angled box; the box is sized to accommodate the angled note.

Unlike ANGLE=, LANGLE= does not reset any other options. Therefore, LANGLE= is easier to use because you do not need to repeat options after a text break.

LANGLE= has the same effect on the text as ANGLE=, except when an angle of 90 degrees or -90 degrees is specified. The result depends on the statement in which you use the option:

- \square *With the TITLE statement:*

Figure 7.28 on page 219 shows how LANGLE=90 degrees and LANGLE=-90 degrees positions and rotates titles.

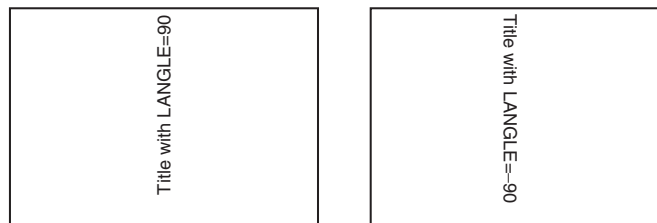
LANGLE=90

angles the title 90 degrees (counterclockwise) so that it reads from bottom to top. The title is centered horizontally and positioned at the top of the picture.

LANGLE=-90

angles the title -90 degrees (clockwise) so that it reads from top to bottom. The title is centered horizontally and positioned at the top of the picture.

Figure 7.28 Positioning Titles with the LANGLE= Option



- *With the FOOTNOTE statement:*

Figure 7.29 on page 220 shows how `LANGLE=90` degrees and `LANGLE=-90` degrees positions and rotates footnotes.

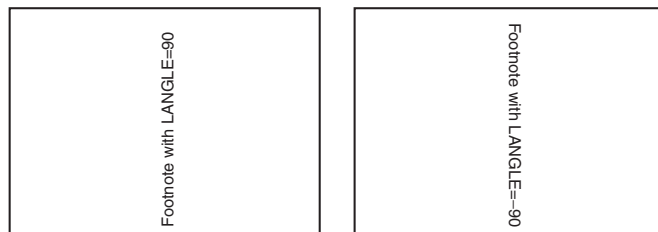
`LANGLE=90`

angles the footnote 90 degrees (counterclockwise) so that it reads from bottom to top. The footnote is centered horizontally and positioned as the bottom of the picture.

`LANGLE=-90`

angles the footnote -90 degrees (clockwise) so that it reads from top to bottom. The footnote is centered horizontally and positioned at the bottom of the picture.

Figure 7.29 Positioning Footnotes with the `LANGLE=` Option



- *With the NOTE statement:*

Figure 7.30 on page 220 shows how `LANGLE=90` degrees and `LANGLE=-90` degrees positions and rotates notes.

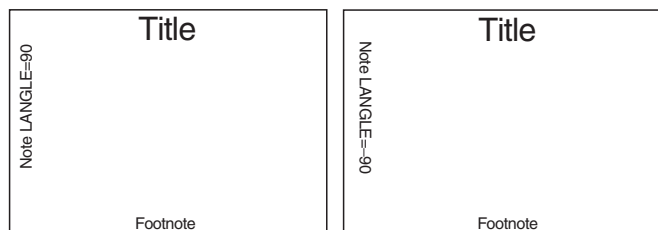
`LANGLE=90`

positions the note at the top of the left edge of the procedure output area, angled 90 degrees (counterclockwise) so that it reads from bottom to top.

`LANGLE=-90`

positions the note at the top of the left edge of the procedure output area, angled -90 degrees (clockwise) so that it reads from top to bottom.

Figure 7.30 Positioning Notes with the `LANGLE=` Option



See also: the option `ANGLE=` on page 213.

Not supported by: Java, ActiveX

`LINK= 'url'`

specifies a uniform resource locator (url) that a title or footnote links to.

The text-string that you use to specify the url can contain occurrences of the variables `#BYVAL`, `#BYVAR`, and `#BYLINE`, as described in text-string on page 222.

Note: If the title or footnote is being output through an ODS markup destination (such as HTML) and the corresponding ODS option NOGTITLE or NOGFOOTNOTE is specified, then the title or footnote is rendered in the body of the HTML file rather than in the graphic itself. Specifying NOGTITLE or NOGFOOTNOTE results in increasing the amount of space allowed for the procedure output area, which can result in increasing the size of the graph. Space that would have been used for the title or footnote is devoted instead to the graph. You might need to be aware of this possible difference if you are using annotate or map coordinates. Δ

See also: “Controlling Where Titles and Footnotes are Rendered” on page 492

LSPACE=*line-space* <units>

LS=*line-space* <units>

specifies the amount of spacing *above* lines of note and title text and the amount of spacing *below* lines of footnote text. For notes and titles, the spacing is measured from the capline of the current line to the font minimum of the line above. For footnotes, the spacing is measured from the font minimum of the current line to the capline of the line below. By default, LSPACE=1.

Note: LSPACE= may be reset by ANGLE= or JUSTIFY=, or by MOVE= with absolute coordinates. See “Using Options That Can Reset Other Options” on page 225 for details. Δ

Not supported by: Java, ActiveX

MOVE=(*x,y*) <units>

M=(*x,y*) <units>

positions subsequent text or lines anywhere on the graphics output area using *x* and *y* as absolute or relative coordinates. The following table shows the specifications for absolute and relative coordinates:

Absolute Coordinates	Relative Coordinates
<i>x</i> <units>	\pm <i>x</i> <units>
<i>y</i> <units>	\pm <i>y</i> <units>

Commas between coordinates are optional; you can use blanks instead.

The starting point for lines specified with relative coordinates begins with the end of the most recently drawn text or line in the current statement. If no text or line has been drawn in the current statement, a warning is issued and the relative move is measured from where a zero-length text string would have ended, given the normal placement for the statement. You can mix relative and absolute coordinates.

MOVE= overrides a JUSTIFY= specified for the same text string.

If a NOTE, FOOTNOTE, or TITLE statement uses MOVE= to position the text so that the statement does not use its default position, the text of the next NOTE, FOOTNOTE, or TITLE statement occupies the unused position and no blank lines are displayed.

Note: If you specify MOVE= with at least one absolute coordinate and if the option follows one text string and precedes another, some options can be reset to their default values. See “Using Options That Can Reset Other Options” on page 225 for details. Δ

Featured in: “Example 2. Specifying Logarithmic Axes” on page 229 and “Example 6. Enhancing Titles” on page 238

Not supported by: Java, ActiveX

ROTATE=*degrees*

R=*degrees*

specifies the angle at which *each character* of text is rotated with respect to the baseline of the text string. The angle is measured from the current text baseline angle, which is specified by ANGLE= or LANGLE=. By default, the baseline is horizontal. A positive value for *degrees* rotates the character counterclockwise; a negative value rotates it clockwise. By default, ROTATE=0 (parallel to the baseline).

Figure 7.31 on page 222 shows how characters are positioned when ROTATE=90 is used with the default (horizontal) baseline.

Figure 7.31 Tilting Characters with the ROTATE= Option



See also: the option ANGLE= on page 213.

Featured in: “Example 6. Enhancing Titles” on page 238.

Not supported by: Java, ActiveX

text-string(s)

is one or more strings up to 200 characters. You must enclose text strings in single or double quotation marks. The text appears exactly as you type it in the statement, including uppercase and lowercase characters and blanks.

To use single quotation marks or apostrophes within the title, you can either

- use a pair of single quotation marks together:

```
footnote 'All''s Well That Ends Well';
```

- enclose the text in double quotation marks:

```
footnote "All's Well That Ends Well";
```

Because FOOTNOTE, NOTE, and TITLE statements concatenate all text strings, the strings must contain the correct spacing. With a series of strings, add blanks at the beginning of a text string rather than at the end, as in this example:

```
note color=red 'Sales:' color=blue ' 2000';
```

With some fonts, you produce certain characters by specifying a hexadecimal value. A trailing **x** identifies a string as a hexadecimal value. For example, this statement* produces the title **Profits Increase £ 3,000**:

```
title font=swiss 'Profits Increase ' '18'x '3,000';
```

For more information see “Specifying Special Characters” on page 81.

In addition, you can embed one or more of the following in the string:

#BYLINE

substitutes the entire BY line without leading or trailing blanks for **#BYLINE** in the text string, and displays the BY line in the footnote, note, or title produced by the statement.

* This statement assumes you are using a U.S. key map.

#BYVAL n | #BYVAL(*BY-variable-name*)

substitutes the current value of the specified BY variable for #BYVAL in the text string and displays the value in the footnote, note, or title produced by the statement. Specify the variable with one of these:

- | | |
|-------------------------|---|
| <i>n</i> | specifies which variable in the BY statement #BYVAL should use. The value of <i>n</i> indicates the position of the variable in the BY statement. For example, #BYVAL2 specifies the second variable in the BY statement. |
| <i>BY-variable-name</i> | names the BY variable. For example, #BYVAL(YEAR) specifies the BY variable, YEAR. <i>Variable-name</i> is not case sensitive. |

Featured in: “Example 7. Using BY-group Processing to Generate a Series of Charts” on page 240 and “Example 9. Combining Graphs and Reports in a Web Page” on page 248.

#BYVAR n | #BYVAR(*BY-variable-name*)

substitutes the name of the BY-variable or label associated with the variable (whatever the BY line would normally display) for #BYVAR in the text string and displays the name or label in the footnote, note, or title produced by the statement. Specify the variable with one of these:

- | | |
|-------------------------|---|
| <i>n</i> | specifies which variable in the BY statement #BYVAR should use. The value of <i>n</i> indicates the position of the variable in the BY statement. For example, #BYVAR2 specifies the second variable in the BY statement. |
| <i>BY-variable-name</i> | names the BY variable. For example, #BYVAR(SITES) specifies the BY variable, SITES. <i>Variable-name</i> is not case sensitive. |

A BY variable name displayed in a title, note, or footnote is always in uppercase. If a label is used, it appears as specified in the LABEL statement.

For more information, see “Substituting BY Line Values in a Text String” on page 226.

UNDERLIN=0...3**U=0...3**

underlines subsequent text. Values of 1, 2 and 3 underline with an increasingly thicker line. UNDERLIN=0 halts underlining for subsequent text.

Underlines follow the text baseline. If you use an LANGLE= or ANGLE= option for the line of text, the underline is drawn at the same angle as the text.

Underlines do not break up to follow rotated characters. See the option ROTATE= on page 222.

To make the text and the underline the same color, specify a COLOR= *before* the UNDERLIN= that precedes the text string. To make the text a different color, specify COLOR= *after* the UNDERLIN=.

Note: UNDERLIN= may be reset by ANGLE= or JUSTIFY=, option, or by the MOVE= option with absolute coordinates. See “Using Options That Can Reset Other Options” on page 225 for details.

Note: The Java applet and ActiveX control underline text when UNDERLIN= is specified, but they do not vary the thickness of the line. △

△

Featured in: “Example 6. Enhancing Titles” on page 238

Not supported by: Java (partial), ActiveX (partial)

Using TITLE and FOOTNOTE Statements

You can define TITLE and FOOTNOTE statements anywhere in your SAS program. They are global and remain in effect until you cancel them or until you end your SAS session. All currently defined FOOTNOTE and TITLE statements are automatically displayed.

You can define up to ten TITLE statements and ten FOOTNOTE statements in your SAS session. A TITLE or FOOTNOTE statement without a number is treated as a TITLE1 or FOOTNOTE1 statement. You do not have to start with TITLE1 and you do not have to use sequential statement numbers. Skipping a number in the sequence leaves a blank line.

You can use as many text strings and options as you want, but place the options before the text strings they modify. See “Using Multiple Options” on page 224.

The most recently specified TITLE or FOOTNOTE statement of any number completely replaces any other TITLE or FOOTNOTE statement of that number. In addition, it cancels all TITLE or FOOTNOTE statements of a higher number. For example, if you define TITLE1, TITLE2, and TITLE3, resubmitting the TITLE2 statement cancels TITLE3.

To cancel individual TITLE or FOOTNOTE statements, define a TITLE or FOOTNOTE statement of the same number without options (a null statement):

```
title4;
```

But remember that this will cancel all other existing statements of a higher number.

To cancel all current TITLE or FOOTNOTE statements, use the RESET= graphics option in a GOPTIONS statement:

```
goptions reset=footnote;
```

Specifying RESET=GLOBAL or RESET=ALL also cancels all current TITLE and FOOTNOTE statements as well as other settings.

Using the NOTE Statement

NOTE statements are local, not global, and they must be defined within a procedure or RUN-group with which they are used. They remain in effect for the duration of the procedure that includes NOTE statements in any of its RUN-groups or until you end your SAS session. All notes defined in the current RUN group, as well as those defined in previous RUN-groups, are displayed in the output as long as the procedure remains active.

You can use as many text strings and options as you want, but place the options before the text strings they modify. See “Using Multiple Options” on page 224.

Using Multiple Options

In each statement you can use as many text strings and options as you want, but you must place the options before the text strings they modify. Most options affect all text strings that follow them in the same statement, unless the option is explicitly reset to another value. In general, TITLE, FOOTNOTE, and NOTE statement options stay in effect until one of these events occurs:

- the end of the statement is reached
- a new specification is made for that option.

For example, this statement specifies that one part of the note be red and another part blue, but the height for all the text is 4:

```
note height=4 color=red 'Red Tide'
      color=blue ' Effects on Coastal Fishing';
```

Setting Defaults

You can set default characteristics for titles (including TITLE1 definitions), footnotes, and notes by using the following graphics options in a GOPTIONS statement:

CTITLE=*color*

sets the default color for all titles, footnotes, and notes; overridden by the COLOR= option in a TITLE, FOOTNOTE, or NOTE statement.

CTEXT=*text-color*

sets the default color for all text; overridden by CTITLE= for titles, footnotes, and notes.

FTITLE=*title-font*

sets the default font for TITLE1 definitions; overridden by FONT= in the TITLE1 statement.

FTEXT=*text-font*

sets the default font for all text, including the TITLE1 statement if FTITLE= is not used; overridden by the FONT= option a TITLE, FOOTNOTE, or NOTE statement.

HTITLE=*height<units>*

sets the default height for TITLE1 definitions; overridden by the HEIGHT= option in the TITLE1 statement.

HTEXT=*n<units>*

sets the default height for all text, including the TITLE1 statement if HTITLE= is not used; overridden by the HEIGHT= option a TITLE, FOOTNOTE, or NOTE statement.

See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for a complete description of each option.

Using Options That Can Reset Other Options

The ANGLE=, MOVE=, and JUSTIFY= options affect the position of the text and cause *text breaks*. (To cause a text break, MOVE= must have at least one absolute coordinate.) When a statement contains multiple text strings, the resulting text break can cause the following options to reset to their default values:

- BCOLOR=
- BLANK=
- BOX=
- BSPACE=
- LSPACE=
- UNDERLIN=.

Note: The LANGLE= option does not cause a text break. △

If in a TITLE, FOOTNOTE, or NOTE statement, before the first text string, you use an option that can be reset (such as UNDERLIN=) and before the second string you use an option that resets it (such as JUSTIFY=), the first option does not affect the second

string. In order for the first option to affect the second string, repeat the option and position it *after* the resetting option and *before* the text string.

For example, this statement produces a two-line title in which only the first line is underlined:

```
title underlin=2 'Line 1' justify=left 'Line 2';
```

To underline Line 2, repeat UNDERLIN= *before* the second text string and *after* JUSTIFY=:

```
title underlin=2 'Line 1' justify=left
      underlin=2 'Line 2';
```

Substituting BY Line Values in a Text String

To use the #BYVAR and #BYVAL options, insert the option in the text string at the position you want the substitution text to appear. Both #BYVAR and #BYVAL specifications must be followed by a delimiting character, either a space or other nonalphanumeric character, such as the quote that ends the text string. If not, the specification is completely ignored and its text remains intact and is displayed with the rest of the string. To allow a #BYVAR or #BYVAL substitution to be followed immediately by other text, with no delimiter, use a trailing dot (as with macro variables). The trailing dot is not displayed in the resolved text. If you want a period to be displayed as the last character in the resolved text, use two dots after the #BYVAR or #BYVAL substitution.

If you use a #BYVAR or #BYVAL specification for a variable that is not named in the BY statement (such as #BYVAL2 when there is only one BY-variable or #BYVAL(ABC) when ABC is not a BY-variable or does not exist), or if there is no BY statement at all, the substitution for #BYVAR or #BYVAL does not occur. No error or warning message is issued and the option specification is displayed with the rest of the string. The graph will continue to display a BY line at the top of the page unless you suppress it by using the NOBYLINE option in an OPTION statement.

For more information, see “BY Statement” on page 141.

Note: This feature is not available in the Data Step Graphics Interface or in the Annotate facility since BY lines are not created in a DATA step. \triangle

Example 1. Ordering Axis Tick Marks with SAS Datetime Values

Features:

AXIS statement options:

COLOR=

LABEL=

MAJOR=

MINOR=

OFFSET=

ORDER=

FOOTNOTE statement option:

HEIGHT=

SYMBOL statement options:

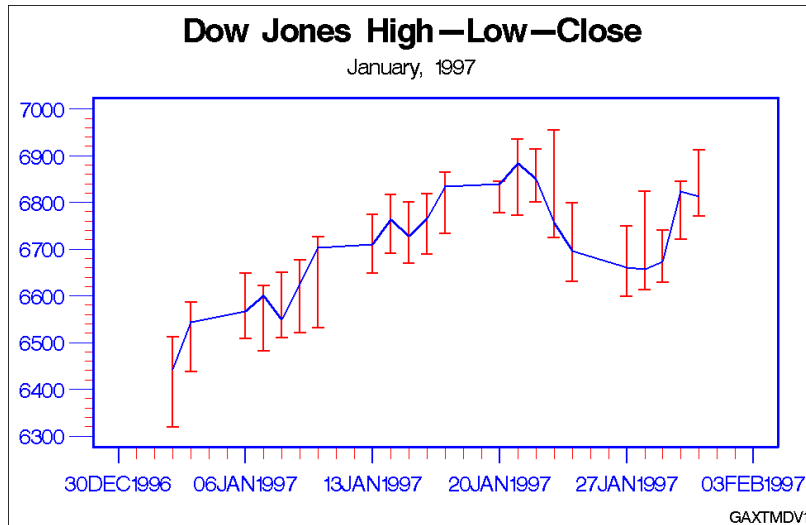
CI=


```

CV=
INTERPOL=
WIDTH=
GOPTIONS statement options:
FTITLE=
GUNIT=
HTEXT=
HTITLE=

```

Sample library member: GAXTMDV1



This example uses SAS datetime values with an `AXIS` statement's `ORDER=` option to set the major tick marks on the horizontal axis. It adjusts the position of the first and last major tick marks.

The example also uses `HILOCTJ` interpolation in a `SYMBOL` statement to join minimum and maximum values. The default unit specification for heights in the graph are percent of the graphics output area as specified by `GUNIT=` in the `GOPTIONS` statement. The `GOPTIONS` statement also specifies the default fonts for `TITLE1` and for other text.

Set the graphics environment. `GUNIT=` specifies the units in percent of the graphics output area. `HTITLE=` specifies the height for `TITLE1` text. `HTEXT=` specifies the height for all other text. `FTITLE=` specifies `SWISSB` as the font for `TITLE1`.

```

goptions reset=global gunit=pct border
        cback=white
        colors=(black blue green red)
        ftitle=swissb ftext=swiss htitle=6
        htext=4;

```

Create the data set. `DOWHLC` contains the high, low, and close values of the Dow Jones Industrial index for each business day for a month.

```

data dowhlc;
  input date date9. high low close;
  format date date9.;
  datalines;

```

```

02JAN1997    6511.38    6318.96    6442.49
03JAN1997    6586.42    6437.10    6544.09
...more data lines...
30JAN1997    6621.82    6481.75    6600.66
31JAN1997    6621.82    6481.75    6600.66
;

```

Prepare the data for a high-low plot. DOWHLC2 generates three records for each date, storing each date's high, low, and close values in variable DOW.

```

data dowhlc2;
  set dowhlc;
  drop high low close;
  dow=high; output;
  dow=low; output;
  dow=close; output;

```

Define titles and footnote. HEIGHT=3 in the FOOTNOTE statement overrides the height specified by HTEXT= in the GOPTIONS statement.

```

title1 'Dow Jones High-Low-Close';
title2 'January, 1997';
footnote height=3 justify=right 'GAXTMDV1

```

Define symbol characteristics. INTERPOL=HILOCTJ specifies that the minimum and maximum values of DOW are joined by a vertical line with a horizontal tick mark at each end. The close values are joined by straight lines. CV= colors the vertical lines, and CI= colors the line that joins the close values. WIDTH= controls the thickness of the line that joins the close points.

```

symbol interpol=hiloctj
  cv=blue
  ci=red
  width=2;

```

Define characteristics of the horizontal axis. ORDER= uses a SAS date value to set the major tick marks. OFFSET= moves the first and last tick marks to make room for the tick mark value. COLOR= makes all axis elements red. MAJOR= and MINOR= modify the size and color of the major and minor tick marks.

```

axis1 order=('30DEC96'd to '03FEB97'd by week)
  offset=(3,3)
  color=blue
  label=none
  major=(height=3 width=2)
  minor=(number=6 color=red height=2 width=1)
  width=3;

```

Define characteristics of the vertical axis. LABEL=NONE suppresses the AXIS label. The COLOR= suboption in MINOR= overrides the COLOR= option.

```

axis2 color=blue
  label=none
  major=(height=3)
  minor=(number=4 color=red height=1)
  offset=(2,2);

```

Generate the plot and assign AXIS definitions. HAXIS= assigns AXIS1 to the horizontal axis, and VAXIS= assigns AXIS2 to the vertical axis.

```
proc gplot data=dowhlc2;
  plot dow*date / haxis=axis1
                    vaxis=axis2;
run;
quit;
```

Example 2. Specifying Logarithmic Axes

Features:

AXIS statement options:

LABEL=
 LENGTH=
 LOGBASE=
 LOGSTYLE=
 MAJOR=
 MINOR=
 VALUE=

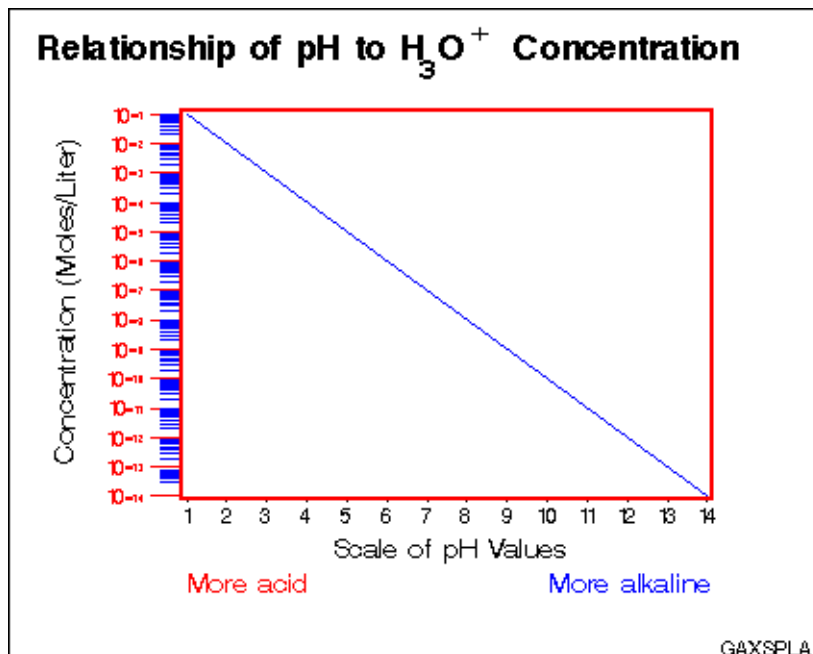
TITLE statement option:

MOVE=

GOPTIONS statement options:

GUNIT=
 VPOS=

Sample library member: GAXSPLA1



This example illustrates the AXIS statement options LOGBASE= and LOGSTYLE=. The horizontal axis represents pH level. The vertical axis, which represents the

concentration of the hydroxide ion expressed as moles per liter, is scaled logarithmically. In addition, this example shows how the TICK= parameter of the VALUE= option modifies individual tick marks.

The example uses the MOVE= option in a TITLE statement to position the title's subscript and superscript text.

Assign the libref and set the graphics environment. GUNIT= specifies units of percent of the graphics output area. VPOS= specifies a resolution for the vertical axis.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red) vpos=250
        ftitle=swissb ftext=swiss htitle=5
        htext=3;
```

Create the data set. CONCENTR contains the pH values and the concentration amount.

```
data concentr;
    input ph conc;
    datalines;
1  1E-1
2  1E-2
...more data lines...
13 1E-13
14 1E-14
;
```

Define title and footnote. MOVE= positions subscript 3 and superscript +. Each new position is relative to the last position specified by MOVE=.

```
title1 'Relationship of pH to H'
        move=(-0,-3) h=4 '3'
        move=(+0,+3) h=5 'O'
        move=(+0,+3) h=4 '+'
        move=(-0,-3) h=5 ' Concentration';
footnote j=r 'GAXSPLA1 ';
```

Define symbol characteristics.

```
symbol interpol=join color=blue;
```

Define characteristics for horizontal axis. LABEL= uses the JUSTIFY= suboption to create a descriptive two-line label that replaces the variable name PH. MINOR=NONE removes all minor tick marks. LENGTH= uses the units specified by the GUNIT= graphics option to control the length of the horizontal axis.

```
axis1 label=(h=4 'Scale of pH Values'
            justify=left color=red
            h=3 'More acid'
            justify=right color=blue
            'More alkaline')
        minor=none
        length=60
        width=3;
```

Define characteristics for vertical axis. LOGBASE=10 scales the vertical axis logarithmically, using a base of 10. Each major tick mark represents a power of 10. LOGSTYLE=EXPAND displays minor tick marks in logarithmic progression. LABEL= uses the ANGLE= suboption to place the label parallel to the vertical axis. VALUE= displays the major tick mark values as 10 plus an exponent. The HEIGHT= suboption

for each TICK= specification affects only the text following it. Units of CM override the default PCT specified by GUNIT=.

```
axis2 logbase=10
      logstyle=expand
      color=red
      label=(angle=90 h=4 color=black
            'Concentration (Moles/Liter)' )
      value=(tick=1 '10' height=1.5 '-14'
            tick=2 '10' height=1.5 '-13'
            tick=3 '10' height=1.5 '-12'
            tick=4 '10' height=1.5 '-11'
            tick=5 '10' height=1.5 '-10'
            tick=6 '10' height=1.5 '-9'
            tick=7 '10' height=1.5 '-8'
            tick=8 '10' height=1.5 '-7'
            tick=9 '10' height=1.5 '-6'
            tick=10 '10' height=1.5 '-5'
            tick=11 '10' height=1.5 '-4'
            tick=12 '10' height=1.5 '-3'
            tick=13 '10' height=1.5 '-2'
            tick=14 '10' height=1.5 '-1')
      major=(height=.75 cm)
      minor=(color=blue height=.5 cm);
```

Generate the plot and assign AXIS definitions. AXIS1 modifies the horizontal axis and AXIS2 modifies the vertical axis.

```
proc gplot data=concentr;
      plot conc*ph / haxis=axis1
                    vaxis=axis2;
run;
quit;
```

Example 3. Rotating Plot Symbols through the Colors List

Features:

GOPTIONS statement options:

COLORS=

HSIZE=

VSIZE=

LEGEND statement options:

LABEL=

SHAPE=

SYMBOL statement options:

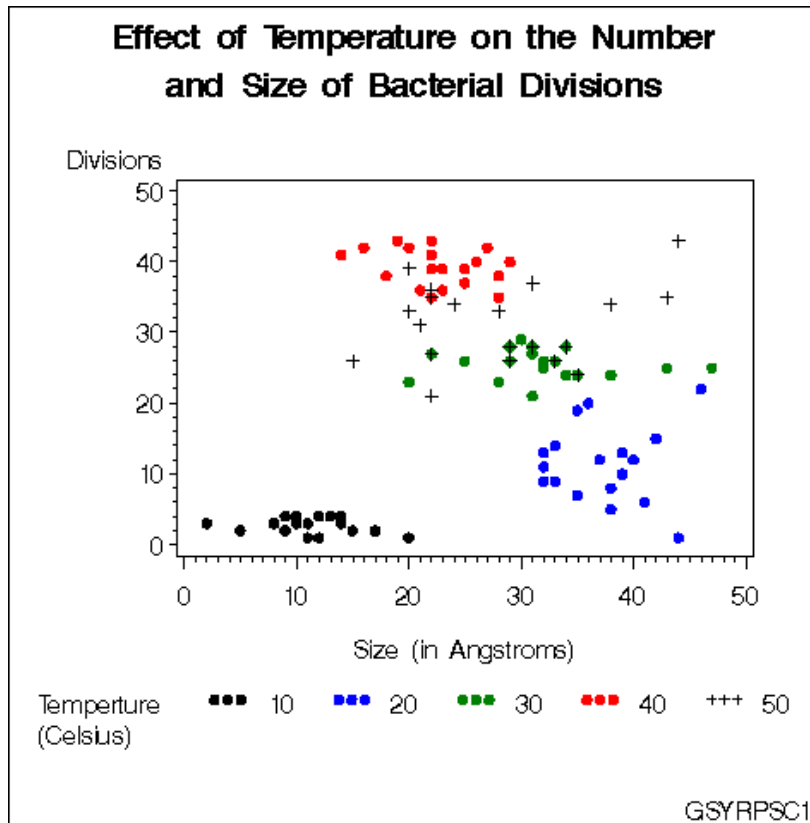
HEIGHT=

VALUE=

TITLE statement option:

JUSTIFY=

Sample library member: GSYRPSC1



This example specifies a plot symbol on a `SYMBOL` statement and rotates the symbol through the colors list. Temperature values in the data are represented by the same plot symbol in a different color. The example also shows how default symbol sequencing provides a default plot symbol if a plot needs more plot symbols than are defined.

The example uses the `GOPTIONS` statement to specify the colors for the color rotation. It also uses a `LEGEND` statement to specify a two-line legend label, and to align the label with the legend values.

Set the graphics environment. `COLORS=` specifies the colors list. This list is used by the `SYMBOL` statement. `HSIZE=` and `VSIZE=` specify the external dimensions of the graph. Units of `IN` override the default `PCT` specified by `GUNIT=`.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swissb htitle=4 htext=3
        hsize=5in vsize=5in;
```

Create the data set. `BACTERIA` contains information about the number and size of bacterial divisions at various temperatures.

```
data bacteria;
  input temp div mass life @@;
  datalines;
10 3 10 1 20 22 46 0 30 23 20 9 40 42 16 16 50 33 20 6
10 1 11 2 20 01 44 2 30 21 31 10 40 41 14 12 50 31 21 7
10 4 14 3 20 13 32 4 30 24 34 9 40 43 22 14 50 34 24 2
...more data lines...
10 3 02 2 20 09 32 5 30 26 32 9 40 39 22 15 50 36 22 5
10 2 05 3 20 07 35 4 30 24 35 15 40 37 25 14 50 24 35 4
10 3 08 1 20 05 38 6 30 23 28 9 40 35 28 16 50 33 28 6
```

```

;

proc sort data=bacteria;
  by temp;
run;

```

Define title and footnote. JUSTIFY= breaks the title into two lines.

```

title1 'Effect of Temperature on the Number'
      justify=center 'and Size of Bacterial
                    Divisions';
footnote1 h=3 j=r 'GSYRPSCL';

```

Define symbol characteristics. HEIGHT= specifies a height for the plot symbols. VALUE= specifies a dot for the plot symbol. Because no color is specified, the symbol is rotated through the colors list. Because the plot needs a fifth symbol, the default plus sign is rotated into the colors list to provide that symbol.

```

symbol1 height=2
        value=dot;

```

Define axis characteristics.

```

axis1 label=('Size (in Angstroms)') length=70;
axis2 label=('Divisions');

```

Define legend characteristics. LABEL= specifies text for the legend label. J=L specifies a new line and left-justifies the second string under the first. POSITION= aligns the top label line with the first (and in this case only) value row. SHAPE= specifies a width and height for legend values.

```

legend1 label=(position=(top left)
              'Temperature' j=1 '(Celsius)')
        shape=symbol(4,2);

```

Generate the plot.

```

proc gplot data=bacteria;
  plot div*mass=temp / frame
                        haxis=axis1
                        vaxis=axis2
                        legend=legend1;
run;
quit;

```

Example 4. Creating and Modifying Box Plots

Features:

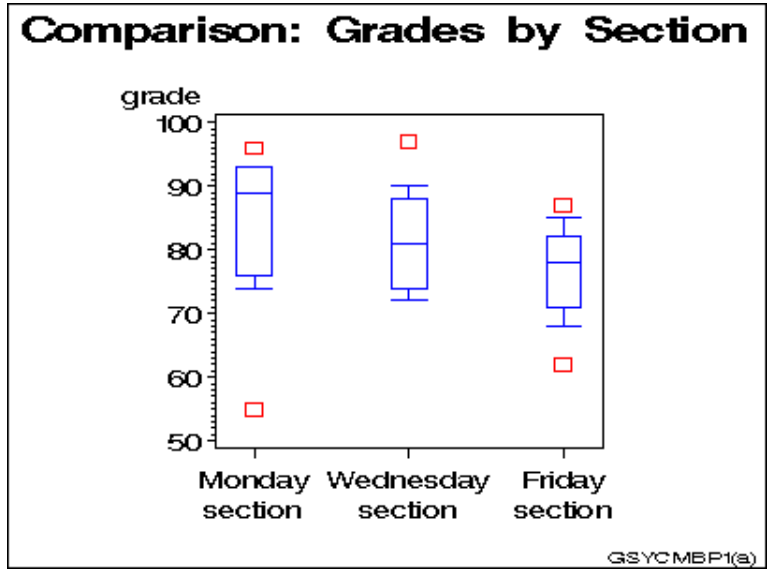
SYMBOL statement options:

```

BWIDTH=
CO=
CV=
HEIGHT=
INTERPOL=
VALUE=

```

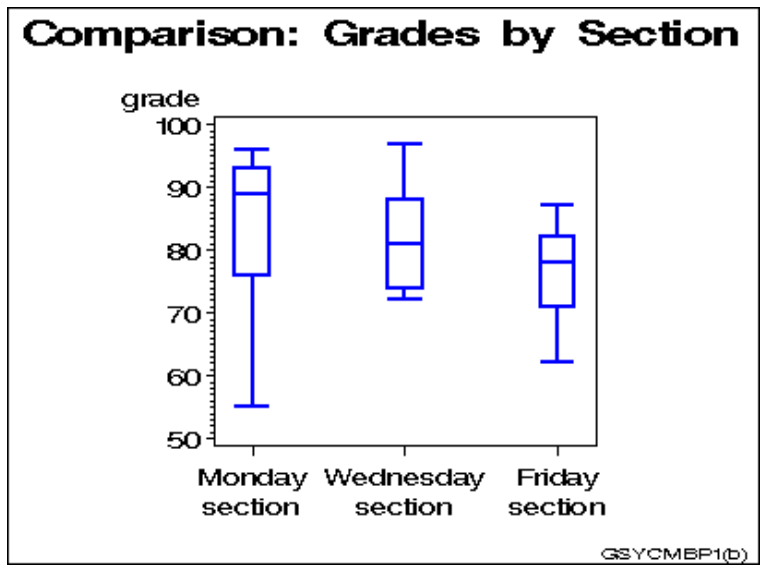
Sample library member: GSYCMBP1



This example shows how to create box plots and how to specify SYMBOL definitions so data outside the box-plot range can be represented with data points. It also shows how to change a box plot's percentile range to see if the new range encompasses the data.

The first plot in the example uses a SYMBOL definition with INTERPOL=BOXT20 to specify a box plot with whisker tops at the 80th percentile and whisker bottoms at the 20th percentile. Data points that are outside this percentile range are represented with squares.

As illustrated in the following output, the example then changes the SYMBOL definition to INTERPOL=BOXT10, which expands the whisker range to the 90th percentile for tops and the 10th percentile for bottoms. There are no data points outside the new percentile range.



Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftitle=swissb ftext=swiss htitle=6
        htext=4;
```

Create the data set. GRADES contains codes to identify each class section, and the grades scored by students in each section.

```
data grades;
    input section $ grade @@;
    datalines;
A 74 A 89 A 91 A 76 A 87 A 93 A 93 A 96 A 55
B 72 B 72 B 84 B 81 B 97 B 78 B 88 B 90 B 74
C 62 C 74 C 71 C 87 C 68 C 78 C 80 C 85 C 82
;
```

Define title and footnote.

```
title1 'Comparison of Grades by Section';
footnote1 j=r h=3 'GSYCMBP1(a) ';
```

Define symbol characteristics. INTERPOL=BOXT20 specifies a box plot with tops and bottoms on its whiskers, and the high and low bounds at the 80th and 20th percentiles. CO= colors the boxes and whiskers. BWIDTH= affects the width of the boxes. VALUE= specifies the plot symbol that marks the data points outside the range of the box plot. CV= colors the plot symbols. HEIGHT= specifies a symbol size.

```
symbol interpol=boxt20
        co=blue
        bwidth=6
        value=square
        cv=red
        height=4;
```

Define axis characteristics.

```
axis1 label=none
      value=(t=1 'Monday' j=c 'section'
            t=2 'Wednesday' j=c 'section'
            t=3 'Friday' j=c 'section')
      offset=(5,5)
      length=50;
```

Generate the first plot.

```
proc gplot data=grades;
    plot grade*section / haxis=axis1
                        vaxis=50 to 100 by 10;
run;
```

Change the footnote.

```
footnote j=r h=3 'GSYCMBP1(b) ';
```

Change symbol characteristics. INTERPOL=BOXT10 changes the high and low bounds to the 90th percentile at the top and the 10th percentile on the bottom. All other symbol characteristics remain unchanged.

```
symbol interpol=boxt10 width=2;
```

Generate the second plot.

```
plot grade*section / haxis=axis1
                                vaxis=50 to 100 by 10;
run;
quit;
```

Example 5. Filling the Area between Plot Lines

Features:

AXIS statement option:

ORDER=

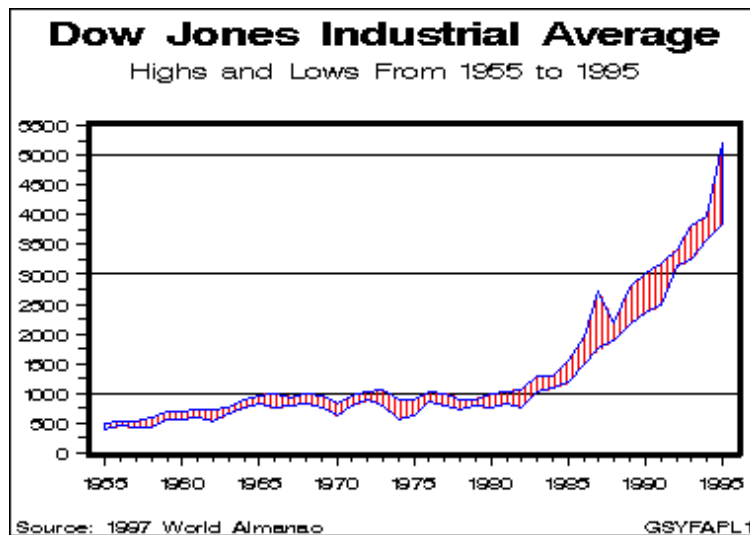
SYMBOL statement options:

CO=

CV=

INTERPOL=

Sample library member: GSYFAPL1



This example shows how to fill the area between two plot lines by concatenating two data sets into one to form a polygon with the data points. It uses a SYMBOL statement to specify a pattern to fill the polygon and to determine the color of the area fill and the outline around the area.

The example plots yearly highs and lows for the Dow Jones Industrial Average. It separates the dependent variables HIGH and LOW to produce an upper plot line and a lower plot line. The dependent variable is named VALUE and the independent variable is named YEAR. When concatenated into one data set, AREA, the data sets form the polygon.

Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swissb htitle=6 htext=3;
```

Create the data set. STOCKS contains yearly highs and lows for the Dow Jones Industrial Average, and the dates of the high and low values each year.

```
data stocks;
  input year @7 hdate date9. @17 high
        @26 ldate date9. @36 low;
  format hdate ldate date9.;
  datalines;
1955 30DEC1955 488.40 17JAN1955 388.20
1956 06APR1956 521.05 23JAN1956 462.35
...more data lines...
1994 31JAN1994 3978.36 04APR1994 3593.35
1995 13DEC1995 5216.47 30JAN1995 3832.08
;
```

Restructure the data so that it defines a closed area. Create the temporary data sets HIGH and LOW.

```
data high(keep=year value)
  low(keep=year value);
  set stocks;
  value=high; output high;
  value=low; output low;
run;
```

Reverse order of the observations in LOW.

```
proc sort data=low;
  by descending year;
```

Concatenate HIGH and LOW to create data set AREA.

```
data area;
  set high low;
```

Define titles and footnote.

```
title1 'Dow Jones Industrial Average';
title2 h=4 'Highs and Lows From 1955 to 1995';
footnote j=1 ' Source: 1997 World Almanac'
        j=r 'GSYFAPL1 ';
```

Define symbol characteristics. INTERPOL= specifies a map/plot pattern to fill the polygon formed by the data points. The pattern consists of medium-density parallel lines at 90 degrees. CV= colors the pattern fill. CO= colors the outline of the area. (If CO= were not used, the outline would be the color of the area.)

```
symbol interpol=m3n90
  cv=red
  co=blue;
```

Define axis characteristics. ORDER= places the major tick marks at 5-year intervals.

```
axis1 order=(1955 to 1995 by 5)
  label=none
  major=(height=2)
  minor=(number=4 height=1)
  offset=(2,2)
  width=3;
axis2 order=(0 to 5500 by 500)
  label=none
```

```
major=(height=1.5) offset=(0,0)
minor=(number=1 height=1);
```

Generate the plot using data set AREA.

```
proc gplot data=area;
  plot value*year / haxis=axis1
                    vaxis=axis2
                    vref=(1000 3000 5000);
run;
quit;
```

Example 6. Enhancing Titles

Features:

GOPTIONS statement options:

```
FTITLE=
FTEXT=
GUNIT=
HTITLE=
HTEXT=
```

TITLE statement options:

```
ANGLE=
BCOLOR=
BLANK=
BOX=
COLOR=
FONT=
HEIGHT=
MOVE=
ROTATE=
UNDERLIN=
```

Sample library member: GTIENTI1



This example illustrates some the ways you can format title text. The same options can be used to format footnotes. The GOPTIONS statement in the example determines the font and heights used for the first title line and all remaining text in the display. GOPTIONS also determines that percentages of the graphics output area are used as the unit of measure for heights in the graph.

Assign the libref and set the graphics environment. FTITLE= assigns the font that is used by the TITLE1 statement. FTEXT= assigns the font for all other text. HTITLE= makes the height of TITLE1 7 percent of the graphics output area, the units defined by the GUNIT= option. HTEXT= makes the height of all other text 5 percent of the graphics output area.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftitle=zapfb ftext=swissb htext=5;
```

Define title1. TITLE1 uses the default font and height defined in the GOPTIONS statement.

```
title1 'This is TITLE1';
```

Define TITLE3. Because TITLE2 is not assigned, the output displays a blank line. UNDERLIN= underlines both text strings.

```
title3 underlin=1
        'TITLE3 Is'
        color=red
        ' Underlined';
```

Define TITLE5. ANGLE= tilts the line of text clockwise 90 degrees and places it at the right edge of the output.

```
title5 color=red
        angle=-90
        'TITLE5 is Angled -90';
```

Define TITLE7. ROTATE= rotates each character in the text string at the specified angle. HEIGHT= overrides HTEXT= in the GOPTIONS statement.

```
title7 height=4
        color=red
        rotate=25
        'TITLE7 is Rotated';
```

Define TITLE8. BOX= draws a green box around the text.

```
title8 color=green
        box=1
        'TITLE8 is Boxed';
```

Define TITLE9. BLANK= prevents the boxed title from being overwritten by TITLE10. The first COLOR= specifies the color of the box border, and BCOLOR= specifies the color of the box background. The second COLOR= specifies the text color.

```
title9 color=red
        box=3
        blank=yes
        bcolor=red
        color=blue
        angle=-25
        'TITLE9 is Angled in a Red Box';
```

Define TITLE10. In this statement, BOX= draws a box around the first text string. BOX= is turned off by the MOVE= that uses absolute coordinates and causes a text break.

```
title10 color=red
        box=1
        bcolor=blue
        move=(5,20)
        font=script
        'TITLE10 is in Script and '
        move=(10,12)
        height=3
        'is Partially Boxed, Positioned with
        Explicit Moves,'
        move=(15,8)
        'and Overlaid by TITLE9';
```

Define footnote.

```
footnote h=3 justify=right 'GTIENTI1 ';
```

Display titles and footnote. All existing titles and footnotes are automatically displayed by the procedure.

```
proc gslide;
run;
quit;
```

Example 7. Using BY-group Processing to Generate a Series of Charts

Features:

AXIS statement options:

```
LABEL=
MAJOR=
MINOR=
NOPLANE
ORDER=
STYLE=
VALUE=
```

BY statement

GOPTIONS statement options:

```
H SIZE=
V SIZE=
```

OPTIONS statement option:

```
NOBYLINE
```

PATTERN statement option:

```
COLOR=
```

TITLE statement:

```
#BYVAL
```

Sample library member: GBYGMSC1

This example uses a BY statement with the GCHART procedure to produce a separate 3D vertical bar chart for each value of the BY variable TYPE. The three charts, which are shown in Display 7.1 on page 243, Display 7.2 on page 244, and Display 7.3 on page 245 following the code, show leading grain producers for 1995 and 1996.

The program suppresses the default BY lines and instead uses #BYVAL in the TITLE statement text string to include the BY variable value in the title for each chart.

The AXIS1 statement that is assigned to the vertical (response) axis is automatically applied to all three graphs generated by the BY statement. This AXIS statement removes all the elements of the response axis except the label. The same AXIS statement also includes an ORDER= option. Because this option is applied to all the graphs, it ensures that they all use the same scale of response values.

Because no subgroups are specified and the PATTERNID= option is omitted, the color specified in the single PATTERN statement is used by all the bars.

Set the graphics environment. HSIZE= and VSIZE= set the horizontal and vertical size of the graphics output area.

```
options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftitle=swissb ftext=swiss htitle=5
        htext=4 hsize=5in vsize=5in;
```

Create the data set GRAINLDR. GRAINLDR contains data about grain production in five countries for 1995 and 1996. The quantities in AMOUNT are in thousands of metric tons. MEGTONS converts these quantities to millions of metric tons.

```
data grainldr;
  length country $ 3 type $ 5;
  input year country $ type $ amount;
  megtons=amount/1000;
  datalines;
1995 BRZ Wheat 1516
1995 BRZ Rice 11236
1995 BRZ Corn 36276
...more data lines...
1996 USA Wheat 62099
1996 USA Rice 7771
1996 USA Corn 236064
;
```

Create a format for the values of COUNTRY.

```
proc format;
  value $country 'BRZ' = 'Brazil'
                'CHN' = 'China'
                'IND' = 'India'
                'INS' = 'Indonesia'
                'USA' = 'United States';
run;
```

Suppress the default BY-line and define a title that includes the BY-value. #BYVAL inserts the value of the BY variable COUNTRY into the title of each report.

```
options nobyline;
title1 'Leading #byval(type) Producers'
      j=c '1995 and 1996';
footnote1 j=r h=3 'GBYGMSC1 ';
```

Specify a color for the bars.

```
pattern1 color=green;
```

Define the axis characteristics for the response axes. ORDER= specifies the range of values for the response axes. ANGLE=90 in the LABEL= option rotates the label 90 degrees. All the other options remove axis elements. MAJOR=, MINOR=, and VALUE= remove the tick marks and values. STYLE=0 removes the line. NOPLANE removes the 3D plane.

```
axis1 order=(0 to 550 by 100)
      label=(angle=90 'Millions of Metric Tons')
      major=none
      minor=none
      value=none
      style=0
      noplane;
```

Define midpoint axis characteristics. SPLIT= defines the character that causes an automatic line break in the axis values.

```
axis2 label=none
      split=' ';
```

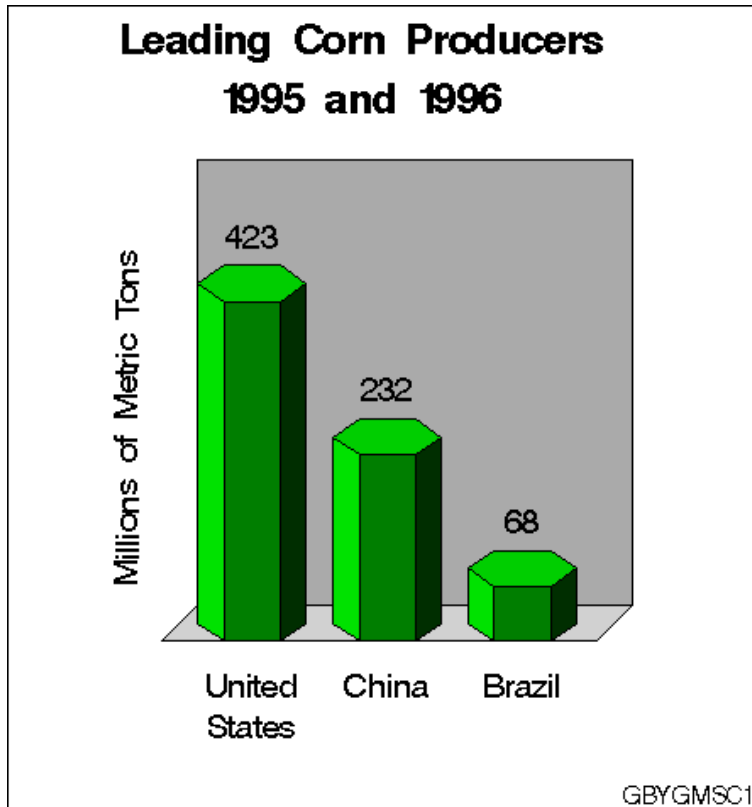
Sort data according to values of BY variable. The data must be sorted before running PROC GCHART with the BY statement.

```
proc sort data=grainldr out=temp;
      by type;
run;
```

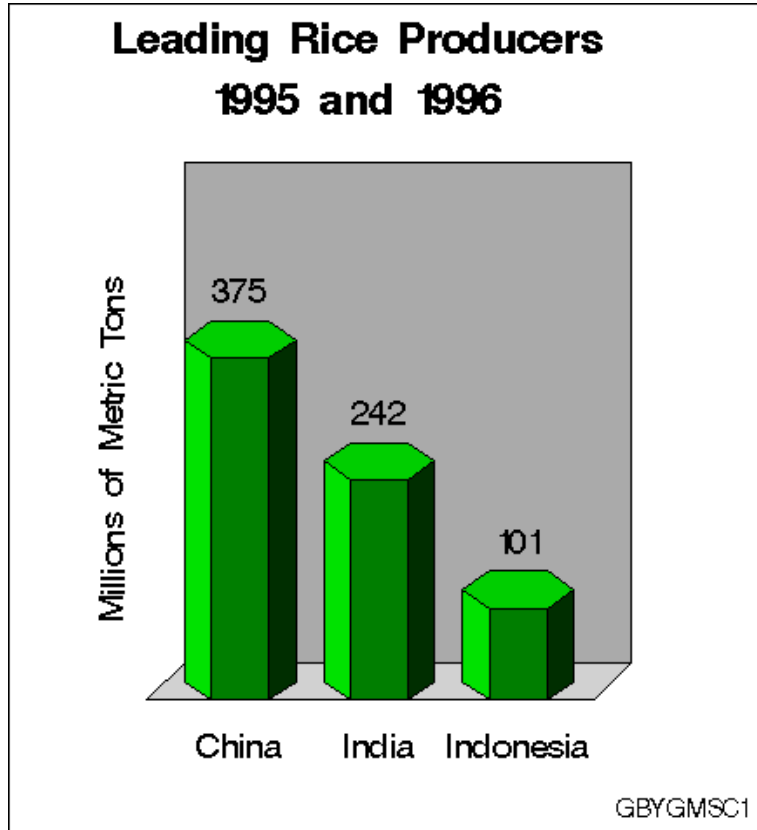
Generate the vertical bar charts using a BY statement. The BY statement produces a chart for each value of SITE. The FORMAT statement assigns the \$COUNTRY. format to the chart variable. Assigning AXIS1 to RAXIS= causes all three charts to have the same response axis.

```
proc gchart data=temp (where=(megtons gt 31))
      by type;
      format country $country.;
      vbar3d country / sumvar=megtons
              outside=sum
              descending
              shape=hexagon
              width=8
              coutline=black
              cframe=grayaa
              maxis=axis2
              raxis=axis1 name='GBYGMSC1';
run;
quit;
```

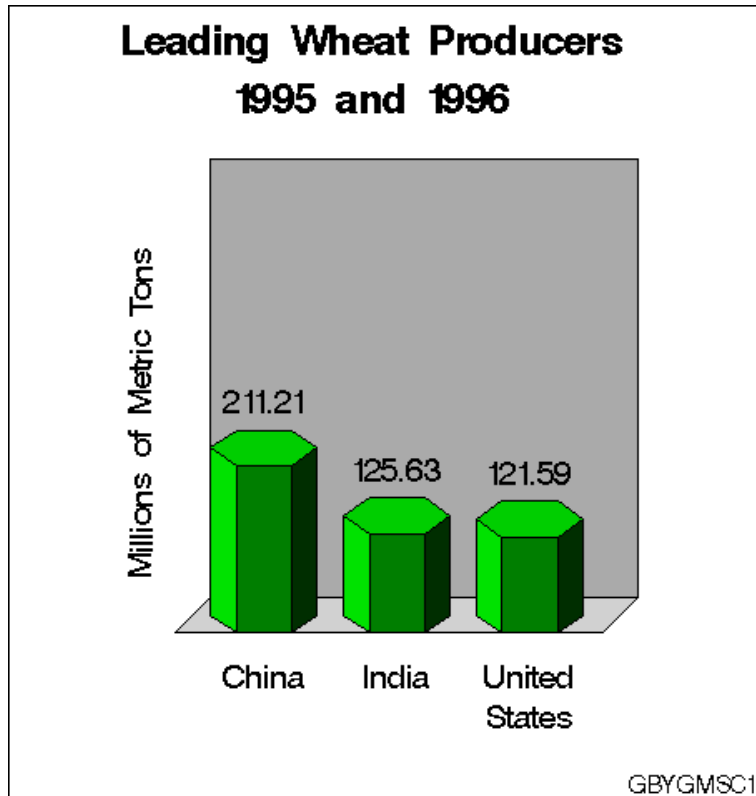

Display 7.1 Output for BY Value Corn



Display 7.2 Output for BY Value Rice



Display 7.3 Output for BY Value Wheat



Example 8. Creating a Simple Web Page with the ODS HTML Statement

Features:

ODS HTML statement options:

BODY=

PATH=

CLOSE

GOPTIONS statement options:

COLORS=

DEVICE=

TRANSPARENCY

NOBORDER (default)

LEGEND statement options:

ACROSS=

LABEL=

MODE=

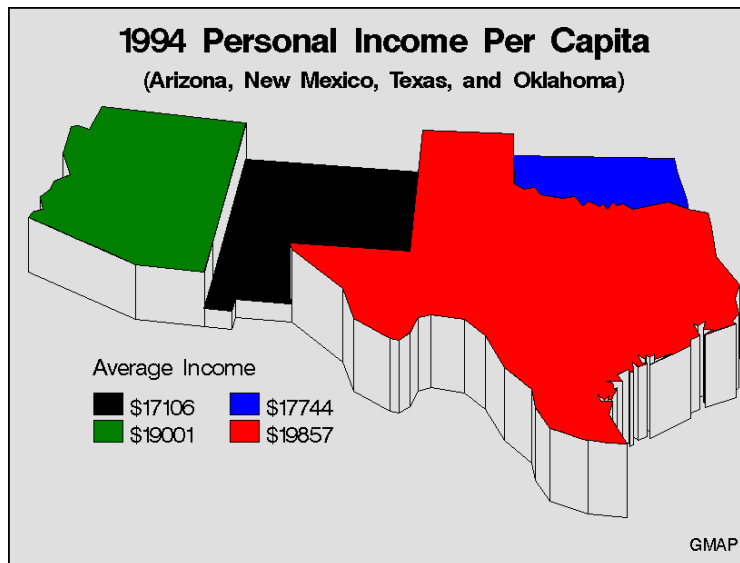
ORIGIN=

SHAPE=

WHERE statement

Sample library member: GONCSWB1

Display 7.4 Displaying a Map in a Web Page



This example illustrates the simplest way to use the ODS HTML statement to create an HTML file and a GIF file that you can display in a Web browser. It generates one body file that displays one piece of SAS/GRAPH output – a map of average per capita income for four states.

This example also illustrates default pattern behavior with maps and explicit placement of the legend on the graph. It shows how the default solid map pattern rotates through every color in a colors list defined in the GOPTIONS statement. By default, the outline color is the first color in the list, in this case, BLACK.

And it shows how to use a LEGEND statement to arrange and position a legend so it fits well with the graph's layout.

Assign the Web-server path. FILENAME assigns the fileref ODSOUT, which is a destination for the HTML and GIF files produced by the example program. To assign that location as the HTML destination for program output, ODSOUT is specified later in the program on the ODS HTML statement's PATH= option. ODSOUT must point to a Web-server location if procedure output is to be viewed on the Web.

```
filename odsout 'path-to-Web-server-space';
```

Close the ODS Listing destination for procedure output, and set the graphics environment. To conserve system resources, ODS LISTING CLOSE closes the Listing destination for procedure output. Thus, the graphics output is not displayed in the GRAPH window, although it is written to the graphics catalog and to the GIF files. COLORS= on the GOPTIONS statement defines a list of four colors for the graph.

```
ods listing close;
goptions reset=global gunit=pct cback=white
         colors=(black blue green red)
         ftext=swiss ftitle=swissb htitle=6 htext=4;
```

Create the data set INCOME. INCOME contains state codes for four states and the average income of each state.

```

data income;
  input state income;
  datalines;
04    19001
35    17106
40    17744
48    19857
;

```

Assign graphics options for producing the ODS HTML output. DEVICE=GIF causes the ODS HTML statement to generate the graphics output as GIF files. TRANSPARENCY causes the graphics output to use the Web-page background as the background of the graph. Because the default setting NOBORDER is used, the edge of the graph is not visible on the Web page.

```
goptions device=gif transparency noborder;
```

Open the ODS HTML destination. BODY= names the file for storing HTML output. PATH= specifies the ODSOUT fileref as the destination for all the HTML and GIF files.

```
ods html body='income_body.html'
      path=odsout;
```

Define titles and a footnote for the map. By default, any defined titles and footnotes are included in the graphics output (GIF file).

```
title '1994 Personal Income Per Capita';
title2 f=swissb '(Arizona, New Mexico, Texas, and Oklahoma)';
footnotel h=3 j=r 'GMAP ';
```

Define legend characteristics. ACROSS= defines the number of columns in the legend. ORIGIN= specifies an exact location for the legend. MODE= allows the legend to share the output area. LABEL= specifies a legend label and left-justifies it above the legend values. SHAPE= specifies a size and shape for the legend values.

```
legend across=2
      origin=(10,20)
      mode=share
      label=(position=top
            justify=left
            'Average Income')
      shape=bar(4,4);
```

Generate the prism map. Because the NAME= option is omitted, SAS/GRAPH assigns the default name GMAP to the GRSEG entry in the graphics catalog. This is the name that is assigned to the GIF file created by the ODS HTML statement.

```
proc gmap map=maps.us data=income;
  format income dollar6.0;
  id state;
  prism income / discrete
          legend=legend;
run;
quit;
```

Close the ODS HTML destination, and open the ODS Listing destination. You must close the HTML destination before you can view the output with a browser. ODS LISTING opens the Listing destination so that the destination is again available for displaying output during this SAS session.

```
ods html close;
ods listing;
```

Example 9. Combining Graphs and Reports in a Web Page

Features:

AXIS statement options:

```
LENGTH=
VALUE=
```

BY statement

GOPTIONS statement options:

```
BORDER
DEVICE=
TRANSPARENCY
```

ODS HTML statement options:

```
BODY=
CONTENTS=
FRAME=
PATH=
NOGTITLE
```

OPTIONS statement option:

```
NOBYLINE
```

TITLE statement option:

```
#BYVAL
```

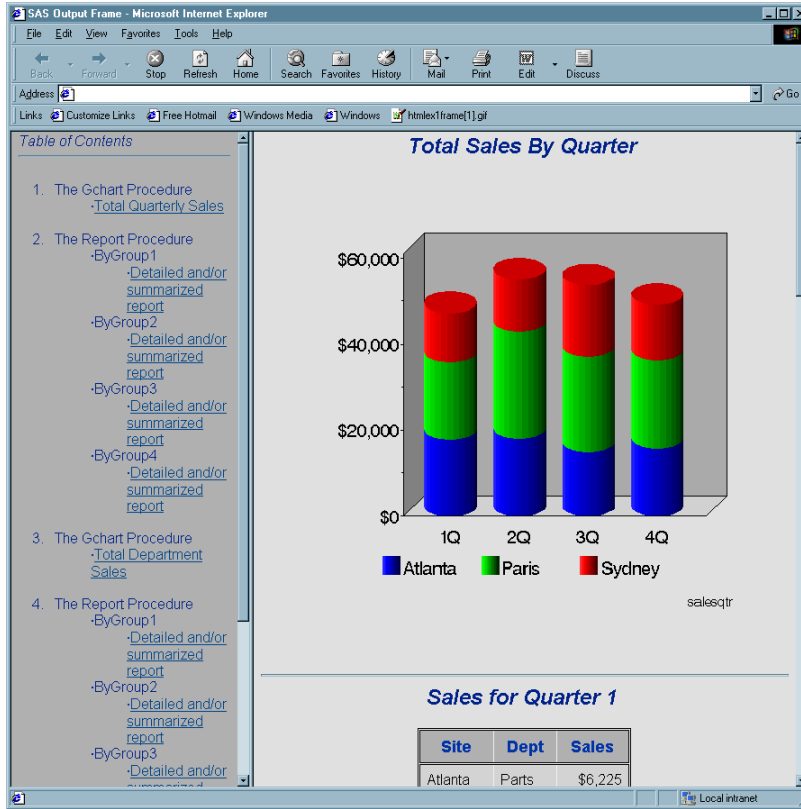
Sample library member: GONCGRW1

This example generates several graphs of sales data that can be accessed from a single Web page. The graphs are two bar charts of summary sales data and three pie charts that break the data down by site. Each bar chart and an accompanying report is stored in a separate body file.

The three pie charts are generated with BY-group processing and are stored in one body file. The program suppresses the default BY lines and instead includes the BY variable value in the title for each chart. The SAS/GRAPH titles are displayed in the HTML output instead of in the graphics output.

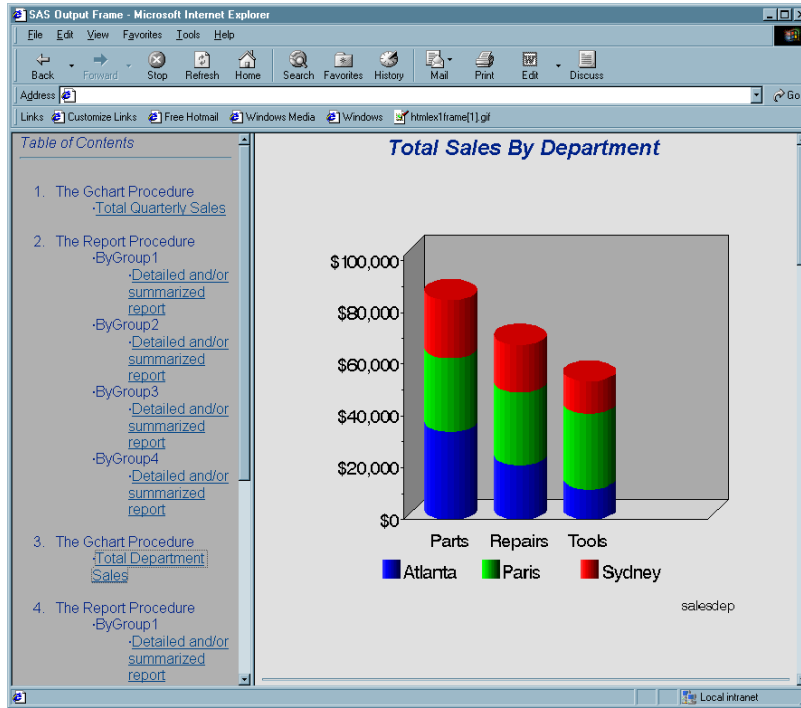
The Web page contains two frames, one that displays a Table of Contents for all the graphs, and one that serves as the display area. Links to each piece of output appear in the table of contents, which is displayed in the left frame. Initially the frame file displays the first body file, which contains a bar chart and a report, as shown in the following figure.

Display 7.5 Browser View of Bar Chart and Quarterly Sales Report



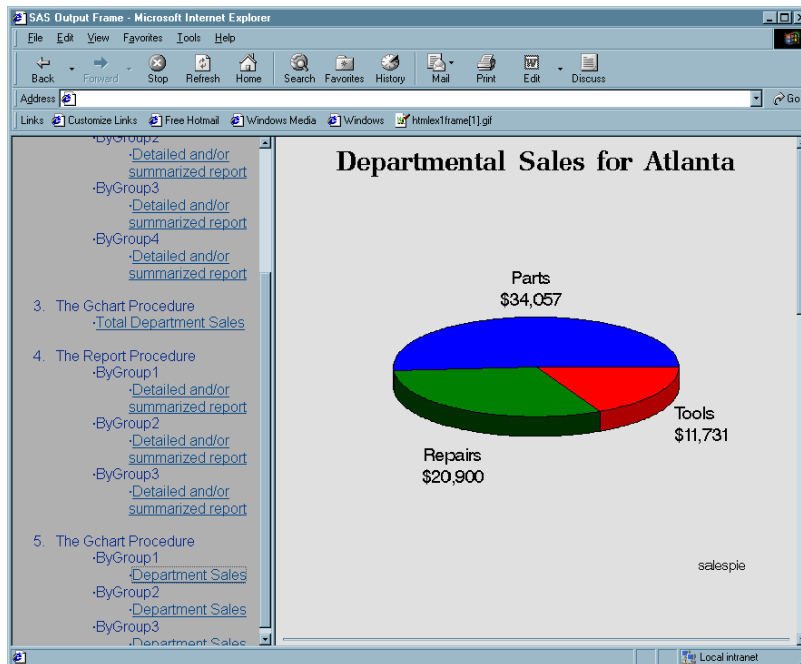
Notice that the chart title is displayed outside the graph as part of the HTML file. Select the link to *Total Department Sales* to display the second bar chart, as shown in the following figure.

Display 7.6 Browser View of Bar Chart and Department Sales Report



Selecting any link for *Department Sales* displays the corresponding pie chart as shown in the following figure.

Display 7.7 Browser View of Pie Charts of Site Sales



Because the pie charts are stored in one file, you can easily see all three by scrolling through the file.

Additional features include AXIS statements that specify the same length for both midpoint axes, so that the bar charts are the same width even though they have a different number of bars.

Assign the Web-server path. FILENAME assigns the fileref ODSOUT, which specifies a destination for the HTML and GIF files produced by the example program. To assign that location as the HTML destination for program output, ODSOUT is specified later in the program on the ODS HTML statement's PATH= option. ODSOUT must point to a Web-server location if procedure output is to be viewed on the Web.

```
filename odsout 'path-to-Web-server-space';
```

Close the ODS Listing destination for procedure output, and set the graphics environment. To conserve system resources, ODS LISTING CLOSE closes the Listing destination for procedure output. On the GOPTIONS statement, HSIZE= and VSIZE= set the horizontal and vertical size of the graphics output area. DEVICE=GIF causes the ODS HTML statement to generate the graphics output as GIF files. TRANSPARENCY causes the graphics output to use the Web-page background as the background of the graph. BORDER is used so that the border around the graphics output area will be compatible with the borders that are created for nongraphics output.

```
ods listing close;
goptions reset=global gunit=pct border
         colors=(blue green red) ctext=black
         hsize=5in vsize=5in ftitle=zapfb
         ftext=swiss htitle=6 htext=4
         device=gif transparency;
```

Create the data set TOTALS. The data set contains quarterly sales data for three manufacturing sites for one year.

```
data totals;
  length dept $ 7 site $ 8;
  input dept site quarter sales;
  datalines;
Parts   Sydney  1 4043.97
Parts   Atlanta 1 6225.26
Parts   Paris   1 3543.97
Repairs Sydney  1 5592.82
Repairs Atlanta 1 9210.21
Repairs Paris   1 8591.98
Tools   Sydney  1 1775.74
Tools   Atlanta 1 2424.19
Tools   Paris   1 5914.25
Parts   Sydney  2 3723.44
Parts   Atlanta 2 11595.07
Parts   Paris   2 9558.29
Repairs Sydney  2 5505.31
Repairs Atlanta 2 4589.59
Repairs Paris   2 7538.56
Tools   Sydney  2 2945.17
Tools   Atlanta 2 1903.99
Tools   Paris   2 7868.34
Parts   Sydney  3 8437.96
Parts   Atlanta 3 6847.91
Parts   Paris   3 6789.85
Repairs Sydney  3 4426.46
Repairs Atlanta 3 5011.66
```

```
Repairs Paris 3 6510.38
Tools Sydney 3 3767.10
Tools Atlanta 3 3048.52
Tools Paris 3 9017.96
Parts Sydney 4 6065.57
Parts Atlanta 4 9388.51
Parts Paris 4 8509.08
Repairs Sydney 4 3012.99
Repairs Atlanta 4 2088.30
Repairs Paris 4 5530.37
Tools Sydney 4 3817.36
Tools Atlanta 4 4354.18
Tools Paris 4 6511.70
;
```

Open the ODS HTML destination. FRAME= names the HTML file that integrates the contents and body files. CONTENTS= names the HTML file that contains the table of contents to the HTML procedure output. BODY= names the file for storing the HTML output. The contents file links to each of the body files written to the HTML destination. PATH= specifies the ODSOUT fileref as the HTML destination for all the HTML and GIF files. NOGTITLE suppresses the graphics titles from the SAS/GRAPH output and displays them through the HTML page.

```
ods html frame='sales_frame.html'
      contents='sales_contents.html'
      body='sales_body1.html'
      path=odsout
      nogtitle;
```

Define title and footnote. TITLE1 uses the font and height specified by FTITLE= and HTITLE= in the GOPTIONS statement.

```
title1 'Total Sales By Quarter';
footnote j=r h=3 'salesqtr ';
```

Define axis characteristics for the first bar chart. In AXIS2, LENGTH= specifies the length of the midpoint axis.

```
axis1 order=(0 to 60000 by 20000)
      minor=(number=1)
      label=none;
axis2 label=none length=70pct
      value=('1Q' '2Q' '3Q' '4Q');
```

Suppress the legend label and define the size of the legend values.

```
legend1 label=none shape=bar(4,4);
```

Generate the vertical bar chart of quarterly sales. NAME= specifies the name of the catalog entry. Because the PATH= destination is a file storage location and not a specific file name, the name SALESQTR.GIF is assigned to the GIF file, matching the named assigned to the GRSEG on NAME=. DES= specifies the description that is stored in the graphics catalog and used in the Table of Contents.

```
proc gchart data=totals;
  format sales dollar8.;
  vbar3d quarter / discrete
        sumvar=sales
        shape=cylinder
```

```

        subgroup=site
        cframe=grayaa
        caxis=black
        width=12
        space=4
        legend=legend1
        maxis=axis2
        raxis=axis1
        des='Total Quarterly Sales'
        name='salesqtr';

run;
quit;

```

Sort the data set for the report of quarterly sales. The data must be sorted in order of the BY variable before running PROC REPORT with BY-group processing.

```

proc sort data=totals out=qtrsrt;
  by quarter site;
run;

```

Reset the footnote and suppress the BY-line. We suppress the by-line because otherwise #BYVAL inserts the value of the BY variable into the title of each report.

```

footnotel;
options nobyline;

```

Generate a report of quarterly sales. Because the HTML body file that references the GCHART procedure output is still open, the report is stored in that file. The chart and report are shown in Display 7.5 on page 249.

```

titlel 'Sales for Quarter #byval(quarter)';
proc report data=qtrsrt nowindows;
  by quarter;
  column quarter site dept sales;
  define quarter / noprint group;
  define site / display group;
  define dept / display group;
  define sales / display sum format=dollar8.;
  compute after quarter;
    site='Total';
  endcomp;
  break after site / summarize style=rowheader;
  break after quarter / summarize style=rowheader;
run;

```

Open a new body file for the second bar chart and report. Assigning a new body file closes SALES_BODY1.HTML. The contents and frame files, which remain open, will contain links to all body files.

```

ods html body='sales_body2.html' path=odsout;

```

Define title and footnote for second bar chart.

```

titlel 'Total Sales By Department';
footnotel j=r h=3 'salesdep ';

```

Define axis characteristics. These AXIS statements replace the ones defined earlier. As before, LENGTH= defines the length of the midpoint axis.

```
axis1 label=none
      minor=(number=1);
      order=(0 to 100000 by 20000)
axis2 label=none length=70pct;
```

Generate the vertical bar chart of departmental sales.

```
proc gchart data=totals;
  format sales dollar8.;
  vbar3d dept / shape=cylinder
             subgroup=site
             cframe=grayaa
             width=12
             space=4
             sumvar=sales
             legend=legend1
             maxis=axis2
             raxis=axis1
             caxis=black
             des='Total Department Sales'
             name='salesdep';

run;
quit;
```

Sort the data set for the report of department sales. The data must be sorted in order of the BY variable before running PROC REPORT with BY-group processing.

```
proc sort data=totals out=deptsort;
  by dept site;
run;
```

Reset the footnote, define a report title, and generate the report of department sales. #BYVAL inserts the value of the BY variable into the title of each report. The chart and report are shown in Display 7.5 on page 249.

```
footnote1;
title1 'Sales for #byval(dept)';
proc report data=deptsort nowindows;
  by dept;
  column dept site quarter sales;
  define dept / noprint group;
  define site / display group;
  define quarter / display group;
  define sales / display sum format=dollar8.;
  compute after dept;
    site='Total';
  endcomp;
  break after site / summarize style=rowheader;
  break after dept / summarize style=rowheader;
run;
```

Open a new body file for the pie charts. Assigning a new file as the body file closes SALES_BODY2.HTML. The contents and frame files remain open. GTITLE displays the titles in the graph.

```
ods html body='sales_body3.html' gtitle path=odsout;
```

Sort data set in order of the BY variable before running the GCHART procedure with BY-group processing.

```
proc sort data=totals out=sitesort;
  by site;
run;
```

Define title and footnote. #BYVAL inserts the value of the BY variable SITE into the title for each output.

```
title 'Departmental Sales for #byval(site)';
footnote j=r h=3 'salespie ';
```

Generate a pie chart for each site. All the procedure output is stored in one body file. Because BY-group processing generates multiple graphs from one PIE3D statement, the name assigned by NAME= is incremented to provide a unique name for each piece of output.

```
proc gchart data=sitesort;
  format sales dollar8.;
  by site;
  pie3d dept / noheading
           coutline=black
           sumvar=sales
           des='Department Sales'
           name='salespie';

run;
quit;
```

Close the ODS HTML destination, and open the ODS Listing destination.

```
ods html close;
ods listing;
```

Example 10. Creating a Bar Chart with Drill-down for the Web

Features:

GOPTIONS statement option:

RESET=

ODS HTML statement options:

BODY=

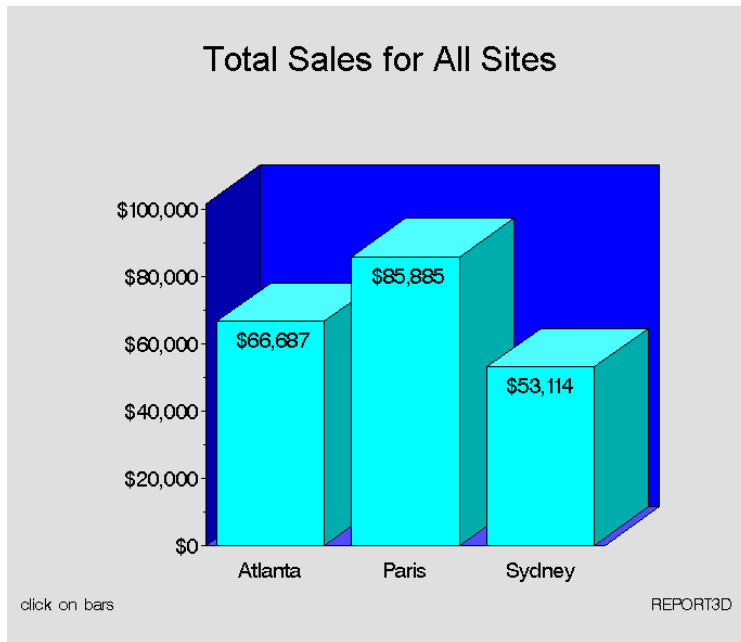
NOGTITLE

PATH=

Sample library member: GONDDCW1

This example shows you how to create a drill-down graph in which the user can select an area of the graph in order to display additional information about the data. The program creates one vertical bar chart of total sales for each site and three reports that break down the sales figures for each site by department and quarter. The following figure shows the bar chart of sales.

Display 7.8 Vertical Bar Chart of Total Sales



Display 7.9 on page 256 shows the PROC REPORT output that appears when you click on the bar for Atlanta.

Display 7.9 PROC REPORT Output Displayed in a Web Browser

Dept	Quarter	Sales
Parts	1	\$6,225
	2	\$11,595
	3	\$6,848
	4	\$9,389
Repairs	1	\$9,210
	2	\$4,590
	3	\$5,012
	4	\$2,088
Tools	1	\$2,424
	2	\$1,904
	3	\$3,049
	4	\$4,354
Total		\$66,687

For additional information about this program, see “Details” on page 259.

Assign the Web-server path. FILENAME assigns the fileref ODSOUT, which specifies a destination for the HTML and GIF files produced by the example program. To assign that location as the HTML destination for program output, ODSOUT is specified later

in the program on the ODS HTML statement's PATH= option. ODSOUT must point to a Web-server location if procedure output is to be viewed on the Web.

```
filename odsout 'path-to-Web-server-space';
```

Close the ODS Listing destination for procedure output, and set the graphics environment. To conserve system resources, ODS LISTING CLOSE closes the Listing destination for procedure output. On the GOPTIONS statement, DEVICE=GIF causes the ODS HTML statement to generate the graphics output as GIF files. TRANSPARENCY causes the graphics output to use the Web-page background as the background of the graph.

```
ods listing close;
goptions reset=global gunit=pct
        colors=(black blue green red)
        hsize=7 in vsize=5 in ftitle=zapfb
        ftext=swiss htitle=6 htext=4
        device=gif transparency noborder;
```

Add the HTML variable to TOTALS and create the NEWTOTAL data set. The HTML variable SITEDRILL contains the targets for the values of the variable SITE. Each HREF value specifies the HTML body file and the name of the anchor within the body file that identifies the target graph.

```
data newtotal;
  set totals;
  length sitedrill $40;
  if site='Atlanta' then
    sitedrill='HREF="report_deptsales.html#IDX1"';

  else if site='Paris' then
    sitedrill='HREF="report_deptsales.html#IDX2"';

  else if site='Sydney' then
    sitedrill='HREF="report_deptsales.html#IDX3"';
run;
```

Open the ODS HTML destination. BODY= names the file for storing HTML output. PATH= specifies the ODSOUT fileref as the HTML destination for all the HTML and GIF files. NOGTITLE suppresses the graph titles from the SAS/GRAPH output and displays them in the HTML.

```
ods html path=odsout
        body='report_body.html'
        nogtitle;
```

Define title and footnote.

```
title1 'Total Sales for All Sites';
footnote1 h=3 j=1 'click on bars' j=r 'REPORT3D ';
```

Assign a pattern color for the bars. Each bar in the graph uses the same PATTERN definition.

```
pattern color=cyan;
```

Define axis characteristics. The VBAR3D statement assigns AXIS1 to the response axis and AXIS2 to the midpoint axis.

```
axis1 order=(0 to 100000 by 20000)
      minor=(number=1)
```

```

label=none;
axis2 label=none offset=(9,9);

```

Generate the vertical bar chart of total sales for each site. HTML= specifies SITEDRILL as the variable that contains the name of the target. Specifying HTML= causes SAS/GRAPH to add an image map to the HTML body file. NAME= specifies the name of the catalog entry.

```

proc gchart data=newtotal;
  format sales dollar8.;
  vbar3d site / discrete
           width=15
           sumvar=sales
           inside=sum
           html=sitedrill
           coutline=black
           cframe=blue
           maxis=axis2
           raxis=axis1
           name='report3d ';
run;
quit;

```

Open the file for the PROC REPORT output. Assigning a new body file closes REPORT_BODY.HTML.

```
ods html body='report_deptsales.html' path=odsout;
```

Sort the data set NEWTOTAL. The data must be sorted in order of the BY variable before running PROC REPORT with BY-group processing.

```

proc sort data=newtotal;
  by site dept quarter;
run;
quit;

```

Clear the footnote.

```
goptions reset=footnotel;
```

Suppress the default BY-line and define a title that includes the BY-value. #BYVAL inserts the value of the BY variable SITE into the title of each report.

```

options nobyline;
title1 'Sales Report for #byval(site)';

```

Print a report of departmental sales for each site.

```

proc report data=newtotal nowindows;
  by site;
  column site dept quarter sales;
  define site / noprint group;
  define dept / display group;
  define quarter / display group;
  define sales / display sum format=dollar8.;
  compute after site;
    dept='Total';
  endcomp;
  break after site / summarize style=rowheader page;
run;

```



```
quit;
```

Close the ODS HTML destination, and open the ODS Listing destination.

```
ods html close;
ods listing;
```

Details

This section provides additional information about the pieces of this program and how they work together to generate SAS/GRAPH output with drill-down functionality. It describes

- how an HREF value is built
- how the HTML= option creates an image map in the HTML file
- how the HTML file references the SAS/GRAPH output.

Building an HREF value

In the DATA step, the variable SITEDRILL is assigned a string that defines the link target for a data value. For example,

```
if site='Atlanta' then
  sitedrill='HREF="report_deptsales.html#IDX1"';
```

The link target is specified by the HTML HREF attribute. The HREF value tells the Web page where to link to when a user selects the region associated with the value **Atlanta**.

For example, clicking on the first bar in the chart links to the target defined by **report_deptsales.html#IDX1**. This target is composed of a filename and an anchor. The file, **report_deptsales.html**, is generated by the PROC REPORT step. **IDX1** is the anchor that identifies the section of the file that contains the report for the first BY group, **Atlanta**.

Because anchor names increment, in order to assign them accurately you must know how many pieces of output your program generates and in what order. For example, this table lists in order the pieces of output generated by this example and their default anchor names:

Procedure	Output	Anchor name
GCHART	report3d.gif	IDX
REPORT	Atlanta report	IDX1
REPORT	Paris report	IDX2
REPORT	Sydney report	IDX3

Creating an image map

The HTML= option in the GCHART procedure is assigned the variable with the target information – in this case, SITEDRILL.

```
html=sitedrill
```

This option causes SAS/GRAPH to generate in the HTML body file the MAP and AREA elements that compose the image map. It loads the HREF attribute value from SITEDRILL into the AREA element. This image map, which is named **gqcke00k_map**, is stored in **report_body.html** (ODS generates unique map names each time you run the program, so the next time this program runs, the map name will be different):

```
<MAP NAME="gqcke00k_map">
  <AREA SHAPE="POLY"
    HREF="report_deptsales.html#IDX3"
    COORDS="423,409,423,242,510,242,510,409" >
  <AREA SHAPE="POLY"
    HREF="report_deptsales.html#IDX2"
    COORDS="314,409,314,139,401,139,401,409" >
  <AREA SHAPE="POLY"
    HREF="report_deptsales.html#IDX1"
    COORDS="205,409,205,199,292,199,292,409" >
</MAP>
```

The AREA element defines the regions within the graph that you can select to link to other locations. It includes attributes that define the shape of the region (SHAPE=) and position of the region (COORDS=) as well as the link target (HREF=).

The value assigned to the HREF= attribute is contained in the variable assigned to HTML=, in this case SITEDRILL.

Referencing SAS/GRAPH output

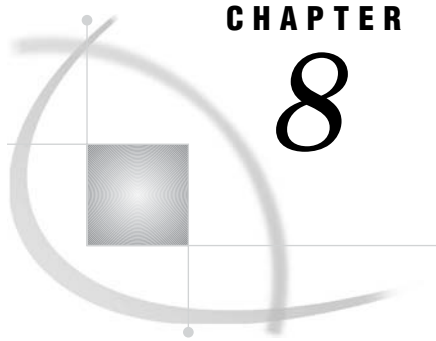
In the GOPTIONS statement, DEVICE=GIF causes SAS/GRAPH to create GIF files from the SAS/GRAPH output. It also adds to the open body file an IMG element that points to the GIF file. In this case, SAS/GRAPH adds the following IMG element to **report_body.html**:

```
<IMG SRC="report3d.gif" USEMAP="#gqcke00k_map">
```

The IMG element tells the Web page to get the image from the file **report3d.gif**. It also tells the Web page to use the image map **#report3d_map** to define the hot spots of the bar chart.

See Also

- For more information on the BY, LABEL, OPTIONS, and WHERE statements in base SAS software, see *SAS Language Reference: Dictionary*.



CHAPTER

8

Graphics Options and Device Parameters Dictionary

<i>Introduction</i>	261
<i>Specifying Graphics Options and Device Parameters</i>	261
<i>Specifying Units of Measurement</i>	262
<i>Dictionary of Graphics Options and Device Parameters</i>	262

Introduction

This chapter provides a detailed description of all of the graphics options and device parameters used with SAS/GRAPH software. These include

- all graphics options used by the GOPTIONS statement
- all device parameters that can be specified as options in the ADD and MODIFY statements in the GDEVICE procedure
- all device parameters that appear as fields in the GDEVICE windows.

The descriptions provide the syntax, defaults, and required information for each option and parameter.

The graphics options and device parameters are intermixed and listed alphabetically. When the graphics option and device parameter have the same name, they are discussed in the same dictionary entry and the description uses only that name and does not distinguish between the option and the parameter except where the distinction is necessary.

For a list of all the graphics options, see “GOPTIONS Statement” on page 146. For a list of all the device parameters, see “ADD Statement” on page 921.

If the syntax for the graphics option and the device parameter is different, both forms are shown. If the syntax is the same, one form is shown.

Specifying Graphics Options and Device Parameters

Use a GOPTIONS statement to specify the graphics options. Some graphics options can also be specified in an OPTIONS statement. Use the GDEVICE procedure to specify the device parameters. (See “GOPTIONS Statement” on page 146 and Chapter 31, “The GDEVICE Procedure,” on page 915 for details.)

Note: The syntax for device parameters is the syntax for specifying parameters when using the GDEVICE procedure statements. With the GDEVICE windows, simply enter values into fields in the windows. △

Note: The values that you specify for any option or parameter must be valid for the device. If you specify a value that exceeds the device's capabilities, SAS/GRAPH software reverts to values that can be used with the device. \triangle

Specifying Units of Measurement

When the syntax of an option includes *units*, use one of these unless the syntax specifies otherwise:

CELLS	character cells
CM	centimeters
IN	inches
PCT	percentage of the graphics output area
PT	points (there are approximately 72 points in an inch).

If you omit *units*, a unit specification is searched for in this order:

- 1 the value of GUNIT= in a GOPTIONS statement
- 2 the default unit, CELLS.

Dictionary of Graphics Options and Device Parameters

ADMGDF

Specifies whether to write an ADMGDF or GDF file when the GSFNAME= and GSFMODE= graphics options are used with a GDDM device driver.

Used in: GOPTIONS statement

Default: NOADMGDF

Restriction: GDDM device drivers on IBM mainframe systems only

Syntax

ADMGDF | NOADMGDF

ADMGDF

instructs the GDDM device driver to write out an ADMGDF file.

NOADMGDF

instructs the GDDM device driver to write out a GDF file.

ASPECT

Sets the aspect ratio for graphics elements.

Used in: GOPTIONS statement GDEVICE procedure GDEVICE Detail window

Default: device dependent

Restriction: not supported by Java or ActiveX

Syntax

ASPECT=*scaling-factor*

scaling-factor

is a nonnegative integer or real number that determines the ratio of width to height for graphics elements. If you specify ASPECT=1, each graphics element has equal horizontal and vertical scaling factors; ASPECT=2 scales the graphics element twice as wide as its height; and so on. If ASPECT= is not specified or is set to 0 or null, SAS/GRAPH uses the aspect ratio of the hardware device.

Details

The aspect ratio affects many graphics characteristics, such as the shape of software characters and the roundness of pie charts. Some graphics drivers do not produce correct output if the aspect ratio is anything other than the default. When you use a device that uses local scaling (that is, the device itself can scale the output, for example, some plotters), use ASPECT= to tell SAS/GRAPH the scaling factor. If you change ASPECT, you can use the GTESTIT procedure to run a sample graphics output to inspect the effects of the changes.

Note: You may get more reliable results if you use the default aspect ratio and use the HSIZE= and VSIZE= graphics options to set the dimensions. △

AUTOCOPY

Specifies whether to generate hardcopy automatically.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Defaults: GOPTIONS: NOAUTOCOPY; GDEVICE: AUTOCOPY=N

Restrictions: device dependent; not supported by Java or ActiveX

Syntax

GOPTIONS: AUTOCOPY | NOAUTOCOPY

GDEVICE: AUTOCOPY=Y | N

AUTOCOPY

AUTOCOPY=Y

prints a copy of the graph automatically.

NOAUTOCOPY

AUTOCOPY=N

suppresses printing a copy of the graph. A blank **Autocopy** field in the Parameters window is the same as AUTOCOPY=N.

Details

AUTOCOPY is used only for older terminals that have printers attached directly to the device.

AUTOFEED

Specifies whether devices with continuous paper or automatic paper feed should roll or feed the paper automatically for the next graph.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Defaults: GOPTIONS: AUTOFEED (if a device is specified); GDEVICE: AUTOFEED=Y

Restrictions: device dependent; not supported by Java or ActiveX

See also: PPDFILE

Syntax

GOPTIONS: AUTOFEED | NOAUTOFEED

GDEVICE: AUTOFEED=Y | N

AUTOFEED

AUTOFEED=Y

causes the device to feed new paper automatically for the next graph. A blank **Autofeed** field in the Parameters window is the same as AUTOFEED=Y.

NOAUTOFEED

AUTOFEED=N

suppresses the automatic paper feed.

Details

For PostScript devices, if AUTOFEED is unaltered, the PostScript file is unchanged. If you specify NOAUTOFEED and do not select a PPD file with the PPDFILE option, a PostScript Level 1 manualfeed command is added to the driver output. If you specify NOAUTOFEED and select a PPD that contains a manualfeed option, the procedure code for that manualfeed option is sent. If there is no manualfeed option in the PPD, no manualfeed code is sent. See “PPDFILE” on page 337.

AUTOSIZE

Controls whether to change the size of the character cells in order to preserve the number of rows and columns specified in the device entry.

Used in: GOPTIONS statement

Default: device dependent

Restriction: not supported by Java or ActiveX

See also: DEVOPTS

Syntax

AUTOSIZE=ON | OFF | DEFAULT

ON

changes the cell size in order to preserve the number of rows and columns.

OFF

preserves the device's original cell size and temporarily changes the number of rows and columns.

DEFAULT

uses the default setting (ON or OFF) that is controlled by DEVOPTS bit 50 (see "DEVOPTS" on page 281).

Details

AUTOSIZE is useful when you change the size of the graphics display area using one or more of the options PAPERSIZE, XPIXELS, YPIXELS, XMAX, or YMAX. It lets you control image text size without using PROC GDEVICE. Typically, AUTOSIZE is on for most image drivers and off for all other types of drivers.

Note: If you use HSIZE or VSIZE, the character cell size changes regardless of the AUTOSIZE setting. △

BINDING

Specifies which edge of the document is the binding edge.

Used in: GOPTIONS statement OPTIONS statement

Default: DEFAULTEDGE

Restrictions: PostScript and PCL printers only. PostScript printers require a PPD file. Not supported by Java or ActiveX.

See also: DUPLEX, PPDFILE

Syntax

BINDING=DEFAULTEDGE | LONGEDGE | SHORTEGE

Details

BINDING controls how the page is flipped when DUPLEX is in effect. It does not change the orientation of the graph. DEFAULTEDGE refers to the hardware's

factory-default setting. LONGEDGE and SHORTEdge refer to the paper's long and short edges.

For PostScript printers, a PPD file must also be specified, using the PPDFILE= option. The PPD file contains the command that SAS/GRAPH needs to request the appropriate binding method on the printer being used. If a PPD file is not specified, BINDING= is ignored because SAS/GRAPH will lack the command needed to request the binding method.

BORDER

Specifies whether to draw a border around the graphics output area.

Used in: GOPTIONS statement

Default: NOBORDER

Syntax

BORDER | NOBORDER

Featured in: “Example 8. Creating a Simple Web Page with the ODS HTML Statement” on page 245 and “Example 9. Combining Graphs and Reports in a Web Page” on page 248

Details

The placement of the border on the display is defined by the HSIZE= and VSIZE= graphics options, if used. Otherwise the placement is defined by the XMAX and YMAX device parameters.

CBACK

Specifies the background color of the graphics output.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Gcolors window

Default: as specified in the Gcolors window

Syntax

CBACK=*background-color*

background-color

specifies any SAS/GRAPH color name. See Chapter 6, “SAS/GRAPH Colors and Images,” on page 91 for information about specifying colors.

Details

The CBACK= option is valid on all devices but may be ignored by some (for example, plotters). Specify the default in the Gcolors window of the device entry.

Note: This option overrides the Background and Foreground style attributes in the graph styles. For more information on graph styles, refer to the TEMPLATE procedure documentation in *SAS Output Delivery System: User's Guide*. △

If you explicitly specify a background color with the CBACK= option, the background color you select should contrast with the foreground colors.

If the IBACK= option is in effect, an image will appear in the background in place of the color specified with the CBACK= option.

CBY

Selects the color of the BY lines that appear in the graphics output.

Used in: GOPTIONS statement

Default: (1) CTEXT= graphics option, if used; (2) first color in current colors list

Restriction: not supported by Java or ActiveX

Syntax

CBY=*BY-line-color*

BY-line-color

specifies any SAS/GRAPH color name. See Chapter 6, "SAS/GRAPH Colors and Images," on page 91 for information about specifying colors.

Details

When you use a BY statement with a SAS/GRAPH procedure to process a data set in subgroups, each graph produced by that procedure is headed by a BY line that displays the BY variables and their values that define the current subgroup.

CELL

Controls whether to use cell alignment.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Restriction: not supported by Java or ActiveX

Syntax

GOPTIONS: CELL | NOCELL

GDEVICE: CELL=Y | N

CELL

CELL=Y

causes the device to use cell alignment, in which case SAS/GRAPH attempts to place hardware (or simulated hardware) characters inside character cells. This restriction on the location of characters means that in some cases the SAS/GRAPH procedure may generate axes that do not occupy the entire procedure output area or may be unable to create the requested graph. A blank **cell** field in the Parameters window is the same as CELL=Y.

NOCELL

CELL=N

suppresses cell alignment, causing the procedure to use the entire procedure output area and place axis and tick mark labels without regard to cell alignment.

Details

Specify N in the device entry or use NOCELL in a GOPTIONS statement if you want to preview a graph on a cell-aligned display but intend to produce the final graph on a device that is not cell-aligned, such as a pen plotter.

CHARACTERS

Specifies whether the device's hardware font is used when no font or FONT=NONE is specified in a SAS statement.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Defaults: GOPTIONS: CHARACTERS; GDEVICE: CHARACTERS=Y

Restriction: not supported by Java or ActiveX

Syntax

GOPTIONS: CHARACTERS | NOCHARACTERS

GDEVICE: CHARACTERS=Y | N

CHARACTERS

CHARACTERS=Y

causes SAS/GRAPH to use the device's default hardware font when you do not specify a font in a SAS program. A blank **characters** field in the Parameters window is the same as CHARACTERS=Y.

NOCHARACTERS

CHARACTERS=N

causes SAS/GRAPH to draw the characters using the SIMULATE font and suppresses the use of *all* hardware fonts, regardless of values you specify in other SAS statements.

Details

The hardware font is not used if you changed the HPOS= and VPOS= graphics options from the default, or if you used the HEIGHT= option in a SAS statement *and* the device does not have scalable characters.

CHARREC

Specifies a hardware font for a device by associating a CHARTYPE number with a hardware font. Also defines a default size to use with that font.

Used in: GDEVICE procedure

Default: device dependent

Syntax

CHARREC=(*charrec-list(s)*)

charrec-list

a list of values that correspond to the fields in the Chartype window. *Charrec-list* has this form:

type, rows, cols, 'font', 'Y' | 'N'

<i>type</i>	is the CHARTYPE number and can be an integer from 0 to 9999. (See “CHARTYPE” on page 269 for more information.)
<i>rows</i>	is the number of rows of text in the font that will fit on the display. (See “ROWS” on page 350 for more information.)
<i>cols</i>	is the number of columns of text in the font that will fit on the display. (See “COLS” on page 274 for more information.)
<i>font</i>	is a character string enclosed in quotation marks that contains the name of the corresponding hardware font on the device. (See “FONT NAME” on page 292 for more information.)
Y	represents a scalable font. A scalable font can be displayed at any size. (See “SCALABLE” on page 351 for more information.)
N	represents a nonscalable font. A nonscalable font can be displayed only at a fixed size. (See “SCALABLE” on page 351 for more information.)

For example, these values assign the device’s Helvetica font to be the first hardware font in the CHARTYPE window of the driver entry:

```
charrec=(1, 100, 75, 'helvetica', 'y')
```

CHARTYPE

Selects the number of the default hardware character set.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Restriction: not supported by Java or ActiveX

Syntax

CHARTYPE=*hardware-font-chartype*

hardware-font-chartype

is a nonnegative integer from 0 to 999. *Hardware-font-chartype* refers to the actual number for the hardware font you want to use as listed in the Chartype window of the device entry for the selected device driver. By default, CHARTYPE is 0, which is the default hardware font for the device.

CIRCLEARC

Specifies whether SAS/GRAPH should use the device's hardware circle-drawing capability, if available.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Restriction: not supported by Java or ActiveX

Syntax

GOPTIONS: CIRCLEARC | NOCIRCLEARC

GDEVICE: CIRCLEARC=Y | N

CIRCLEARC

CIRCLEARC=Y

causes SAS/GRAPH to use the built-in hardware circle- and arc-drawing capability of the device. A blank **Circlearc** field in the Parameters window is the same as CIRCLEARC=Y.

Hardware drawing is faster, but not all devices have the capability. SAS/GRAPH device drivers do not try to use the capability if the device does not have it.

NOCIRCLEARC

CIRCLEARC=N

causes SAS/GRAPH to use software move and draw commands to draw circles and arcs.

CMAP

Specifies a color map for the device.

Used in: GDEVICE procedure; GDEVICE Colormap window

Syntax

CMAP=(*from-color* : *to-color*' <...,'*from-color-n* : *to-color-n*'>)

from-color

specifies the name you want to assign to the color designated by the *color* value. In the Colormap window, enter this value in the **From** field.

to-color

specifies any SAS/GRAPH color name up to eight characters long. In the Colormap window, enter this value in the **To** field. See Chapter 6, "SAS/GRAPH Colors and Images," on page 91 for information on specifying colors.

Details

Once you have defined the color mapping, you use the new color name in any color option. For example, if your device entry maps the color name DAFFODIL to the SAS color value PAOY, you can specify the following:

```
pattern1 color=daffodil;
```

and the driver will map this to the color value PAOY.

COLLATE

Specifies whether to collate the output, if collation is supported by the device.

Used in: GOPTIONS statement; OPTIONS statement

Default: NOCOLLATE

Restriction: hardware dependent, PostScript printers require a PPD file; not supported by Java or ActiveX

See also: GPROLOG, PPDFILE

Syntax

COLLATE | NOCOLLATE

Details

A limited number of printers can *collate* output, which means to separate each copy of printed output when you print multiple copies of output.

For PostScript printers, if a device's PPD file has Collate defined as "True", the COLLATE option is supported.

For PCL printers that support collation, use the GPROLOG= option to specify a Printer Job Language (PJM) command to enable the collation. For information on the

appropriate PJJ command, consult the Printer Commands section of your printer's user manual.

COLORS

Specifies the foreground colors used to produce your graphics output if you do not specify colors explicitly in program statements.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Gcolors window

Default: device dependent

Syntax

GOPTIONS: COLORS=<(colors-list | NONE)>

GDEVICE: COLORS=(<colors-list>)

colors-list

specifies one or more SAS color names. If you specify more than one color, separate each name with a blank. See Chapter 6, "SAS/GRAPH Colors and Images," on page 91 for information on specifying colors and using a colors list.

To change some of the colors in the colors list and retain others, you can use a null value for colors you do not want to change. For example, to change COLORS=(RED GREEN BLUE) to COLORS=(WHITE GREEN BROWN), you can specify COLORS=(WHITE,,BROWN).

NONE

tells SAS/GRAPH to use only the colors that you explicitly specify in program statements and to ignore the device's default colors list.

Note: If you specify COLORS=(NONE) and omit a color specification for a graphics element, such as patterns, SAS/GRAPH selects at random one of the colors already specified in your program. △

Featured in: "Example 3. Rotating Plot Symbols through the Colors List" on page 231

Details

The order of the colors in the list is important when you use default colors. For example, the colors used for titles, axes, and surfaces in the G3D procedure are assigned by default according to their position in the colors list.

Note: Colors may be assigned to graph elements in different orders by different devices such as Java and ActiveX. △

If you omit or reset COLORS=, SAS/GRAPH uses the default colors list for the current device. To explicitly reset the colors list to the device default, specify either

```
goptions colors=;
goptions colors=();
```

If you use default patterns with a colors list specified by `COLORS=`, the patterns rotate through every color in the list. If the colors list contains only one color, for example `COLORS=(BLUE)`, the solid pattern is skipped and the patterns rotate through only the appropriate default hatch patterns for the graph.

Note: By default, if black is the first color in a device's colors list, default pattern rotation skips black as a pattern color, but uses black as the area-outline color. Thus, the outline color is never the same as an area's fill color. Using `COLORS=` to change the colors list changes this default pattern behavior. When `COLORS=` is used, all colors in the specified colors list are used in color rotation, and the outline color is the first color in the specified colors list. Thus, the outline color will match any area using the first color as its fill. △

See "PATTERN Statement" on page 169 for more information on pattern rotation.

COLORTBL

An eight-character field in the Gcolors window that is not currently implemented. SAS/GRAPH ignores any value entered into this field.

COLORTYPE

Specifies the color space used by the user-written part of the Metagraphics device driver.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Default: NAME

Syntax

COLORTYPE=NAME | RGB | HLS | GRAY | CMY | CMYK | HSV | HSB

NAME	SAS predefined color names.
RGB	red-green-blue (RGB) color specifications.
HLS	hue-lightness-saturation (HLS) color specifications.
GRAY	gray-scale level.
CMY	cyan-magenta-yellow color specifications.
CMYK	cyan-magenta-yellow-black color specifications.
HSV HSB	hue-saturation-value color specifications. These specifications are also referred to as hue-saturation-brightness (HSB).

See Chapter 6, "SAS/GRAPH Colors and Images," on page 91 for a description of these color types.

Details

Use the COLORTYPE device parameter also to specify the color-naming scheme that is used for devices that support more than one color-naming scheme.

For information about Metagraphics drivers, contact Technical Support.

COLS

Sets the number of columns that the hardware font uses.

Used in: GDEVICE Chartype window; GDEVICE procedure; CHARREC= option

Default: 0

See also: CHARREC

Syntax

See “CHARREC” on page 269 for syntax.

Details

If you are using a device driver from SASHELP.DEVICES, this parameter is already set for hardware fonts that have been defined for your installation. If you are adding to or modifying the hardware fonts available for a particular device driver, specify a positive value for the COLS device parameter. If COLS is greater than 0, it overrides the values of the LCOLS and PCOLS device parameters. For scalable fonts, you can specify 1 for COLS, and the actual number of columns will be computed based on the current text width.

CPATTERN

Selects the default color for PATTERN definitions when a color has not been specified.

Used in: GOPTIONS statement

Default: first color in current colors list

Restriction: not supported by Java or ActiveX

Syntax

CPATTERN=*pattern-color*

pattern-color

specifies any SAS/GRAPH color name. See Chapter 6, “SAS/GRAPH Colors and Images,” on page 91 for information about specifying colors.

Details

CPATTERN= is overridden by any color specification in a PATTERN statement. For details on how CPATTERN= affects the PATTERN statement, see “The Effect of the CPATTERN= Graphics Option” on page 180.

If you specify CPATTERN=, the solid pattern is skipped and the patterns rotate through only the appropriate default hatch patterns for the graph. See “PATTERN Statement” on page 169 for more information on pattern rotation.

CSYMBOL

Specifies the default color for SYMBOL definitions when a color has not been specified.

Used in: GOPTIONS statement

Default: first color in current colors list

Restriction: not supported by Java or ActiveX

Syntax

CSYMBOL=*symbol-color*

symbol-color

specifies any SAS/GRAPH color name. See Chapter 6, “SAS/GRAPH Colors and Images,” on page 91 for information about specifying colors.

Details

CSYMBOL= is overridden by any color specification in a SYMBOL statement. See “SYMBOL Statement” on page 183.

CTEXT

Selects the default color for all text and the border.

Used in: GOPTIONS statement

Default: black for Java and ActiveX devices; for other devices, the first color in current colors list

See also: CTITLE

Restriction: partially supported by Java

Syntax

CTEXT=*text-color*

text-color

specifies any SAS/GRAPH color name. See Chapter 6, “SAS/GRAPH Colors and Images,” on page 91 for information about specifying colors.

Details

The CTITLE= graphics option overrides CTEXT= for all titles, notes, and footnotes, as well as the border. Any other color specifications for text in SAS statements also override the value of CTEXT=.

Note: When you use ODS to send graphics to an HTML destination, and titles and footnotes are rendered as part of the HTML body file instead of the graphic image, you must specify the ODS USEGOPT statement for this option to work. See “Controlling Titles and Footnotes with ODS Output” on page 492 for more information. Δ

CTITLE

Selects the default color for all titles, footnotes, and notes, and the border.

Used in: GOPTIONS statement

Default: (1) color specified by CTEXT=, if used; (2) black for Java and ActiveX devices; for other devices, the first color in current colors list

See also: CTEXT

Syntax

CTITLE=*title-color*

title-color

specifies any SAS/GRAPH color name. See Chapter 6, “SAS/GRAPH Colors and Images,” on page 91 for information about specifying colors.

Details

Any color specification in a TITLE, FOOTNOTE, or NOTE statement overrides the value of CTITLE= for the text. The border, however, still uses the color specified in CTITLE=.

Note: When you use ODS to send graphics to an HTML destination, and titles and footnotes are rendered as part of the HTML body file instead of the graphic image, you must specify the ODS USEGOPT statement for this option to work. See “Controlling Titles and Footnotes with ODS Output” on page 492 for more information. Δ

DASH

Specifies whether to use the device's hardware dashed-line capability, if available.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Restriction: not supported by Java or ActiveX

See also: DASHLINE

Syntax

GOPTIONS: DASH | NODASH

GDEVICE: DASH=Y | N

DASH

DASH=Y

causes SAS/GRAPH to use the built-in hardware dashed-line drawing capability of the device when generating graphics output. A blank **Dash** field in the Parameters window is the same as DASH=Y.

Hardware drawing is faster, but not all devices have the capability. SAS/GRAPH device drivers do not try to use the capability if the device does not have it.

NODASH

DASH=N

causes SAS/GRAPH to draw the dashed lines.

DASHLINE

Specifies which dashed lines should be generated by hardware means if possible.

Used in: GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

See also: DASH

Syntax

DASHLINE=*dashed-line-hex-string*'X

dashed-line-hex-string

is a hexadecimal string 16 characters long that must be completely filled. Each bit in the string corresponds to a line type. See Figure 7.22 on page 208 for line types that correspond to each bit.

To use line type 1, turn on bit 1; to use line type 2, turn on bit 2; and so on. For example, in the following option the first byte is '1000'; only bit 1 is on and only line type 1 is selected:

```
dashline='8000000000000000'x
```

To turn on both bits 1 and 2, specify

```
dashline='c000000000000000'x
```

Bit 1 should always be on because it corresponds to a solid line.

Details

If the DASH device parameter is N in the device entry or if NODASH is used in a GOPTIONS statement, SAS/GRAPH ignores the hexadecimal string in the DASHLINE device parameter.

DASHSCALE

Scales the lengths of the dashes in a dashed line.

Used in: GOPTIONS statement

Default: DASHSCALE=1

Restriction: not supported by Java or ActiveX

Syntax

DASHSCALE=*scaling-factor*

scaling-factor

can be any number greater than 0. For example, GOPTIONS DASHSCALE=.5 reduces any existing dash length by one-half.

Details

Only dashes or spaces with lengths greater than one pixel are scaled. Dots are not scaled because their length is effectively zero. DASHSCALE= always uses software line styles instead of the device's hardware dashed line capabilities.

DELAY

Controls the amount of time between graphs in the animation sequence.

Used in: GOPTIONS statement

Default: 0

Restriction: GIFANIM driver only; not supported by all browsers

Syntax

DELAY=*delay-time*

delay-time

specifies the length of time between graphs in units of 0.01 seconds. For example, to specify a delay of .03 seconds, specify DELAY=3.

Details

SAS/GRAPH puts the DELAY= value into the image file. Based on this value, the browser determines how to display the series of graphs.

DESCRIPTION

Provides a description of the device entry.

Alias: DES

Used in: GDEVICE procedure GDEVICE Detail window

Default: none

Syntax

DESCRIPTION=*'text-string'*

text-string

is a string up to 256 characters long. This is a comment field and does not affect the graphics output.

DEVADDR

Specifies the location of the device to which the output of device drivers is sent.

Used in: GOPTIONS statement

Default: host dependent

Restriction: IBM mainframe systems only

Syntax

DEVADDR=*device-address*

DEVICE

Specifies the device driver to which SAS/GRAPH sends the procedure output. The device driver controls both the form and destination of the output.

Alias: DEV

Used in: GOPTIONS statement OPTIONS statement

Default: device dependent

Syntax

DEVICE=*device-entry*

device-entry

specifies the name of a device entry that is stored in a device catalog.

Details

A device driver can direct graphics output to a hardware device, such as a terminal or a printer, or can create an external file in another graphics file format, such as TIF, GIF, or PostScript. Some device drivers also generate both graphics files and HTML files that can be viewed with a Web browser.

Usually a device driver is assigned by default. If a default driver is not assigned or if you specify RESET=ALL in a GOPTIONS statement, and you do not specify a device driver, SAS/GRAPH prompts you to enter a driver name when you execute a procedure that produces graphics output. If you are producing a graph to the screen and the Graph window is active, SAS/GRAPH selects the display driver for you automatically.

For a description of device drivers and for more information on selecting a device entry and changing device parameters, see Chapter 3, “Device Drivers,” on page 41.

For information on using device drivers to display and print graphics output, see Chapter 4, “SAS/GRAPH Output,” on page 47.

For information on using device drivers to export graphics output to external files, see “About Exporting SAS/GRAPH Output” on page 59. For information on using device drivers to create output for the Web, see “Generating Web Presentations” on page 382.

DEVMAP

Specifies the device map to be used when hardware fonts are used.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host File Options window

Default: device dependent

Restriction: not supported by Java or ActiveX

Syntax

DEVMAP=*device-map-name* | NONE

device-map-name

is a string up to eight characters long that is the name of the device map entry. See Chapter 34, “The GKEYMAP Procedure,” on page 983 for details.

NONE

specifies that you do not want to use a device map. This may cause text to be displayed incorrectly or not at all.

Details

Device maps usually are used only when national characters appear in the text and you want them to display properly.

DEVOPTS

Specifies the hardware capabilities of the device.

Used in: GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Syntax

DEVOPTS='hardware-capabilities-hex-string'X

hardware-capabilities-hex-string

is a hexadecimal string 16 characters long that must be completely filled. The following table lists the hardware capabilities of each bit:

Table 8.1 Device Capabilities Represented in the DEVOPTS String

Bit On	Capability
0	hardware circle generation
1	hardware pie fill supported
2	scalable hardware characters
3	device is a CRT-type (See TYPE device parameter)
4	translate table needed for non-ASCII hosts
5	hardware polygon fill available
6	hardware characters cell-aligned
7	user-definable colors supported
8	hardware polygons with multiple boundaries supported
9	not used
10	not used
11	adjustable hardware line width
12	double-byte font (non-US) supported
13	hardware repaint supported
14	hardware characters supported
15	no hard limit on x coordinate
16	no hard limit on y coordinate
17	not used

Bit On	Capability
18	ability to justify proportional text
19	driver can produce dependent catalog entries
20	device cannot draw in default background color
21	flush device buffer when filled
22	colors defined using HLS
23	colors defined using RGB
24	not used
25	polyline supported
26	polymarker supported
27	graphics clipping supported
28	not used
29	linkable device driver
30	pick CHARTYPE by name in CHARREC entries
31	device dependent pattern support
32	treat SCALABLE=Y CHARREC as metric
33	size CHARTYPE as HW from CHARREC entries
34	device supports rotated arcs
35	device supports target fonts
36	device supports drawing images
37	device supports multiple color maps
38	image rotation direction
39	device requires sublib for image rotation
40	device is a 24 bit truecolor machine
41	device supports setting font attributes
42	use scan line font rendering
43	device can scale images
44	text clipping supported
45	static color device
46	driver does prolog processing
47	driver does epilog processing
48	driver output only uses a file
49	driver output requires a directory or PDS
50	autosize text to fit rows and columns
51	default binding is SHORTEEDGE
52	driver supports duplex printing
53	device does right edge binding

Bit On	Capability
54	ActiveX device
55	Java device

Details

Each capability in the table corresponds to a bit in the value of the DEVOPTS device parameter. For example, if your device can generate hardware pie fills, the second bit in the first byte of the DEVOPTS string should be turned on if you want the driver to use that capability. If your device is capable of generating only hardware circles and pie fills, specify a value of 'C000000000000000'X as your DEVOPTS value (the first byte is '1100' so the first 2 bits of the first byte are set to 1). Many of the hardware capabilities specified in the DEVOPTS string are overridden by graphics options or other device parameters.

CAUTION:

Do not modify the DEVOPTS device parameter unless you are building a Metagraphics driver. If you want to prevent an Institute-supplied driver from using certain hardware capabilities, change the specific device parameter or use the corresponding graphics option. △

If the DEVOPTS string indicates that a capability is available, the driver uses it unless it is explicitly disabled by another device parameter or graphics option. If the DEVOPTS string indicates that the capability is not available, it is not used by the driver, even if the corresponding device parameter or graphics option indicates that it should be used. For example, if the DEVOPTS value indicates that the device can do a hardware pie fill, the driver uses the hardware pie fill capability unless the PIEFILL device parameter is set to N or NOPIEFILL has been specified in a GOPTIONS statement. However, if the DEVOPTS device parameter indicates that the device cannot do a hardware pie fill, the driver does not attempt to use one, even if the PIEFILL device parameter is set to Y or PIEFILL is used in a GOPTIONS statement.

DEVTYPE

Specifies the information required by SAS/GRAPH routines to determine the nature of the output device.

Used in: GDEVICE procedure; GDEVICE Host File Options window

Default: device dependent

Syntax

DEVTYPE=*device-type*

device-type

is a string eight characters long containing either blanks or some token name that is interpreted by the host. *Device-type* can be:

GTERM

indicates that the output device is a graphics device that will be receiving graphics data; most device drivers use this value.

G3270

indicates that the output device is an IBM 3270 graphics data stream. If your device is an IBM 3270 type of device, DEVTYPE= must be G3270.

Note: GTERM and G3270 are SAS/GRAPH device types. Other valid values depend on your operating environment. DEVTYPE supports any of the device-type values supported on the FILENAME statement. Refer to the SAS Help facility for the device types the FILENAME statement supports in your operating environment. In most cases, this field should not be changed. △

DISPLAY

Specifies whether output is displayed on the graphics device but does not affect whether a graph is placed in a catalog.

Used in: GOPTIONS statement

Default: DISPLAY

Restriction: not supported by Java or ActiveX

Syntax

DISPLAY | NODISPLAY

Details

In most cases, NODISPLAY suppresses *all* output except the catalog entry written to the catalog selected in the GOUT= option. Therefore, you usually specify NODISPLAY when you want to generate a graph in a catalog but do not want to display the graph on your monitor or terminal while the catalog entry is being produced.

DISPOSAL

Specifies what happens to the graphic after it is displayed.

Used in: GOPTIONS statement

Default: NONE

Restriction: GIFANIM driver only

Syntax

DISPOSAL=NONE | BACKGROUND | PREVIOUS | UNSPECIFIED

NONE

causes the graphic to be left in place after displaying. This is the default.

BACKGROUND

causes the background color to be returned and the graph erased after displaying.

PREVIOUS

causes the graphic area to be restored with what was displayed in the area previously.

UNSPECIFIED

indicates that no action is necessary.

Details

In Version 6, the ERASE | NOERASE graphics option performed this function for the GIFANIM driver.

DRVINIT

Specifies host commands to be executed before driver initialization.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host Commands window

Restriction: not supported by Java or ActiveX

Syntax

DRVINIT1=*system-command(s)*'

DRVINIT2=*system-command(s)*'

system-command(s)

specifies a character string that is a valid system command and can be in upper- or lowercase letters. You can include more than one command in the string if you separate the commands with a command delimiter, which is host-specific; for example, some operating environments use a semicolon. The length of the entire string cannot exceed 72 characters.

Details

The DRVINIT command is executed before the driver is initialized. DRVINIT is typically used with FILECLOSE=DRIVERTERM to allocate a host file needed by the device driver.

DRVQRY

Specifies whether the device can be queried for information about the current device configuration.

Used in: GDEVICE procedure GDEVICE Detail window

Default: device dependent

Syntax

DRVQRY | NODRVQRY

Details

Generally, this setting is device dependent and you should not change it.

DRVTERM

Specifies host commands to be executed after the driver terminates.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host Commands window

Restriction: not supported by Java or ActiveX

Syntax

DRVTERM1=*'system-command(s)'*

DRVTERM2=*'system-command(s)'*

system-command(s)

specifies a character string that is a valid system command and can be in upper- or lowercase letters. You can include more than one command in the string if you separate the commands with a command delimiter, which is host-specific; for example, some operating environments use a semicolon. The length of the entire string cannot exceed 72 characters.

Details

The DRVTERM command is executed after the driver terminates. DRVTERM is typically used with FILECLOSE=DRIVERTERM to de-allocate a host file and execute utility programs that send the data to the graphics device. For example, DRVTERM might specify commands to send the file to a host print queue.

DUPLEX

Specifies whether to use duplex printing if available on the device.

Used in: GOPTIONS statement; OPTIONS statement

Default: NODUPLEX

Restriction: duplex printers only

See also: BINDING, GSFMODE, PPDFILE

Syntax

DUPLEX | NODUPLEX

Details

When DUPLEX is on, the driver sets up the printer for duplex operation. Before producing the first graph, set GSFMODE=REPLACE on the GOPTIONS statement, and DUPLEX on an OPTIONS or GOPTIONS statement. You can also use the BINDING= option in conjunction with DUPLEX. Before producing the second graph, set GSFMODE=APPEND on the GOPTIONS statement so that the driver knows to place succeeding graphs on the next available side of paper.

If DUPLEX is in effect, the page's inside (binding) margin is set equal to the current HORIGIN setting, and the outside margin is set equal to

$$XMAX - HSIZE - HORIGIN$$

In terms of even- and odd-numbered pages, this means the following:

odd-numbered pages	HORIGIN determines the left margin, and XMAX-HSIZE-HORIGIN determines the right margin
even-numbered pages	XMAX-HSIZE-HORIGIN determines the left margin, and HORIGIN determines the right margin

For PostScript printers, if you do not use the PPDFILE= option to specify a PPD (PostScript Printer Description) file, a generic PostScript Level 1 duplex command is added to the driver output. If PPDFILE= is used, the duplex command is obtained from the PPD file.

ERASE

Specifies whether to erase graph after display.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Defaults: GOPTIONS: NOERASE; GDEVICE: ERASE=N

Restriction: not supported by Java or ActiveX

Syntax

GOPTIONS: ERASE | NOERASE

GDEVICE: ERASE=Y | N

ERASE
ERASE=Y

causes the graph to be erased when you press RETURN after the graph has been displayed.

NOERASE

ERASE=N

causes the graph to remain on the display when you press RETURN after the graph has been displayed. A blank **Erase** field in the Parameters window is the same as ERASE=N.

Details

ERASE is useful for those devices that overlay the graphics area and the message area – that is, those devices that have separate dialog and graphics areas. On other devices, the graph is erased.

EXTENSION

Specifies the file extension for an external graphics file.

Used in: GOPTIONS statement

Default: device dependent

Restriction: not supported by Java or ActiveX

See also: GACCESS, GSFNAME

Syntax

EXTENSION=*file-type*'

file-type

a string up to eight characters long that is a file extension, such as GIF or CGM, that you want to append to an external file.

Details

The extension specified on EXTENSION= is used when the output destination is a storage location. The extension is ignored when the output destination is a file. To specify the output destination, you can use a FILENAME statement, or the graphics options GACCESS= or GSFNAME=.

Assuming the output destination is a storage location,

- if EXTENSION=.', no extension is added to the file name
- if EXTENSION=' or EXTENSION= is not used, the driver's default extension is added to the file name
- if the driver has no default extension, SAS/GRAPH uses the default extension .GSF.

FASTTEXT

Specifies whether to use integer-based font processing for faster font rendering.

Used in: GOPTIONS statement

Default: FASTTEXT

Restriction: not supported by Java or ActiveX

Syntax

FASTTEXT | NOFASTTEXT

FBY

Selects the font for BY lines.

Used in: GOPTIONS statement

Default: (1) font specified by FTEXT=, if used; (2) default hardware font

Restriction: not supported by Java or ActiveX

See also: “BY Statement” on page 141

Syntax

FBY=*BY-line-font*

BY-line-font

specifies the font for all BY-lines on the graphics output. See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for information about specifying fonts.

Details

When you use a BY statement with a SAS/GRAPH procedure to process a data set in subgroups, each graph produced by that procedure is headed by a BY line that displays the BY variables and their values that define the current subgroup.

FCACHE

Specifies the number of software fonts to keep open at one time.

Used in: GOPTIONS statement

Default: FCACHE=3

Restriction: not supported by Java or ActiveX

Syntax

FCACHE=*number-fonts-open*

number-fonts-open

specifies the number of software fonts to keep open. *Number-fonts-open* must be greater than or equal to zero.

Details

Each font requires from 4K to 10K memory. Graphs that use many fonts may run faster if you set the value of *number-fonts-open* to a higher number. However, graphs that use multiple fonts may require too much memory on some computer systems if all the fonts are kept open. In such cases, set the value of *number-fonts-open* to a lower number to conserve memory.

FILECLOSE

Controls when the graphics stream file (GSF) is closed when you are using the device driver to send graphics output to a hardcopy device.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host File Options window

Default: DRIVERTERM (if a device is specified)

Restriction: not supported by Java or ActiveX

See also: “About Exporting SAS/GRAPH Output” on page 59

Syntax

FILECLOSE=DRIVERTERM | GRAPHEND

**DRIVERTERM
DRIVER**

closes the GSF and makes it available to the device after all graphs have been produced and the procedure or driver terminates. A host command may be needed to actually send the GSF to the device. Host commands may be specified with the DRVINIT or DRVTERM parameters or entered in the Host File Options window of the device entry.

If multiple graphs are produced by a procedure, this specification creates one large file. Specifying DRIVERTERM is appropriate for batch processing because it is slightly more efficient to allocate the file only once.

**GRAPHEND
GRAPH**

closes the GSF after each separate graph is produced and releases it to the device before sending another. This method creates smaller files if multiple graphs are produced by a procedure. You can specify a command that sends the graph to the device with the POSTGRAPH parameter or use the Host File Options window.

Specifying GRAPHEND is appropriate for drivers that are used interactively, or for devices that require only one graph per physical file.

FILEONLY

Specifies whether a file or a storage location is the default destination for graphics output.

Used in: GOPTIONS statement

Default: device dependent

Restriction: FILEONLY ignored if the device requires the output destination to be a storage location; not supported by Java or ActiveX

See also: DEVOPTS, GSFNAME

Syntax

FILEONLY | NOFILEONLY

FILEONLY

specifies that a file rather than a storage location is the default destination for graphics output.

NOFILEONLY

specifies that a storage location is the default destination for graphics output, unless a file of the same name exists.

Details

Most devices use FILEONLY as the default. However, devices that require the output destination to be a storage location use NOFILEONLY as the default. For example, the HTML and WEBFRAME devices require a storage location because they produce two types of output (HTML files and GIF image files) that cannot be written to the same file.

To determine what the default is for a particular device, look at the settings for DEVOPTS bits 48 and 49.

For more information, see “Exporting SAS/GRAPH Output with Program Statements” on page 62.

FILL

Specifies whether to use the device’s hardware rectangle-fill capability.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Restriction: not supported by Java or ActiveX

Default: device dependent

Syntax

GOPTIONS: FILL | NOFILL

GDEVICE: FILL=Y | N

FILL**FILL=Y**

causes SAS/GRAPH to use the built-in hardware rectangle-filling capability of the device. A blank **F**ill field in the Parameters window is the same as FILL=Y.

Hardware drawing is faster, but not all devices have the capability. SAS/GRAPH does not try to use the capability if your device does not support it.

NOFILL**FILL=N**

causes SAS/GRAPH to use software fills to fill rectangles.

FILLINC

Specifies the number of pixels to move before drawing the next line in a software fill of a solid area.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Restriction: not supported by Java or ActiveX

See also: FILL, PIEFILL, POLYGONFILL

Syntax

FILLINC= 0...9999

Details

In order for FILLINC to have any effect, a software fill must be used. To force a software fill, use the options NOFILL, NOPIEFILL, and NOPOLYGONFILL in a GOPTIONS statement.

If FILLINC is set to 0 or 1, adjacent lines are used (solid fill with no gaps). If FILLINC is set to 2, a pixel-width line is skipped before drawing the next line of a fill.

This option can be useful for keeping plotters from over saturating a solid area and for speeding the plotting. Some inks spread on paper. The type of paper used also can affect ink spread.

FONT NAME

Specifies the hardware font associated with CHARTYPE.

Used in: GDEVICE Chartype window; GDEVICE procedure; CHARREC= option

Required if adding or modifying a CHARREC

See also: CHARREC

Syntax

See "CHARREC" on page 269 for syntax.

Details

Use FONT NAME if you are adding to or modifying the hardware fonts available for a particular device driver. The fonts that you specify must be valid for the output device. If you are using an Institute-supplied device entry, this parameter already is set for most available hardware fonts.

FONTRES

Controls the resolution of software fonts.

Used in: GOPTIONS statement

Default: NORMAL

Restriction: not supported by Java or ActiveX

See also: FASTTEXT, FCACHE, RENDER, RENDERLIB, SWFONTRENDER

Syntax

FONTRES=NORMAL | PRESENTATION

NORMAL

renders fonts in memory using integer rendering routines, which improves character drawing speed for most host systems. NORMAL has the same effect as specifying the default values for these graphics options.

```
render=memory
renderlib=saswork
fasttext
fcache=0
```

PRESENTATION

disables the storage or use of rendered versions of Bitstream fonts, but produces the fonts at their highest resolution. FONTRES=PRESENTATION has the same effect as specifying these graphics options:

```
render=none
renderlib=saswork
nofasttext
fcache=3
```

FORMAT

Sets the file format of the metacode file produced by the Institute-supplied part of the Metagraphics device driver.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Default: CHARACTER

Restriction: Used only with user-supplied Metagraphics drivers.

Syntax

FORMAT=CHARACTER | BINARY

Details

A blank field defaults to CHARACTER. For information about Metagraphics drivers, contact Technical Support.

FTEXT

Selects the default font for all text.

Used in: GOPTIONS statement

Default: Default hardware font (except the first title)

Restriction: partially supported by Java or ActiveX

See also: FTITLE

Syntax

FTEXT=*text-font*

text-font

specifies the font for all text on the graphics output. See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for information about specifying fonts.

Featured in: “Example 6. Enhancing Titles” on page 238

Details

The FTITLE= graphics option overrides FTEXT= for the *first* title. Not all fonts are supported by the ActiveX and Java devices.

Note: When you use ODS to send graphics to an HTML destination, and titles and footnotes are rendered as part of the HTML body file instead of the graphic image, you must specify the ODS USEGOPT statement for this option to work. See “Controlling Titles and Footnotes with ODS Output” on page 492 for more information. △

FTITLE

Selects the default font for the first TITLE line.

Used in: GOPTIONS statement

Default: (1) font specified by FTEXT=, if used; (2) SWISS font

See also: FTEXT

Syntax

FTITLE=*title-font*

title-font

specifies the font for the TITLE1 statement. See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for information about specifying fonts.

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226 and “Example 6. Enhancing Titles” on page 238

Details

Note: When you use ODS to send graphics to an HTML destination, and titles and footnotes are rendered as part of the HTML body file instead of the graphic image, you must specify the ODS USEGOPT statement for this option to work. See “Controlling Titles and Footnotes with ODS Output” on page 492 for more information. △

FTRACK

Controls the amount of space between letters in the Institute-supplied Bitstream fonts (Brush, Century, Swiss, and Zapf).

Used in: GOPTIONS statement

Default: TIGHT

Restriction: not supported by Java or ActiveX

Syntax

FTRACK=LOOSE | NONE | NORMAL | TIGHT | TOUCH | V5

LOOSE

leaves the most visible space between characters and produces a longer string.

NONE

spacing depends on the size of the font. NONE might produce a shorter or longer string than LOOSE for the same font at different point sizes, because some sizes add space between the characters while others remove it.

NORMAL

is the recommended setting.

TIGHT

reduces the space between characters.

TOUCH

leaves the least visible space between characters.

V5

places a fixed amount of space between the characters and does not adjust for the shape of the character; that is, it does not support kerning. This spacing is compatible with Version 5 Bitstream fonts.

Details

The spacing you specify with FTRACK= affects all Bitstream text in a graph. For example, you cannot produce TIGHT Century type and LOOSE Zapf type simultaneously. This option has no effect on other font types.

Because the value of FTRACK= is stored with the graph, the spacing that you specify when the graph is created is always used when the graph is replayed.

GACCESS

Specifies the format or the destination or both of graphics data written to a device or graphics stream file (GSF).

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host File Options window

Default: device dependent

Restriction: not supported by Java or ActiveX

Syntax

GACCESS=*output-format* | '*output-format destination*'

output-format

specifies the format or the destination (the SAS log or a fileref) of the graphics data. *Output-format* varies according to the operating environment. These values can be specified in all operating environments:

SASGASTD

specifies that a continuous stream of data is written. SASGASTD is the default for most devices and is typically appropriate when the output file will be sent directly to a device. If you specify GACCESS=SASGASTD, use the GSFNAME= and GSFMODE= graphics options or device parameters to direct your graphics output to a GSF.

SASGAEDT

specifies that the file be host-specific editable format. Some hosts allow editing by inserting characters at the end of each record. SASGAEDT is typically used when the output file is to be edited later. If you specify GACCESS=SASGAEDT, use the GSFNAME= and GSFMODE= graphics options or device parameters to direct your graphics output to a GSF.

SASGAFIX

specifies that fixed-length records be written. (The record length is controlled by the value of the GSFLLEN= graphics option or device parameter or the sixth byte of

the PROMPTCHARS value.) The records are padded with blanks where necessary. SASGAFIX is typically used when the output file will be transferred to a computer that requires fixed-length records. If you specify GACCESS=SASGAFIX, use the GSFNAME= and GSFMODE= graphics options or device parameters to direct your graphics output to a GSF.

Note: The value of the GPROTOCOL= graphics option or device parameter can greatly affect the length of the records; for example, if GPROTOCOL=SASGPLCL, the length of the records is doubled. △

SASGALOG

specifies that records are to be written to the SAS log.

GSASFILE

specifies that the records are to be written to the destination whose fileref is GSASFILE. The fileref can point to a specific external file or to an aggregate file location. See “FILENAME Statement” on page 28 for more information on specifying a fileref.

'output-format destination'

specifies the destination in addition to one of these output format values: SASGASTD, SASGAEDT, or SASGAFIX. *Destination* is the physical name of an external file or aggregate file location, or of a device. For details on specifying the physical name of a destination, see the SAS documentation for your operating environment.

This form is not available in all operating environments. See “About Graphics Stream Files” on page 60 for more information on creating graphics stream files.

Note: In the **Gaccess** field of the Host File Options window, you can specify a destination without an output format, in which case the format defaults to SASGASTD. When you specify a value in the **Gaccess** field, you do not need to quote it. △

Operating Environment Information: Depending on your operating environment, you may be able to specify other values for GACCESS=. See the SAS companion for your operating environment for additional values. △

GCLASS

Specifies the output class for IBM printers

Used in: GOPTIONS statement

Default: GCLASS=G

Restriction: used only with IBM3287 and IBM3268 device drivers on z/OS systems only

Syntax

GCLASS=SYSOUT-*class*

Details

Specifies the SYSOUT class to which the IBM3287 and IBM3268 device driver output is written.

GCOPIES

Sets the current and maximum number of copies to print.

Used in: GOPTIONS statements; GDEVICE Parameters window; GDEVICE procedure; OPTIONS statement

Defaults: GOPTIONS: GCOPIES=(0,20) GDEVICE: GCOPIES=0

Restriction: not supported by Java or ActiveX

Syntax

GOPTIONS: GCOPIES=(*current-copies*<,<*max-copies*>)

GDEVICE: GCOPIES=*current-copies*

current-copies

is a nonnegative integer ranging from 0 through 255, but it cannot exceed the *max-copies* value specified. A value of 0 or 1 produces a single copy.

max-copies

is a nonnegative integer ranging from 1 through 255.

If you do not specify GCOPIES, a default number of copies is searched for in this order:

- 1 the number of copies specified on an OPTIONS COPIES setting
- 2 0 current copies, and 20 maximum copies.

Details

Not all devices have the capability to print multiple copies. See the **Gcopies** field in the Parameters window for your device to determine its capabilities.

GDDMCOPY

Instructs the driver to issue either an FSCOPY or GSCOPY call to GDDM when AUTOCOPY is in effect.

Used in: GOPTIONS statement

Default: FSCOPY

Restriction: GDDM device drivers on IBM mainframe systems only

See also: AUTOCOPY

Syntax

GDDMCOPY=FSCOPY | GSCOPY

FSCOPY

used when sending output to an IEEE attached plotter.

GSCOPY

used when creating an ADMPRINT file for output on 3287-type printers.

GDDMNICKNAME

Selects a GDDM nickname for the device to which output is sent.

Alias: GDDMN

Used in: GOPTIONS statement

Restriction: GDDM device drivers on IBM mainframe systems only

Syntax

GDDMNICKNAME=*nickname*

Details

Refer to the SAS Help facility for details on using GDDM drivers and options.

GDDMTOKEN

Selects a GDDM token for the device to which output is sent.

Alias: GDDMT

Used in: GOPTIONS statement

Restriction: GDDM device drivers on IBM mainframe systems only

Syntax

GDDMTOKEN=*token*

Details

Refer to the SAS Help facility for details on using GDDM drivers and options.

GDEST

Specifies the JES SYSOUT destination for IBM printers.

Used in: GOPTIONS statement

Default: LOCAL

Restriction: used only with IBM3287 and IBM3268 device drivers on z/OS systems

Syntax

GDEST=*destination*

GEND

Appends an ASCII string to every graphics data record that is sent to a device or file.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Gend window

Restriction: not supported by Java or ActiveX

See also: GSTART

Syntax

GEND=*'string'* <...*'string-n'*>

'string'

can be either of the following:

*'hex-string'*X

'character-string'

In a GOPTIONS statement or in the GDEVICE procedure ADD or MODIFY statement, you can specify multiple strings with the GEND= option. In this case, you can mix the formats, specifying some as ASCII hexadecimal strings and some as character strings. Multiple strings are concatenated automatically.

In the GEND window, enter the hexadecimal string without either quotation marks or a trailing x. Note, however, that the string must be entered as a hexadecimal string.

PROC GOPTIONS always reports the value as a hexadecimal string.

Details

GEND is useful if you are creating a file and want to insert a carriage return at the end of every record. You can also use GEND in conjunction with the GSTART= graphics option or device parameter.

If you must specify the long and complicated initialization strings required by some devices (for example, PostScript printers), it is easier to use the GOPTIONS GEND= option rather than the GDEVICE Gend window because it is easier to code the string as text with GEND= than it is to convert the string to its ASCII representation, which is required to enter the string in the GDEVICE Gend window.

Note: On non-ASCII hosts, only ASCII hexadecimal strings produce consistent results in all instances because of the way the character strings are translated. In addition, the only way to specify a value for GEND that can be used by all hosts is to use an ASCII hexadecimal string; therefore, using an ASCII hexadecimal string to specify a value for GEND is the recommended method. △

GPILOG

Sends a string to a device or file after all graphics commands are sent.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Gepilog window

Restriction: not supported by Java or ActiveX

See also: PREGPILOG, POSTGPILOG

Syntax

GPILOG=*'string'* <...*'string-n'*>

'string'

can be either of the following:

*'hex-string'*X

'character-string'

In a GOPTIONS statement or in the GDEVICE procedure ADD or MODIFY statement, you can specify multiple strings with the GPILOG= option. In this case, you can mix the formats, specifying some as ASCII hexadecimal strings and some as character strings. Multiple strings are concatenated automatically.

In the Gepilog window, enter the hexadecimal string without either quotation marks or a trailing x. Note, however, that the string must be entered as a hexadecimal string.

PROC GOPTIONS always reports the value as a hexadecimal string.

Details

GPILOG can be used in conjunction with the GPROLOG= graphics option or device parameter.

If you must specify the long and complicated initialization strings required by some devices (for example, PostScript printers), it is easier to use the GOPTIONS GPILOG= option rather than the Gepilog window because it is easier to code the string as text with GPILOG= than it is to convert the string to its ASCII representation, which is required to enter the string in the Gepilog window.

Note: On non-ASCII hosts, only ASCII hexadecimal strings produce consistent results in all instances because of the way the character strings are translated. In addition, the only way to specify a value for GPILOG that can be used by all hosts is to use an ASCII hexadecimal string; therefore, using an ASCII hexadecimal string to specify a value for GPILOG is the recommended method. △

GFORMS

Specifies the JES form name for IBM printers.

Used in: GOPTIONS statement

Default: STD

Restriction: used only with IBM3287 and IBM3268 device drivers on z/OS systems only

Syntax

GFORMS=*'forms-code'*

GOUTMODE

Appends to or replaces the graphics output catalog.

Used in: GOPTIONS statement

Default: APPEND

Restriction: not supported by Java or ActiveX

Syntax

GOUTMODE=APPEND | REPLACE

APPEND

adds each new graph to the end of the current catalog.

REPLACE

replaces the contents of the catalog with the graph or graphs produced by a single procedure.

CAUTION:

If you specify REPLACE, the *entire contents of the catalog are replaced, not just graphs of the same name*. Graphs are added to the catalog for the duration of the procedure, but when the procedure ends and a new procedure begins, the contents of the catalog are deleted and the new graph or graphs are added. △

GPROLOG

Sends a string to device or file before graphics commands are sent.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Gprolog window

Restriction: not supported by Java or ActiveX

See also: PREGPROLOG, POSTGPROLOG

Syntax

GPROLOG=*'string'* <...*'string-n'*>

'string'

can be either of the following:

*'hex-string'*X

'character-string'

In a GOPTIONS statement or in the GDEVICE procedure ADD or MODIFY statement, you can specify multiple strings with the GPROLOG= option. In this case, you can mix the formats, specifying some as ASCII hexadecimal strings and some as character strings. Multiple strings are concatenated automatically.

In the GPROLOG window, enter the hexadecimal string without either quotation marks or a trailing x. Note, however, that the string must be entered as a hexadecimal string.

PROC GOPTIONS always reports the value as a hexadecimal string.

Details

GPROLOG can be used in conjunction with the GEPiLOG= graphics option or device parameter.

If you must specify the long and complicated initialization strings required by some devices (for example, PostScript printers), it is easier to use the GOPTIONS GPROLOG= option rather than the GDEVICE Gprolog window because it is easier to code the string as text with GPROLOG= than it is to convert the string to its ASCII representation, which is required to enter the string in the GDEVICE Gprolog window.

Note: On non-ASCII hosts, only ASCII hexadecimal strings produce consistent results in all instances because of the way the character strings are translated. In addition, the only way to specify a value for GEND that can be used by all hosts is to use an ASCII hexadecimal string; therefore, using an ASCII hexadecimal string to specify a value for GEND is the recommended method. △

GPROTOCOL

Specifies the protocol module to use when routing output directly to a printer or creating a graphics stream file (GSF) to send to a device attached to your host by a protocol converter.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host File Options window

Restriction: not supported by Java or ActiveX

Default: host dependent

Syntax

GPROTOCOL=*module-name*

***module-name* can be one of these**

SASGPADE*

SASGPAGL*

SASGPASC
 SASGPAXI*
 SASGPCAB*
 SASGPCHK*
 SASGPDAT*
 SASGPDCA*
 SASGPHEX
 SASGPHYD*
 SASGPIDA*
 SASGPIDX*
 SASGPIMP*
 SASGPIOC*
 SASGPISI*
 SASGPI24*
 SASGPLCL*
 SASGPNET*
 SASGPMIC*
 SASGPRTM*
 SASGPSCS*
 SASGPSTD
 SASGPSTE*
 SASGPTCX*
 SASGPVAT*
 SASGP497*
 SASGP71

*Valid only for IBM mainframe systems.

Details

GPROTOCOL= specifies whether the graphics data generated by the SAS/GRAPH device driver should be altered and how the data should be altered. Unless you are using a protocol converter on an IBM mainframe, most devices do not require that the data be altered, and ordinarily, you do not have to change the default of GPROTOCOL.

On IBM hosts, the protocol module converts the graphics output to a format that can be processed by protocol converters. On other hosts, it can be used to produce a file in ASCII hexadecimal format.

Refer to the SAS Help facility for descriptions of these protocol modules.

Operating Environment Information: GPROTOCOL is valid only in certain operating environments. Δ

GRAPHRC

Specifies whether to return a step code at graphics procedure termination.

Used in: GOPTIONS statement

Restriction: not supported by Java or ActiveX

Default: GRAPHRC

Syntax

GRAPHRC | NOGRAPHRC

GRAPHRC

allows a return code at procedure termination. If the return code is not 0, the entire job may terminate.

NOGRAPHRC

always returns a step code of 0, even if the SAS/GRAPH program produced errors. As a result, the entire job's return code is unaffected by errors in any graphics procedure. NOGRAPHRC also overrides the ERRABEND system option.

Details

You typically use this option when you are running multiple jobs in a batch environment. It is useful primarily in an z/OS batch environment.

GSFLEN

Controls the length of records written to the graphics stream file (GSF).

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host File Options window

Default: device dependent

Restriction: not supported by Java or ActiveX

See also: PROMPTCHARS

Syntax

GSFLEN=*record-length*

record-length

must be a nonnegative integer up to five digits long (0...99999). GSFLEN= specifies the length of the records written by the driver to a GSF or to the device.

If GSFLEN is 0, SAS/GRAPH uses the sixth byte of the PROMPTCHARS string to determine the length of the records. If the sixth byte of the PROMPTCHARS string is 00, the device driver sets the record length.

If you specify `GACCESS=SASGAFIX` and omit `GSFLEN=`, SAS/GRAPH uses the default length for the device.

Some values of the `GPROTOCOL` device parameter cause each byte in the data stream to be expanded to two bytes. This expansion is done after the length of the record is set by `GSFLEN`. If you are specifying a value for `GPROTOCOL` that does this (for example, `SASGPHEX`, `SASGPLCL`, or `SASGPAGL`), specify a value for `GSFLEN` that is half of the actual record length desired. For example, a value of 64 produces a 128-byte record after expansion by the `GPROTOCOL` module.

GSFMODE

Specifies the disposition of records written to a graphics stream file (GSF) or to a device or communications port by the device driver.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host File Options window

Default: REPLACE

Restriction: not supported by Java or ActiveX

See also: GACCESS, GSFNAME

Syntax

GSFMODE=APPEND | PORT | REPLACE

APPEND

adds the records to the end of a GSF designated by the `GACCESS=` or `GSFNAME=` graphics option or device parameter. If the file does not already exist, it is created.

The destination can be either a specific file or an aggregate file storage location.

If the destination of the GSF is a specific file and you specify `APPEND`, SAS/GRAPH will add the new records to an existing GSF of the same name.

If the destination of the GSF is a file location and not a specific file, SAS/GRAPH will add the records to an external file whose name matches the name of the newly created catalog entry. For more information on how SAS/GRAPH names catalog entries, see “Exporting SAS/GRAPH Output with Program Statements” on page 62.

Note: Some viewers of bitmapped output can view only one graph, even though multiple graphs are stored in the file. Therefore it may appear that a file contains only one graph when in fact it contains multiple graphs. △

PORT

sends the records to a device or communications port. The `GACCESS=` graphics option or device parameter should point to the desired port or device.

REPLACE

replaces the existing contents of a GSF designated by the `GACCESS=` or `GSFNAME=` graphics option or device parameter. If the file does not exist, it is created. `REPLACE` is always the default, regardless of the destination of the GSF.

If the destination of the GSF is a specific file and you specify `REPLACE`, SAS/GRAPH will replace an existing GSF with the contents of a newly created GSF of the same name.

If the destination of the GSF is a file location and not a specific file, SAS/GRAPH will replace an external file whose name matches the name of the newly created catalog entry. For more information on how SAS/GRAPH names catalog entries, see “Exporting SAS/GRAPH Output with Program Statements” on page 62.

Details

When you create a GSF, the GSFNAME= or GACCESS= graphics option or device parameter controls where the output goes, and GSFMODE= controls how the driver writes graphics output records. If the output is to go to a file, specify APPEND or REPLACE. If the output is to go directly to a device or to a communications port, specify PORT. See “About Graphics Stream Files” on page 60 for more information on creating a graphics stream file.

GSFNAME

Specifies the fileref of the file or aggregate file location to which graphics stream file records are written.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host File Options window

Restriction: Not valid for IBM32xx, linkable, Metagraphics, Java, or ActiveX drivers.

See also: GACCESS, GSFMODE

Syntax

GSFNAME=*fileref*

fileref

specifies a fileref that points to the destination for the graphics stream file (GSF) output. *Fileref* must be a valid SAS fileref up to eight characters long and must be assigned with a FILENAME statement prior to running a SAS/GRAPH procedure that uses that fileref. The destination specified by the FILENAME statement can be either a specific file or an aggregate file location. See “FILENAME Statement” on page 28 for additional information on the FILENAME statement.

Details

Whether the resulting graphs are stored as one file or many files depends on both the type of destination and the setting of the GSFMODE= option.

If you specify a fileref with GSFNAME= and forget the FILENAME statement that defines the fileref, and if a destination is specified by the GACCESS= graphics option or device parameter, SAS/GRAPH assigns that destination to the fileref and sends the graphics output there. See also “GACCESS” on page 296.

See “About Graphics Stream Files” on page 60 for more information on creating graphics stream files.

GSFPROMPT

Specifies whether to write prompt messages to the graphics stream file (GSF).

Used in: GOPTIONS statement

Default: NOGSFPROMPT

Restriction: not supported by Java or ActiveX

Syntax

GSFPROMPT | NOGSFPROMPT

Details

When the GSF is processed by another program, that program can display the prompt messages. The default, NOGSFPROMPT, is compatible with Release 6.06.

Although the prompt messages appear if the graphics device is in eavesdrop mode, they do not wait for user response. If GSF PROMPT is on, the prompt messages are sent with the GSF to the device, regardless of the status of the graphics options PROMPT, GACCESS=, GSFMODE=, or GSFNAME=.

GFSIZE

Sets the number of lines of display used for graphics for devices whose displays can be divided into graphics and text areas.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Restriction: not supported by Java or ActiveX

Default: device dependent

Syntax

GFSIZE=*lines*

lines

specifies the number of lines to be used for graphics. *Lines* is a nonnegative integer up to three digits long (0...999), and can be larger or smaller than the total number of lines that can be displayed at one time. If the number is larger, scroll the graph to see it all. If GFSIZE is 0, all lines are used for text.

GSTART

Prefixes every record of graphics data sent to a device or file with a string of characters.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Gstart window

Default: none

Restriction: not supported by Java or ActiveX

See also: GEND

Syntax

GSTART=*'string <... 'string-n'>*

'string'

can be either of the following:

'hex-string'X

'character-string'

In a GOPTIONS statement or in the GDEVICE procedure ADD or MODIFY statement, you can specify multiple strings with the GSTART= option. In this case, you can mix the formats, specifying some as ASCII hexadecimal strings and some as character strings. Multiple strings are concatenated automatically.

In the GSTART window, enter the hexadecimal string without either quotation marks or a trailing x. Note, however, that the string must be entered as a hexadecimal string.

PROC GOPTIONS always reports the value as a hexadecimal string.

Details

GSTART is useful when sending a file to a device that requires each record be prefixed with some character. You can use GSTART= in conjunction with the GEND= graphics option or device parameter.

If you must specify the long and complicated initialization strings required by some devices (for example, PostScript printers), it is easier to use the GOPTIONS GSTART= option rather than the GDEVICE Gstart window because it is easier to code the string as text with GSTART= than it is to convert the string to its ASCII representation, which is required to enter the string in the GDEVICE Gstart window.

Note: On non-ASCII hosts, only ASCII hexadecimal strings produce consistent results in all instances because of the way the character strings are translated. In addition, the only way to specify a value for GEND that can be used by all hosts is to use an ASCII hexadecimal string; therefore, using an ASCII hexadecimal string to specify a value for GEND is the recommended method. △

GUNIT

Specifies the default unit of measure to use with height specifications.

Used in: GOPTIONS statement

Default: CELLS

Restriction: partially supported by Java or ActiveX

Syntax

GUNIT=*units*

units must be one of

CELLS	character cells
CM	centimeters
IN	inches
PCT	percentage of the graphics output area
PT	points (there are approximately 72 points in an inch).

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226 and “Example 3. Rotating Plot Symbols through the Colors List” on page 231

Details

Used with options in the AXIS, FOOTNOTE, LEGEND, NOTE, SYMBOL, and TITLE statements and in some graphics options. If you specify a value but do not specify an explicit unit, the value of the GUNIT= graphics option is used.

GWAIT

Specifies the time between each graph displayed in a series.

Used in: GOPTIONS statement

Default: GWAIT=0

Restriction: not supported by Java or ActiveX

Syntax

GWAIT=*seconds*

seconds

specifies the number of seconds between graphs. *Seconds* can be any reasonable positive integer. By default, GWAIT=0, which means that you must press the RETURN key between each display in a series of graphs.

Details

GWAIT= enables you to view a series of graphs without having to press the ENTER key (or the RETURN or END key, depending on your device) between each display. For example, if you specify GWAIT=5, five seconds elapse between the display of each graph in a series. If you use the NOPROMPT graphics option, the GWAIT= graphics option is disabled.

GWRITER

Specifies the name of the external writer used with IBM printers.

Used in: GOPTIONS statement

Default: SASWTR

Restriction: Used only with IBM3287 and IBM3268 device drivers on z/OS systems

Syntax

GWRITER=*'writer-name'*

HANDSHAKE

Specifies the type of flow control used to regulate the flow of data to a hardcopy device.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: host dependent

Restriction: not supported by Java or ActiveX

Syntax

HANDSHAKE=HARDWARE | NONE | SOFTWARE | XONXOFF

HARDWARE

HARD

specifies that SAS/GRAPH instruct the device to use the hardware CTS and RTS signals. (This is not appropriate for some devices.)

NONE

specifies that SAS/GRAPH send data without providing flow control. Specify NONE only if the hardware or interface program you are using provides its own flow control.

SOFTWARE

SOFT

specifies that SAS/GRAPH use programmed flow control with plotters in eavesdrop mode.

XONXOFF

X

specifies that SAS/GRAPH instruct the device to use ASCII characters DC1 and DC3. (This is not appropriate for some devices.)

Details

HANDSHAKE regulates flow of control by specifying how and if a device can signal to the host to temporarily halt transmission and then resume it. Flow control is important

because it is possible to send commands to a hardcopy device faster than they can be executed.

HANDSHAKE can be used when you are using a protocol converter, interface program, or host computer that can perform XONXOFF or hardware handshaking. You also can use this option if you are routing output through flow-control programs of your own, as in a multiple-machine personal computer environment where the graphics plotter is a shared resource. SAS/GRAPH software sends output to a server (the file transfer does not require flow control). The server queues incoming graphs and sends them to the plotter. The server, rather than SAS/GRAPH software, is responsible for handling flow control.

If you do not use HANDSHAKE, the value in the driver entry is used.

If you use HANDSHAKE=XONXOFF or HANDSHAKE=HARDWARE, SAS/GRAPH does not actually do the handshaking. It tells the device which type of handshake is being used. The protocol converter, interface program, or host computer actually does the handshake.

Note: If you are creating a graphics stream file using a driver for a plotter and you specify HANDSHAKE=SOFTWARE, the software that you use to send the file to the plotter must be able to perform a software handshake. You will probably want to specify one of the alternative values if you route output to a file. △

HBX

Specifies the height of BX lines generated when you use BX-group processing.

Used in: GOPTIONS statement

Default: 1 cell unless HTEXT= is used

Restriction: not supported by Java or ActiveX

See also: “BX Statement” on page 141

Syntax

HBX=*BY-line-height* <*units*>

***BY-line-height* <*units*>**

specifies the height of BX-line text; by default *BY-line-height* is 1. If you specify HBX=0, the BX headings are suppressed. For a description of *units*, see “Specifying Units of Measurement” on page 262.

Note: If a value for *units* is not specified, the current units associated with the GUNIT graphics option are used. △

Details

When you use a BX statement with a SAS/GRAPH procedure to process a data set in subgroups, each graph produced by that procedure is headed by a BX line that displays the BX variables and their values that define the current subgroup.

HEADER

Specifies the command that executes a user-supplied program to create HEADER records for the driver.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Restriction: Used only with user-supplied Metagraphics drivers.

See also: HEADERFILE

Syntax

HEADER=*command*'

command

specifies a command that runs a user-written program that creates the file of HEADER records. *Command* is a string up to 40 characters long.

Details

For information about Metagraphics drivers, contact Technical Support.

HEADERFILE

Specifies the fileref for the file from which the Metagraphics driver reads HEADER records.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Restriction: Used only with user-supplied Metagraphics drivers.

See also: HEADER

Syntax

HEADERFILE=*fileref*

fileref

specifies a valid SAS fileref up to eight characters long. *Fileref* must have been previously assigned with a FILENAME statement or a host command before running the Metagraphics driver. See “FILENAME Statement” on page 28 for details.

Details

For information about Metagraphics drivers, contact Technical Support.

HORIZIN

Sets the horizontal offset from the lower-left corner of the display area to the lower-left corner of the graph.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Detail window

Restriction: not supported by Java or ActiveX

See also: VORIGIN

Syntax

HORIZIN=*horizontal-offset* <IN | CM | PT>

***horizontal-offset* <IN | CM | PT>**

must be a nonnegative number and may be followed by a unit specification, either IN for inches (default), or CM for centimeters, or PT for points. If you do not specify HORIZIN, a default offset is searched for in this order:

- 1 the left margin specification on an OPTIONS LEFTMARGIN setting
- 2 HORIZIN setting in the device catalog.

Details

The display area is defined by the XMAX and YMAX device parameters. By default, the origin of the graphics output area is the lower-left corner of the display area; the graphics output is offset from the lower-left corner of the display area by the values of HORIZIN and VORIGIN. HORIZIN + HSIZE cannot exceed XMAX. See “About the Graphics Output Area” on page 34 for details.

HOSTSPEC

Stores FILENAME statement options in the device entry.

Used in: GDEVICE procedure; GDEVICE Host File Options window

Syntax

HOSTSPEC=*'text-string'*

text-string

specifies FILENAME statement options that are valid for the operating environment. *Text-string* accepts characters in upper or lower case. See the SAS documentation for your operating environment for details.

Details

HOSTSPEC may be used when the driver dynamically allocates a graphics stream file or spool file. It can specify the attributes of the file, such as record format or record length. It cannot be used with Metagraphics drivers.

HPOS

Specifies the number of columns in the graphics output area.

Used in: GOPTIONS statement

Default: device dependent: the value of the LCOLS or PCOLS device parameter

Restriction: not supported by Java or ActiveX

See also: PCOLS, LCOLS, VPOS

Syntax

HPOS=*columns*

columns

specifies the number of columns in the graphics output area, which is equivalent to the number of hardware characters that can be displayed horizontally. Specifying HPOS=0 causes the device driver to use the default hardware character cell width for the device.

Details

The HPOS= graphics option overrides the values of the LCOLS or PCOLS device parameters and temporarily sets the number of columns in the graphics output area. HPOS= does not affect the width of the graphics output area but merely divides it into columns. Therefore, you can use HPOS= to control cell width.

The values specified in the HPOS= and VPOS= graphics options determine the size of a character cell for the graphics output area and consequently the size of many graphics elements, such as hardware text. The larger the size of the HPOS= and VPOS= values, the smaller the size of each character cell.

See “Procedure Output and the Graphics Output Area” on page 34 for more information.

HSIZE

Sets the horizontal size of the graphics output area.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Detail window

Restriction: partially supported by Java or ActiveX

See also: VSIZE, XMAX

Syntax

H_{SIZE}=*horizontal-size* <IN | CM | PT>

horizontal-size <IN | CM | PT>

specifies the width of the graphics output area; *horizontal-size* must be a positive number and may be followed by a unit specification, either IN for inches (default), or CM for centimeters, or PT for points.

If you do not specify H_{SIZE}=, a default size is searched for in this order:

- 1 the horizontal size is calculated as

$$\text{XMAX} - \text{LEFTMARGIN} - \text{RIGHTMARGIN}$$

Note that LEFTMARGIN and RIGHTMARGIN are used in the OPTIONS statement.

- 2 H_{SIZE} setting in the device catalog.

Featured in: “Example 3. Rotating Plot Symbols through the Colors List” on page 231

HTEXT

Specifies the default height of the text in the graphics output.

Used in: GOPTIONS statement

Default: 1 cell

Restriction: partially supported by Java

Syntax

H_{TEXT}=*text-height* <*units*>

text-height <*units*>

specifies the height of the text; by default *text-height* is 1. For a description of *units*, see “Specifying Units of Measurement” on page 262.

Note: If a value for *units* is not specified, the current units associated with the GUNIT graphics option are used. \triangle

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226

Details

H_{TEXT}= is overridden by the HTITLE= graphics option for the *first* TITLE line.

Note: When you use ODS to send graphics to an HTML destination, and titles and footnotes are rendered as part of the HTML body file instead of the graphic image, you must specify the ODS USEGOPT statement for this option to work. See “Controlling Titles and Footnotes with ODS Output” on page 492 for more information. △

HTITLE

Selects the default height used for the first TITLE line.

Used in: GOPTIONS statement

Default: 2 cells unless HTEXT= is used

Syntax

HTITLE=*title-height* <*units*>

title-height <*units*>

specifies the height of the text in the TITLE1 statement. By default, *title-height* is 2. For a description of *units*, see “Specifying Units of Measurement” on page 262.

Note: If a value for *units* is not specified, the current units associated with the GUNIT graphics option are used. △

Featured in: “Example 1. Ordering Axis Tick Marks with SAS Datetime Values” on page 226

Details

If you omit HTITLE=, TITLE1 uses the height specified by the HTEXT= graphics option, if used.

Note: When you use ODS to send graphics to an HTML destination, and titles and footnotes are rendered as part of the HTML body file instead of the graphic image, you must specify the ODS USEGOPT statement for this option to work. See “Controlling Titles and Footnotes with ODS Output” on page 492 for more information. △

IBACK

Specifies an image file to display in a graph’s background area.

Restriction: partially supported by Java

See also: CBACK, IMAGESTYLE

Syntax

IBACK=*fileref* | '*external-file*' | '*URL*'

fileref

specifies a fileref that points to the image file you want to use. *Fileref* must be a valid SAS fileref up to eight characters long and must have been previously assigned with a FILENAME statement.

external-file

specifies the complete file name of the image file you want to use. The format of external-file varies across operating environments.

URL

specifies the URL of the image file that you want to use.

Details

The image can be used with any procedures that produce a picture or support the CBACK= option. The IBACK option is supported by the Graph applet and the Map applet, but it is not supported by the Contour applet. See Chapter 9, “Introducing SAS/GRAPH Output for the Web,” on page 369 for information about these applets.

This option overrides the BackgroundImage and Image styles attribute in the graph styles. For more information on graph styles, refer to the TEMPLATE procedure documentation in *SAS Output Delivery System: User’s Guide*.

ID

Specifies the description string used by the Metagraphics driver.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Restriction: Used only with user-supplied Metagraphics drivers.

Syntax

ID=*description*'

description

is a character string up to 70 characters long. If this field is blank, the name and description of the graph as specified in the PROC GREPLAY window of the GREPLAY procedure are used.

Details

For information about Metagraphics drivers, contact Technical Support.

IMAGEPRINT

Enables or disables image output

Used in: GOPTIONS statement

Default: IMAGEPRINT

Restriction: not supported by Java or ActiveX

Syntax

IMAGEPRINT | NOIMAGEPRINT

IMAGEPRINT

default value specifies that any images are to be included in graphics output.

NOIMAGEPRINT

specifies that images are to be withheld from graphics output.

IMAGESTYLE

Specifies the way to display the image file that is specified on the IBACK= option.

Default: TILE

Restriction: not supported by Java

Syntax

IMAGESTYLE= TILE | FIT

TILE

tile the image within the specified area. This copies the images as many times as needed to fit the area.

FIT

fit the image within the background area. This stretches the image, if necessary.

Details

Note: This option overrides the BackgroundImage and Image styles attribute in the graph styles. For more information on graph styles, refer to the TEMPLATE procedure documentation in *SAS Output Delivery System: User's Guide*. △

INTERACTIVE

Sets level of interactivity for Metagraphics driver.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Default: USER

Restriction: Used only with user-supplied Metagraphics drivers.

Syntax

INTERACTIVE=USER | GRAPH | PROC

USER

specifies that the user-written part of the driver be executed outside of SAS/GRAPH.

PROC

specifies that the user-written part of the Metagraphics driver be invoked after the procedure is complete.

GRAPH

specifies that the user-written part be invoked for each graph.

Details

For information about Metagraphics drivers, contact Technical Support.

INTERLACED

Specifies whether images are to be displayed as they are received in the browser.

Used in: GOPTIONS statement

Default: NONINTERLACED

Restriction: driver dependent, GIF series of drivers only

Syntax

INTERLACED | NONINTERLACED

Details

With interlacing it is possible to get a rough picture of what a large image will look like before it is completely drawn in your browser. Your browser may allow you to set an option that will determine how images are displayed.

INTERPOL

Sets the default interpolation value for the SYMBOL statement.

Used in: GOPTIONS statement

Restriction: not supported by Java or ActiveX

Syntax

INTERPOL=*interpolation-method*

interpolation-method

specifies the default interpolation to be used when the INTERPOL= option is not specified in the SYMBOL statement. See “SYMBOL Statement” on page 183 for the complete syntax of all interpolation methods.

ITERATION

Specifies the number of times to repeat the animation loop.

Used in: GOPTIONS statement

Default: 0

Restriction: GIFANIM driver only

Syntax

ITERATION=*iteration-count*

iteration-count

specifies the number of times that your complete GIF animation loop is repeated. It is assumed that the animation is always played once; this option specifies how many times the animation is repeated. *Iteration-count* can be a number from 0...65535. A value of 0 causes the animation to loop continuously.

Details

In Version 6, the GCOPIES graphics option controlled iteration for the GIFANIM driver.

KEYMAP

Selects the keymap to use.

Used in: GOPTIONS statement

Default: installation dependent

Restriction: not supported by Java or ActiveX

Syntax

KEYMAP=*key-map-name* | NONE

key-map-name

specifies the name of a keymap. See Chapter 34, “The GKEYMAP Procedure,” on page 983 for details.

NONE

suppresses the keymap assigned by default to a non-U.S. keyboard. If you specify `KEYMAP=NONE`, text may display incorrectly or not at all.

Details

Non-default key maps usually are used only with non-U.S. keyboards.

LCOLS

Sets the number of columns in the graphics output area for landscape orientation.

Used in: GDEVICE procedure; GDEVICE Detail window

Default: device dependent

See also: HPOS, LROWS, PCOLS

Syntax

`LCOLS=landscape-columns`

landscape-columns

must be a nonnegative integer up to three digits long (0...999).

Details

Either the LROWS and LCOLS pair of device parameters or the PROWS and PCOLS pair of device parameters are required and must be nonzero.

The HPOS= graphics option overrides the value of LCOLS.

See “Procedure Output and the Graphics Output Area” on page 34 for more information.

LFACTOR

Selects the default hardware line thickness.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Restriction: Used only with devices that can draw hardware lines of varying thicknesses. Not supported by Java or ActiveX.

Syntax

`LFACTOR=line-thickness-factor`

line-thickness-factor

can range from 0 through 9999. A value of 0 for LFACTOR is the same as a factor of 1. Lines are drawn *line-thickness-factor* times as thick as normal.

Details

LFACTOR is useful when you are printing graphics output on a plotter. Depending on the orientation and type of device, some plotters may require LFACTOR=10 to get the same thickness of lines as on the display of some devices.

LROWS

Sets the number of rows in the graphics output area for landscape orientation.

Used in: GDEVICE procedure; GDEVICE Detail window

Default: device dependent

See also: LCOLS, PROWS, VPOS

Syntax

LROWS=*landscape-rows*

landscape-rows

is a nonnegative integer up to three digits long (0...999).

Details

Either the LROWS and LCOLS pair of device parameters or the PROWS and PCOLS pair of device parameters are required and must be nonzero.

The VPOS= graphics option overrides the value of LROWS.

See "Procedure Output and the Graphics Output Area" on page 34 for more information.

MAXCOLORS

Sets the total number of colors that can be displayed at once.

Used in: GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

See also: PENMOUNTS

Syntax

MAXCOLORS=*number-of-colors*

number-of-colors

must be an integer in the range 2 through 256. The total number of colors includes the foreground colors plus the background color.

Details

The PENMOUNTS= graphics option overrides the value of MAXCOLORS.

MAXPOLY

Sets the maximum number of vertices for hardware-drawn polygons.

Used in: GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Syntax

MAXPOLY=*number-of-vertices*

number-of-vertices

is a nonnegative integer up to four digits long. A value of 0 means that there is no limit to the number of vertices that can be specified in the hardware's polygon-drawing command. The maximum value of MAXPOLY depends on the number of vertices your device can process.

MODEL

Specifies the model number of the output device.

Used in: GDEVICE procedure; GDEVICE Detail window

Default: device dependent

Syntax

MODEL=*model-number*

model-number

is a nonnegative integer up to five digits long that is the Institute-designated model number for the corresponding device. It is not the same as a manufacturer's model number.

Details

Do not change this field in Institute-supplied drivers or in drivers that you copy from Institute-supplied drivers.

MODULE

Specifies the name of the corresponding executable driver module for the device.

Used in: GDEVICE procedure; GDEVICE Detail window

Default: device dependent

Syntax

MODULE=*driver-module*

driver-module

is a literal string up to eight characters long. All standard driver modules begin with the characters SASGD.

Details

Do not change this field in Institute-supplied drivers or in drivers that you copy from Institute-supplied drivers.

NAK

Specifies the negative response for software handshaking for Metagraphics drivers.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Restriction: Used only with user-supplied Metagraphics drivers.

Syntax

NAK=*'negative-handshake-response'*X

negative-handshake-response

is a hexadecimal string up to 16 characters long.

Details

For information about Metagraphics drivers, contact Technical Support.

OFFSHADOW

Controls the width and depth of the drop shadow in legend frames.

Used in: GOPTIONS statement

Default: (0.0625, – 0.0625) IN

Restriction: not supported by Java or ActiveX

Syntax

OFFSHADOW=(*x* <*units*>, *y* <*units*>) | (*x,y*) <*units*>

x,y

specify the width (*x*) and depth (*y*) of the drop shadow generated by the LEGEND statement.

If a value for *units* is not specified, the current units associated with the GUNIT graphics option are used. For a description of *units*, see “Specifying Units of Measurement” on page 262.

Details

The values specified by OFFSHADOW= are used with the CSHADOW= and CBLOCK= options in a LEGEND statement. For details, see “LEGEND Statement” on page 151.

PAPERDEST

Specifies which output bin the printer should use if multiple bins are available on the device.

Used in: GOPTIONS statement; OPTIONS statement

Default: 1 (the upper output bin)

Restrictions: hardware dependent, PostScript printers require a PPD file; not supported by Java or ActiveX

See also: PAPERSOURCE, PPDFILE

Syntax

PAPERDEST=*bin*

bin

specifies the name or number of the output bin. Values for *bin* depend on the type of printer and can be one of the following:

bin the name or number of the output bin – for example, PAPERDEST=4, PAPERDEST=BIN2, PAPERDEST=SIDE

'long bin name' a character string that is the name of the output bin – for example, PAPERDEST='Top Output Bin'. Names with blanks or special characters must be quoted.

For PostScript printers, the value for *bin* must correspond to an OutputBin value in the PPD file.

For PCL printers, consult the printer's documentation for valid bin values. If a numeric value exceeds the maximum bin value allowed for the printer, a warning

message is issued . For string values, the string is checked against a list of strings that are valid for the driver (for example, 'UPPER', 'LOWER', or 'OPTIONALOUTBIN n ', where n is the bin number). If the string is not valid for the driver, a warning message is issued.

PAPERFEED

Specifies the increment of paper that is ejected when a graph is completed.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Detail window

Default: PAPERFEED=0.0 IN

Restriction: device dependent; not supported by Java or ActiveX

Syntax

PAPERFEED=*feed-increment* <IN | CM>

feed-increment <IN | CM>

must be a nonnegative number and may be followed by a unit specification, either IN for inches (default) or CM for centimeters.

Details

PAPERFEED does not control the total length of the ejection. If you specify PAPERFEED=1, the driver ejects paper in 1-inch increments until the total amount of paper ejected is at least half an inch greater than the size of the graph last printed. If you specify PAPERFEED=8.5 IN, the paper is ejected in increments of 8.5 inches, measuring from the origin of the first graph.

PAPERFEED is provided mainly for plotters that use fanfold or roll paper. If you are using fanfold paper, specify a value for PAPERFEED that is equal to the distance between the perforations.

PAPERLIMIT

Sets the width of the paper used with plotters.

Used in: GOPTIONS statement

Default: maximum dimensions specified in the device driver

Restriction: ZETA plotters and KMW rasterizers

Syntax

PAPERLIMIT=*width* <IN | CM>

width <IN | CM>

specifies the paper width in IN for inches (default) or CM for centimeters. If PAPERLIMIT= is not specified, the maximum dimensions of the graph are restricted by the hardware limits of the graphics device.

Details

If you want to use a driver with a device that has a larger plotting area than the device for which the driver is intended (for example, using the ZETA887 driver with a ZETA 836 plotter), the PAPERLIMIT= graphics option can be used to override the size limit of the driver.

PAPER SIZE

Specifies the name of a paper size.

Used in: GOPTIONS statement; OPTIONS statement

Default: device dependent

Restriction: hardware dependent, PostScript printers require a PPD file; not supported by Java or ActiveX

See also: PAPERSOURCE, PPDFILE

Syntax

PAPER SIZE='size-name'

size-name

specifies the name of a paper size, such as LETTER, LEGAL, or A4.

If you do not specify PAPER SIZE=, the PAPER SIZE= setting on an OPTIONS statement is used. If no OPTIONS statement sets a paper size, the value for paper size is device dependent:

- The universal printing devices use the size specified in the Page Setup dialog.
- All other printer devices use the LETTER paper size.

Details

Typically, you might use PAPER SIZE= with the Output Delivery System (ODS). For some printers, PAPER SIZE= overrides the PAPERSOURCE= selection.

For PostScript devices, the name must match the name of a paper size in the PPD file. Refer to the PPD file for a list of valid names. *Size-name* is case-insensitive and can contain a subset of the full name. For example, if the name in the PPD file is *Page Size A4/A4, you can specify PAPER SIZE='A4'. If a PPD file is not specified, PAPER SIZE= is ignored.

For PCL devices, the device driver searches the SAS Registry for supported paper size values. To see the supported list of sizes, submit the following statements:

```
proc registry listhelp
    startat='options\papersize';
run;
```

For more information about the SAS Registry, refer to the SAS Help facility.

PAPERSOURCE

Specifies which paper tray the printer should use if multiple trays are available on the device.

Used in: GOPTIONS statement; OPTIONS statement

Default: device dependent

Restriction: hardware dependent, PostScript printers require a PPD file; not supported by Java or ActiveX

See also: PAPERDEST, PAPERSIZE, PPDFILE

Syntax

PAPERSOURCE=*tray*

tray

specifies the name or number of the paper tray. Values for *tray* depend on the type of printer and can be one of the following:

tray the name or number of the paper tray, for example,
PAPERSOURCE=3, PAPERSOURCE=TRAY3,
PAPERSOURCE=Upper

'*long tray name*' a character string that is the name of the paper tray, for example,
PAPERSOURCE='Optional Output Tray'. Names with blanks or
special characters must be quoted.

Details

On some printers, if PAPERSIZE= is also specified, it overrides the setting on PAPERSOURCE=.

For PostScript printers, a tray number, such as PAPERSOURCE='tray3', must correspond to an InputSlot value in the PPD file.

For PCL printers, consult the printer's documentation for valid tray values. If a numeric value exceeds the maximum tray value allowed for the printer, a warning message is issued. For string values, the string is checked against a list of strings that are valid for the driver:

- 'AUTO'
- 'HCP' or 'HCPn', where *n* is a number from 2 to 21
- 'MANUAL'
- 'MANUAL_ENVELOPE'
- 'TRAYn', where *n* is 1, 2, or 3.

If the string is not valid for the driver, a warning message is issued.

PAPERTYPE

Specifies the name of a paper type.

Used in: GOPTIONS statement; OPTIONS statement

Default: PLAIN

Restriction: hardware dependent, PostScript printers require a PPD file; not supported by Java or ActiveX

See also: PPDFILE

Syntax

PAPERTYPE=*'type-name'*

type-name

specifies the name of a paper type. Valid values depend on the type of printer.

For PostScript devices, *type-name* must match the name of a paper type in the PPD file, such as TRANSPARENCY or PLAIN. Refer to the PPD file for a list of valid names. *Type-name* is case-insensitive and can contain a subset of the full name. For example, if the name in the PPD file is *MediaType Plain/Paper you can specify PAPERTYPE='PLAIN/PAPER'.

For PCL devices, *type-name* specifies the name of a paper type that is available on the current printer, such as GLOSSY, PLAIN, SPECIAL, or TRANSPARENCY. Consult your printer's user manual for the complete list of available paper types on your printer.

Details

For PostScript devices, if a PPD file is not specified, PAPERTYPE= is ignored.

PATH

Sets the increment of the angle for hardware text rotation.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Default: PATH=0

Restriction: Used only with user-supplied Metagraphics drivers.

Syntax

PATH=*angle-increment*

angle-increment

is an integer in the range 0 to 360 that specifies the angle at which to rotate the text baseline. A value of 0 means that the device uses its default orientation. Specify 0 if your device does not perform string angling in hardware.

Details

For information about Metagraphics drivers, contact Technical Support.

PCLIP

Specifies whether a clipped polygon is stored in its clipped or unclipped form.

Used in: GOPTIONS statement

Default: NOPCLIP

Restriction: not supported by Java or ActiveX

See also: POLYGONCLIP

Syntax

PCLIP |NOPCLIP

PCLIP

stores clipped polygons with the graph in the default catalog WORK.GSEG, or in the catalog you specify.

NOPCLIP

stores the unclipped form of the polygon and causes the polygon to be clipped when replayed.

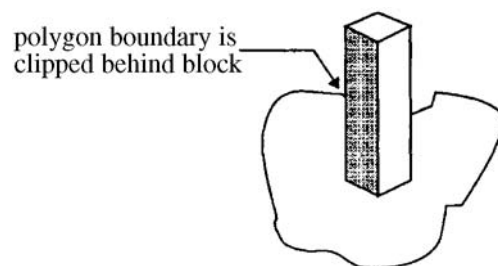
Details

The effects of this option are only seen when you use the graphics editor to edit a graph.

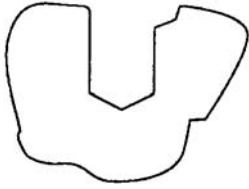
When a procedure produces a graph with intersecting polygons or blanking areas, it clips portions of the polygons to prevent the ones behind from showing through. When the graph is created and stored in a catalog, if PCLIP is in effect, the clipped form of the polygon is stored with it. If NOPCLIP is specified, the complete polygon is stored in the catalog and the graph is clipped each time it is replayed.

For example, suppose you create a block map like the one in Figure 8.1 on page 331.

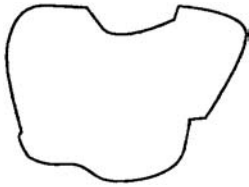
Figure 8.1 Intersecting Polygons



The block clips the boundary of the map area polygon. If you specify PCLIP, the map area polygon is stored in its clipped form, as shown in Figure 8.2 on page 332.

Figure 8.2 Clipped Polygon with PCLIP Option

NOPClip stores the map area in its unclipped form, as shown in Figure 8.3 on page 332.

Figure 8.3 Polygon with NOPCLIP Option

In this case, when the graph is recalled from the catalog, the map area polygon must be clipped before it is displayed with the block. If you plan to edit the graph with the graphics editor, specify NOPCLIP so polygons retain their original form.

PCOLS

Sets the number of columns in the graphics output area for portrait orientation.

Used in: GDEVICE procedure; GDEVICE Detail window

Default: device dependent

See also: HPOS, LCOLS, PROWS

Syntax

PCOLS=*portrait-columns*

portrait-columns

must be a nonnegative integer up to three digits long (0...999).

Details

Either the LROWS and LCOLS pair of device parameters or the PROWS and PCOLS pair of device parameters are required and must be nonzero.

The HPOS= graphics option overrides the value of PCOLS.

See “Procedure Output and the Graphics Output Area” on page 34 for more information.

PENMOUNTS

Specifies the number of active pens or colors.

Used in: GOPTIONS statement

Default: device dependent

Restriction: not supported by Java or ActiveX

See also: MAXCOLORS

Syntax

PENMOUNTS=*active-pen-mounts*

active-pen-mounts

specifies the number of pens for a plotter with multiple pens. After the specified number of pens have been used, you are prompted to change the pens.

Details

For devices that are not pen plotters, PENMOUNTS= can be used to indicate the number of colors that can be displayed at one time. In this case, PENMOUNTS= performs the same function as the MAXCOLORS device parameter except that the value specified for MAXCOLORS includes the background color and PENMOUNTS only refers to foreground colors. Thus, PENMOUNTS=4 implies MAXCOLORS=5.

PENMOUNTS= overrides the value of the MAXCOLORS device parameter. You can specify MAXCOLORS= in a GOPTIONS statement as a synonym for PENMOUNTS=.

PENSORT

Specifies whether plotters draw graphics elements in order of color.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Restriction: not supported by Java or ActiveX

Default: device dependent

Syntax

GOPTIONS: PENSORT | NOPENSORT

GDEVICE: PENSORT=Y | N

PENSORT

PENSORT=Y

causes the plotter to draw all graphics elements of one color at one time. For example, it draws all the red elements in the output, then all the blue elements, and

so forth. This specification is compatible with previous releases. Use it for plotters with real pens.

NOPENSORT

PENSORT=N

causes the plotter to draw each element as it is encountered, regardless of its color. For example, the plotter might draw a red circle, then a blue line, and then a red line, and so forth. This method is best for electrostatic printers implemented with Metagraphics drivers of TYPE=PLOTTER. In addition, NOPENSORT enables you to specify non-standard color names.

PIEFILL

Specifies whether to use the device's hardware pie-fill capability.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Restriction: not supported by Java or ActiveX

Syntax

GOPTIONS: PIEFILL | NOPIEFILL

GDEVICE: PIEFILL=Y | N

PIEFILL

PIEFILL=Y

causes SAS/GRAPH to use the built-in hardware capability of the device, if available, to fill pies and pie sections. A blank **piefill** field in the Parameters window is the same as PIEFILL=Y.

Hardware drawing is faster, but not all devices have the capability. SAS/GRAPH does not try to use the capability if your device does not support it.

NOPIEFILL

PIEFILL=N

causes SAS/GRAPH to fill pies and pie sections using software pie fills.

POLYGONCLIP

Specifies the type of clipping used when two polygons overlap.

Used in: GOPTIONS statement

Default: device dependent

Restriction: not supported by Java or ActiveX

See also: PCLIP

Syntax

POLYGONCLIP | NOPOLYGONCLIP

POLYGONCLIP

specifies polygon clipping, which enables a clipped polygon to be filled with a hardware pattern. POLYGONCLIP affects only graphs that have blanking areas or intersecting polygons.

NOPOLYGONCLIP

specifies line clipping; a polygon that has been line-clipped cannot use a hardware pattern.

Details

Clipping is the process of removing part of one polygon when two polygons intersect. For example, in a block map, a block may overlap the boundary of its map area. In this case, the polygon that makes up the map area is clipped so that you do not see the boundary line behind the block. (See Figure 8.1 on page 331 for an illustration of a clipped polygon.) The type of clipping used by a graph affects whether a clipped area can use hardware patterns.

POLYGONCLIP is affected by the PCLIP graphics option:

POLYGONCLIP with PCLIP or NOPCLIP

all areas can use hardware patterns

NOPOLYGONCLIP with NOPCLIP

all areas use only software patterns

NOPOLYGONCLIP with PCLIP

areas may use either hardware or software patterns depending on the nature of the clipped polygons.

Under some conditions the polygons may not be clipped correctly. Specifying both POLYGONCLIP and NOPCLIP will produce the correct graph.

POLYGONFILL

Specifies whether to use the device's hardware polygon-fill capability.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Restriction: not supported by Java or ActiveX

Syntax

GOPTIONS: POLYGONFILL | NOPOLYGONFILL

GDEVICE: POLYFILL=Y | N

POLYGONFILL**POLYFILL=Y**

causes SAS/GRAPH to use the built-in hardware capability of the device to fill polygons. A blank **polyfill** field in the Parameters window is the same as POLYGONFILL.

Hardware drawing is faster, but not all devices have the capability. SAS/GRAPH does not try to use the capability if your device does not support it.

NOPOLYGONFILL**POLYFILL=N**

causes SAS/GRAPH to use software fills to fill polygons.

POSTGEPILOG

Specifies data to send immediately after the data that is stored in the Gepilog field of the device entry is sent.

Used in: GOPTIONS statement

Default: Null string

Restriction: not supported by Java or ActiveX

See also: GEPILOG, PREGGEPILOG

Syntax

POSTGEPILOG=*'string'*

'string'

can be either of the following:

*'hex-string'*X

'character-string'

PROC GOPTIONS always reports the value as a hexadecimal string.

POSTGPROLOG

Specifies the data to send immediately after the data that is stored in the Gprolog field of the device entry is sent.

Used in: GOPTIONS statement

Default: Null string

Restriction: not supported by Java or ActiveX

See also: GPROLOG, PREGPROLOG

Syntax

POSTGPROLOG=*'string'*

'string'

can be either of the following:

*'hex-string'*X

'character-string'

PROC GOPTIONS always reports the value as a hexadecimal string.

POSTGRAPH

Specifies host commands to be executed after the graph is produced.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host Commands window

Restriction: not supported by Java or ActiveX

See also: FILECLOSE

Syntax

POSTGRAPH1=*'system-command(s)'*

POSTGRAPH2=*'system-command(s)'*

system-command(s)

specifies one or more valid system commands. The string can contain upper- or lowercase characters. Separate multiple commands with a command delimiter, which is host-specific; for example, some operating environments use a semicolon. The total length of the string cannot exceed 72 characters. The commands are executed right after the graph is produced.

Details

If you want to use a host command to send output to the device after each graph executes, use the POSTGRAPH parameter with FILECLOSE=GRAPHEND.

PPDFILE

Specifies the location of an external file containing PostScript Printer Description (PPD) information.

Used in: GOPTIONS statement

Restriction: PostScript printers only

See also: BINDING, COLLATE, DUPLEX, PAPERDEST, PAPERSIZE, PAPERSOURCE, PAPERTYPE, REVERSE

Syntax

PPDFILE=*fileref* | '*external-file*'

fileref

specifies a fileref that points to the PPD file you want to use. *Fileref* must be a valid SAS fileref up to eight characters long and must have been previously assigned with a FILENAME statement. See “FILENAME Statement” on page 28 for additional information.

external-file

specifies the complete filename of the PPD file you want to use. The format of *external-file* varies across operating environments. For details, see the SAS documentation for your operating environment.

Details

A PostScript Printer Description (PPD) file is a text file that contains commands required to access features of the device. These files are available from Adobe. Also, many printer manufacturers provide the appropriate PPD file for their PostScript printers.

PREGEPILOG

Specifies data to send immediately before the data that is stored in the Gepilog field of the device entry is sent.

Used in: GOPTIONS statement

Default: Null string

Restriction: not supported by Java or ActiveX

See also: GEPILOG, POSTGEPILOG

Syntax

PREGEPILOG=*'string'*

'string'

can be either of the following:

*'hex-string'*X

'character-string'

PROC GOPTIONS always reports the value as a hexadecimal string.

PREGPROLOG

Specifies the data to send immediately before the data that is stored in the Gprolog field of the device entry is sent.

Used in: GOPTIONS statement

Default: Null string

Restriction: not supported by Java or ActiveX

See also: GPROLOG, POSTGPROLOG

Syntax

PREGPROLOG=*'string'*

'string'

can be either of the following:

*'hex-string'*X

'character-string'

PROC GOPTIONS always reports the value as a hexadecimal string.

PREGRAPH

Specifies host commands to be executed before the graph is produced.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host Commands window

Restriction: not supported by Java or ActiveX

See also: FILECLOSE

Syntax

PREGRAPH1=*'system-command(s)'*

PREGRAPH2=*'system-command(s)'*

system-command(s)

specifies one or more valid system commands. The string can contain upper- or lowercase characters. Separate multiple commands with a command delimiter, which is host-specific; for example, some operating environments use a semicolon. The total length of the string cannot exceed 72 characters. The commands are executed immediately before the graph is produced.

Details

The PREGRAPH parameter should be used with FILECLOSE=GRAPHEND.

PROCESS

Specifies the command that translates the metafile into commands for the device.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Restriction: Used only with user-supplied Metagraphics drivers.

See also: INTERACTIVE

Syntax

PROCESS=*command*'

command

specifies the command that translates the metafile produced by the Metagraphics driver into commands for the device. The command runs your program to produce the output. *Command* is a string up to 40 characters long.

Details

PROCESS is required if the value of the INTERACTIVE device parameter is PROC or GRAPH.

For information about Metagraphics drivers, contact Technical Support.

PROCESSINPUT

Specifies the fileref for the file that contains input for the user-written part of the Metagraphics driver.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Restriction: Used only with user-supplied Metagraphics drivers.

Syntax

PROCESSINPUT=*fileref*

fileref

specifies a valid SAS fileref up to eight characters long. *Fileref* must be assigned with a FILENAME statement or a host command before running the Metagraphics driver. See “FILENAME Statement” on page 28 for additional information.

Details

For information about Metagraphics drivers, contact Technical Support.

PROCESSOUTPUT

Specifies the fileref for the file that receives output from the user-written part of the Metagraphics driver.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Restriction: Used only with user-supplied Metagraphics drivers.

Syntax

PROCESSOUTPUT=*fileref*

fileref

specifies a valid SAS fileref up to eight characters long. *Fileref* must be assigned with a FILENAME statement or a host command before running the Metagraphics driver. See “FILENAME Statement” on page 28 for additional information.

Details

For information about Metagraphics drivers, contact Technical Support.

PROMPT

Specifies whether prompts are issued.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Restriction: not supported by Java or ActiveX

Default: device dependent

Syntax

GOPTIONS: PROMPT | NOPROMPT

GDEVICE: PROMPT=0...7

PROMPT

causes all prompts to be displayed.

NOPROMPT

suppresses all prompts. NOPROMPT overrides the GWAIT= graphics option.

PROMPT=0...7

in the GDEVICE procedure, specifies the level of prompting:

0	provides no prompting
1	issues startup messages only. Startup messages are messages such as PLEASE PRESS RETURN TO CONTINUE.
2	signals end of graph if device is a video display or sends message to change paper if device is a plotter.
3	combines the effects of 1 and 2.
4	sends a message to mount pens if the device is a plotter.
5	combines the effects of 4 and 1.
6	combines the effects of 4 and 2.
7	sends all prom

Note: If you specify either 0 for the PROMPT device parameter or NOPROMPT in a GOPTIONS statement for a display device, the display clears immediately after the graph is drawn. △

In the GDEVICE Parameters window, the PROMPT parameter consists of four fields that describe the type of prompt:

start up

issues a message to turn the device on (if the device is a hardcopy device) or the message PLEASE PRESS RETURN AFTER EACH BELL TO CONTINUE.

end of graph

signals, usually by a bell, when the graph is complete (valid for video displays only).

mount pens

issues a message to mount pens in a certain order and (for certain devices only) to ask for pen priming strokes for plotters.

change paper

prompts the user to change the paper (valid for plotters only).

Enter an X for each prompt that you want to be given. If no Xs appear in these fields, no prompt messages are issued, and the device does not wait for you to respond between graphs.

PROMPTCHARS

Selects the prompt characters to be used by SAS/GRAPH device drivers.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: host dependent

Restriction: not supported by Java or ActiveX

See also: GSFLN, HANDSHAKE

Syntax

PROMPTCHARS=*'prompt-chars-hex-string'*X

prompt-chars-hex-string

is an 8-byte hexadecimal string that is specified as 16 hexadecimal characters. In GDEVICE procedure statements, enclose the string in single quotation marks, followed by an X. In the Parameters window, enter the hexadecimal string without either quotation marks or a trailing X.

Note: Bytes 1, 4, and 5 are the safest for you to change because you are most likely to know the correct value for them. Check with Technical Support before changing any of the other bytes. △

The following list describes each byte in the string:

byte 1

is the ASCII code of the system prompt character (for software handshaking). The system prompt character is the last character that the host sends before waiting for a response from the plotter. For example, 11 means the host sends an XON or DC1 character as a prompt. If the host does not send a special character for a prompt, set this byte to 00.

byte 2

is the ASCII code of the echo-terminator character (for software handshaking). This character is sent at the beginning of each record.

byte 3

prevents splitting commands across records if the value is 01. If you are creating a graphics stream file to send to a device at a later time, and there is the possibility that extra characters will be added between records during transmission, setting the third byte to 01 reduces the likelihood that the extra characters will be interpreted as graphics commands and cause stray lines or other device characters. If the third byte is set to 00, the driver makes the records as long as possible and splits device commands across records if necessary. Setting the third byte to 00 is more efficient but is more likely to result in device errors if output is written to a file and later transmitted to the device.

byte 4

is the line-end character (for software handshaking). It indicates that more data can be sent. This character is almost always a carriage-return character, 0D.

byte 5

specifies turnaround delay in tenths of a second (for software handshaking). The turnaround delay is the amount of time the device waits after receiving the prompt character before sending the line-end character. For example, a value of 05 represents a half-second delay.

byte 6

sets default record length using a hexadecimal value 00–FF. This byte sets the length of the records sent to the device or to a file. If this byte is set to 00 (the default), SAS/GRAPH uses the longest record length possible for the device. To specify an alternate length, set the sixth byte to the hexadecimal value for the desired length. For example, to generate records of length 80, specify 50 for the sixth byte. If the GSFLEN device parameter or graphics option is specified, its value overrides the value of the sixth prompt character.

Some values of the GPROTOCOL device parameter cause each byte in the data stream to be expanded to two bytes. This expansion is done after the length of the record is set by PROMPTCHARS. If you are specifying a value for GPROTOCOL that does this (for example, SASGPHEX, SASGPLCL, or SASGPAGL), specify a value for the sixth byte of PROMPTCHARS that is half of the actual record length desired. For example, a hexadecimal value of 40 (64 decimal) produces a 128-byte record after expansion by the GPROTOCOL module.

bytes 7 and 8
are unused and should be set to 0000.

Details

PROMPTCHARS is most commonly used to specify parameters used in software handshaking (see “HANDSHAKE” on page 311), but it also can be used to control the length of records written by most drivers. You also can use the GSFLLEN= graphics option for this purpose.

PROWS

Sets the number of rows in the graphics output area for portrait orientation.

Used in: GDEVICE procedure; GDEVICE Detail window

Default: device dependent

See also: LROWS, PCOLS, VPOS

Syntax

PROWS=*portrait-rows*

portrait-rows

is a nonnegative integer up to three digits long (0...999).

Details

Either the LROWS and LCOLS pair of device parameters or the PROWS and PCOLS pair of device parameters are required and must be nonzero.

The VPOS= graphics option overrides the value of PROWS.

See “Procedure Output and the Graphics Output Area” on page 34 for more information.

QMSG

Specifies whether log messages are held until after the graphics output is displayed.

Used in: GDEVICE procedure; GDEVICE Detail window

Default: device dependent

Syntax

GOPTIONS: QMSG | NOQMSG

GDEVICE: QMSG=Y | N

QMSG QMSG=Y

queues driver messages while the device is in graphics mode (default for video devices).

NOQMSG QMSG=N

prevents the queuing of messages (default for plotters, cameras, and printers).

Details

Message queuing is desirable on display devices that do not have a separate dialog and graphics area. If messages are not queued, they are written to the log as the graphics output is being generated. This behavior may cause problems on some devices.

A blank **Queued messages** field in the Parameters window can mean either Y or N, depending on the device.

RECTFILL

Specifies which rectangle fills should be performed by hardware.

Used in: GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

See also: FILL

Syntax

RECTFILL=*'rectangle-fill-hex-string'*X

rectangle-fill-hex-string

is a hexadecimal string that is 16 characters long. In GDEVICE procedure statements, enclose the string in single quotation marks, followed by an X. In the Parameters window, enter the hexadecimal string without either quotation marks or a trailing X.

The following table shows which bit position (left-to-right) within the hexadecimal string controls each fill pattern.

Bit	Fill pattern	Bit	Fill pattern
1	R1	9	L4
2	R2	10	L5
3	R3	11	X1

Bit	Fill pattern	Bit	Fill pattern
4	R4	12	X2
5	R5	13	X3
6	L1	14	X4
7	L2	15	X5
8	L3	16	S

For example, if you want the driver to use only the L1 and R1 fills in hardware, the first and sixth bits of the first byte of the hexadecimal string should be turned on, which corresponds to a value of '84000000000000'X ('84'X is equivalent to '1 0 0 0 0 1 0 0' in binary). If a particular hardware rectangle fill is not available or not to be used (as indicated by the value of RECTFILL), the fill is generated by the software.

See "PATTERN Statement" on page 169 for an illustration of the fill patterns.

Details

Note: Not all devices support this capability. If FILL=N is specified or the NOFILL option is used in a GOPTIONS statement, RECTFILL is ignored. \triangle

RENDER

Controls the creation and disposition of rendered Bitstream fonts.

Used in: GOPTIONS statement

Default: MEMORY

Restriction: not supported by Java or ActiveX

See also: RENDERLIB

Syntax

RENDER=APPEND | DISK | MEMORY | NONE | READ

APPEND

creates files to store rendered versions of Bitstream fonts if the files do not already exist, reads previously rendered characters from the font files, and appends rendered versions of new characters to the font files when the SAS/GRAPH procedure terminates.

DISK

creates files to store rendered versions of Bitstream fonts if the files do not already exist, reads previously rendered characters from the font files, and appends rendered versions of new characters to the font files as they are encountered. This method is slower on some hosts, but it may work in memory-constrained conditions where the other rendering methods fail.

MEMORY

renders all fonts in memory without creating any font files on disk. Font files are not used even if they already exist. New characters are not written to existing font files when SAS/GRAPH procedures terminate.

This is the default and should be the fastest method on hosts that support virtual memory.

NONE

disables the font rendering features.

READ

reads existing rendered font files but does not create new font files or write new characters to existing font files. This is useful only when font files already exist in the rendered font library.

Details

The memory capacity and input/output characteristics of your host system determine which value for the RENDER= option provides the best performance.

RENDERLIB

Specifies the SAS data library in which rendered font files are stored.

Used in: GOPTIONS statement

Default: WORK

Restriction: not supported by Java or ActiveX

See also: RENDER

Syntax

RENDERLIB=*libref*

libref

specifies a previously defined libref that identifies the SAS library. The default library is WORK. See “LIBNAME Statement” on page 29 for more information on assigning a libref.

REPAINT

Specifies how many times to redraw the graph.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Restriction: not supported by Java or ActiveX

Syntax

REPAINT=*redraw-factor*

redraw-factor

is a nonnegative integer up to three digits long (0...999).

Details

Use this option with printers that produce light images after only one pass. This option also is useful for producing transparencies; multiple passes make the colors more solid or more intense.

Not all devices have this capability.

RESET

Resets graphics options to their defaults and/or cancels global statements.

Used in: GOPTIONS statement

Syntax

RESET=ALL | GLOBAL | *statement-name* | (*statement-name(s)*)

ALL

sets all graphics options to defaults and cancels all global statements.

GLOBAL

cancels all global statements (AXIS, FOOTNOTE, LEGEND, PATTERN, SYMBOL, and TITLE). Options in the GOPTIONS statement are unaffected.

statement-name

resets or cancels only the specified global statements. For example, RESET=PATTERN cancels all PATTERN statements only. To cancel several statements at one time, enclose the statement names in parentheses. For example, RESET=(TITLE FOOTNOTE AXIS).

Note: RESET=GOPTIONS sets all graphics options to defaults but does not cancel any global statements. △

Featured in: “Example 10. Creating a Bar Chart with Drill-down for the Web” on page 255

Details

RESET=ALL or RESET=GOPTIONS must be the first option specified in the GOPTIONS statement; otherwise, the graphics options that precede RESET= in the GOPTIONS statement are reset. Other options can follow the RESET= graphics option in the statement.

REVERSE

Specifies whether to print the output in reverse order, if reverse printing is supported by the device.

Used in: GOPTIONS statement

Default: NOREVERSE

Restrictions: hardware dependent, PostScript printers require a PPD file; not supported by Java or ActiveX

See also: PPDFILE

Syntax

REVERSE | NOREVERSE

Details

The purpose of REVERSE is to control the stacking order of printer output, depending on how the printer stacks paper. On some printers, reverse implies using the alternate output bin (back of the printer).

For PCL devices, REVERSE sends output to the LOWER out bin, which is the face-up output bin.

For PostScript devices, if the PPD file has an “OutputOrder” entry and one of its entries is “Reverse,” the device supports reverse order printing and the appropriate PostScript code to activate reverse will be used. If the PPD file does not have an “OutputOrder” entry but does have a “PageStackOrder” entry and corresponding OutputBin value, then reverse order printing is supported indirectly, using the PPD file’s PageStackOrder/OutputBin entries.

Note: Some PostScript devices implement Reverse as the default output mode for one of the output bins. In this case, selecting either the “reverse” output bin or specifying REVERSE mode produces identical results. △

ROTATE

Specifies whether and how to rotate the graph.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Detail window

Restriction: not supported by Java or ActiveX

Syntax

GOPTIONS: ROTATE=LANDSCAPE | PORTRAIT ROTATE | NOROTATE

GDEVICE: ROTATE=LANDSCAPE | PORTRAIT

ROTATE | NOROTATE

specifies whether to rotate the graph 90 degrees from its default orientation.

ROTATE=LANDSCAPE

specifies landscape orientation (the graph is wider than it is high).

ROTATE=PORTRAIT

specifies portrait orientation (the graph is higher than it is wide).

If you do not specify a rotation, a default is searched for in this order:

- 1 the ORIENTATION setting on an OPTIONS statement
- 2 device-dependent default.

ROTATION

Sets the increment of the angle by which the device can rotate any given letter in a string of text in a Metagraphics driver.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Default: ROTATION=0

Restriction: Used only with user-supplied Metagraphics drivers.

Syntax

ROTATION=*angle-increment*

angle-increment

specifies the increment of the angle at which to rotate individual characters, for example, every 5 degrees, every 45 degrees, and so on. *Angle-increment* is an integer in the range 0 to 360. A value of 0 means that the device uses its default character rotation. Specify 0 if your device does not perform hardware character rotation.

Details

For information about Metagraphics drivers, contact Technical Support.

ROWS

Specifies the number of rows the hardware font uses in graphics output.

Used in: GDEVICE Chartype window; GDEVICE procedure; CHARREC= option

Default: 0

See also: CHARREC

Syntax

See “CHARREC” on page 269 for syntax.

Details

If you are using a device driver from SASHELP.DEVICES, this parameter already is set for hardware fonts that have been defined for your installation. For scalable fonts, you can specify 1 for ROWS, and the actual number of rows will be computed based on the current text width. If you are adding to or modifying hardware fonts available for a particular device driver, specify a positive value for the ROWS device parameter. If ROWS is greater than 0, it overrides the values of the LROWS and PROWS device parameters.

SCALABLE

Specifies whether a font is scalable.

Used in: GDEVICE Chartype window; GDEVICE procedure; CHARREC= option

Default: device dependent

See also: CHARTYPE

Syntax

See “CHARREC” on page 269 for syntax.

Details

A hardware font is scalable if it can be used with any combination of rows and columns. Use the SCALABLE device parameter if you are adding to or modifying the fonts available for a particular device driver. If you are using a device driver from SASHELP.DEVICES, this parameter already is set for hardware fonts that have been defined for your installation.

SIMFONT

Specifies a software font to use if the default hardware font cannot be used.

Used in: GOPTIONS statement

Default: SIMULATE

Restriction: not supported by Java or ActiveX

Syntax

SIMFONT=*software-font*

software-font

specifies a software font to use instead of the default hardware font. By default, this is the SIMULATE font, which is stored in the SASHELP.FONTS catalog.

Details

SAS/GRAPH substitutes the software font specified by the SIMFONT= option for the default hardware font in these cases:

- when you use the NOCHARACTERS option in a GOPTIONS statement
- when you specify a non-default value for the HPOS= or VPOS= graphics option and your device does not have scalable hardware characters
- when you replay a graph using a device driver other than the one used to create the graph
- when you specify an angle or rotation for your hardware text that the device is not capable of producing
- when you specify a hardware font that is not supported by your device.

See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for details.

SPEED

Selects pen speed for plotters with variable speed selection.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Restriction: not supported by Java or ActiveX

Syntax

SPEED=*pen-speed*

pen-speed

specifies a percentage (1 through 100) of the maximum pen speed for the device. For example, SPEED=50 slows the drawing speed by half. In general, slowing the drawing speed produces better results.

By default, the value of SPEED is the normal speed for the device.

SWAP

Specifies whether to reverse BLACK and WHITE in the graphics output.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Defaults: GOPTIONS: NOSWAP; GDEVICE: SWAP=N

Restriction: not supported by Java or ActiveX

Syntax

GOPTIONS: SWAP | NOSWAP

GDEVICE: SWAP=Y | N

SWAP

SWAP=Y

swaps BLACK for WHITE and vice versa.

NOSWAP

SWAP=N

does not swap the colors. A blank **Swap** field in the Parameters window is the same as SWAP=N.

Details

SWAP does not affect the background color and only affects BLACK and WHITE foreground colors specified as predefined SAS color names. SWAP ignores BLACK and WHITE specified in HLS, RGB, or gray-scale format. This option is useful when you want to preview a graph on a video device and send the final copy to a printer that uses a white background.

```
goptions reset=all cback=blue ctitle=black swap;
title1 h=8 'swap test';
title2 h=8 'another title';
proc gslide border;
run;
```

SWFONTRENDER

Specifies the method used to render software fonts.

Used in: GOPTIONS statement

Default: device dependent

Restriction: not supported by Java or ActiveX

Syntax

SWFONTRENDER = POLYGON | SCANLINE

SWFONTRENDER = POLYGON

uses polygon rendering

SWFONTRENDER = SCANLINE

uses scanline rendering

Details

SWFONTRENDER determines the method used to render software text to a vector graphics file. In some graphics formats, SCANLINE rendering may produce better quality output when displayed to the screen, but the text may become unrecognizable if the output is resized, or it may be distorted if the output is replayed on a device with a

different resolution than the original device. If the software text is rendered as a POLYGON, resizing the graph will not distort the text.

SYMBOL

Specifies whether to use the device's hardware symbol-drawing capability.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

Restriction: not supported by Java or ActiveX

See also: SYMBOLS

Syntax

GOPTIONS: SYMBOL | NOSYMBOL

GDEVICE: SYMBOL=Y | N

SYMBOL

SYMBOL=Y

causes SAS/GRAPH to use the built-in symbol-drawing capability of the device, if available. A blank **Symbol** field in the Parameters window is the same as SYMBOL=Y.

Hardware drawing is faster, but not all devices have the capability. SAS/GRAPH does not try to use the capability if your device does not support it.

NOSYMBOL

SYMBOL=N

causes SAS/GRAPH to draw the symbols using software fonts.

SYMBOLS

Specifies which symbols can be generated by hardware.

Used in: GDEVICE procedure; GDEVICE Parameters window

Default: device dependent

See also: "SYMBOL Statement" on page 183

Syntax

SYMBOLS='hardware-symbols-hex-string'X

hardware-symbols-hex-string

is a hexadecimal string that is 16 characters long and must be completely filled. This table shows which bit position (left-to-right) within the hexadecimal string controls each hardware symbol.

Bit to turn on	Symbol Description	Symbol
1	PLUS	+
2	X	×
3	STAR	*
4	SQUARE	□
5	DIAMOND	◇
6	TRIANGLE	△
7	HASH	#
8	Y	Y
9	Z	Z
10	PAW	⋮
11	POINT	.
12	DOT	●
13	CIRCLE	○

For example, if you want the driver to do only the PLUS and X symbols in hardware, the first and second bits of the first byte of the hexadecimal string should be turned on, which would correspond to a value of 'C000000000000000'X ('C0'X is equivalent to '1 1 0 0 0 0 0 0' in binary).

Details

These are not the only symbols that can be generated for graphics output but are the symbols that can be drawn by the hardware. SAS/GRAPH can draw other symbols.

Note: Not all devices are capable of drawing every symbol. If a particular hardware symbol is not available or not to be used (as indicated by the value of SYMBOLS), the symbol is generated by the software. If the value of the SYMBOL device parameter in the device entry is N or the NOSYMBOL graphics option is used, the value of SYMBOLS is ignored. △

TARGETDEVICE

Displays the output as it would appear on a different device. Also, specifies the device driver for the PRINT command.

Alias: TARGET

Used in: GOPTIONS statement

Restriction: not supported by Java or ActiveX

Syntax

TARGETDEVICE=*target-device-entry*

target-device-entry

specifies the name of a device entry in a catalog.

Details

Use TARGETDEVICE= to specify a device driver when you want to:

- preview graphics output on your monitor as it would appear on a different output device. For details, see “Previewing Output” on page 52.
- print output from the Graph window or the Graphics Editor window with the PRINT command. For details, see “Printing Graphics Output” on page 51.
- specify a device driver for graphics output created by the ODS HTML statement.

TRAILER

Specifies the command that creates TRAILER records for the Metagraphics driver.

Used in: GDEVICE procedure; GDEVICE Metagraphics window

Restriction: Used only with user-supplied Metagraphics drivers

See also: TRAILERFILE

Syntax

TRAILER=*command*'

command

specifies a command that runs a user-written program that creates the TRAILER file. *Command* is a string up to 40 characters long.

Details

For information about Metagraphics drivers, contact Technical Support.

TRAILERFILE

Specifies the fileref of the file from which the Metagraphics driver reads TRAILER records.

Used in: GDEVICE procedure GDEVICE Metagraphics window

Restriction: Used only with user-supplied Metagraphics drivers

See also: TRAILER

Syntax

TRAILERFILE=*fileref*

fileref

specifies a valid SAS fileref up to eight characters long. *Fileref* must have been previously assigned with a FILENAME statement or a host command before running the Metagraphics driver. See “FILENAME Statement” on page 28 for additional information on the FILENAME statement.

Details

For information about Metagraphics drivers, contact Technical Support.

TRANSPARENCY

Specifies whether the background of the image should appear to be transparent when the image is displayed in the browser.

Used in: GOPTIONS statement

Default: NOTRANSPARENCY

Restriction: GIF series of drivers only

Syntax

TRANSPARENCY | NOTRANSPARENCY

Featured in: “Example 8. Creating a Simple Web Page with the ODS HTML Statement” on page 245

Details

When the image is displayed and TRANSPARENCY is in effect, the browser’s background color replaces the driver’s background color, causing the image to appear transparent.

TRANTAB

Selects a translate table for your system that performs ASCII-to-EBCDIC translation.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Host File Options window

Default: host dependent

Restriction: not supported by Java or ActiveX

Syntax

TRANTAB=*table* | *user-defined-table*

table

specifies a translate table stored as a SAS/GRAPH catalog entry. *Table* can be one of the following:

SASGTAB0 (default translate table for your operating environment)

GTABVTAM

GTABTCAM

user-defined-table

specifies the name of a user-created translate table.

Details

TRANTAB is set by the SAS Installation Representative and is needed when an EBCDIC host sends data to an ASCII graphics device. See the SAS/GRAPH installation instructions for details. You can also create your own translate table using the TRANTAB procedure. For a description of the TRANTAB Procedure, see *Base SAS Procedures Guide*.

TYPE

Specifies the type of output device to which graphics commands are sent.

Used in: GDEVICE procedure; GDEVICE Detail window

Default: device dependent

Syntax

TYPE=CAMERA | CRT | EXPORT | PLOTTER | PRINTER

CAMERA

specifies a film-recording device.

CRT

specifies a monitor or terminal.

EXPORT

identifies the list in which the device appears under SAS/ASSIST software. This is used for drivers that produce output to be exported to other software applications, such as CGM or HPGL.

PLOTTER

specifies a pen plotter.

PRINTER

specifies a printer

Details

You should not modify this value for Institute-supplied device drivers.

UCC

Sets the user-defined control characters for the device.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Parameters window

Restriction: device dependent; not supported by Java or ActiveX

Syntax

`UCC='control-characters-hex-string'X`

control-characters-hex-string

is a hexadecimal string that can be up 32 bytes (64 characters) long. You only need to specify up to the last non-zero byte; the remaining bytes will be set to zero.

Details

Not all devices support this feature, and the meaning of each byte of the string varies from device to device.

UCC values for specific devices are listed in the SAS Help facility for SAS/GRAPH. Typically the UCC byte position is indicated by a bracketed value. For example, UCC[2] refers to the second byte of the string. Refer to the Help for the UCC device parameter for details.

USERINPUT

Determines whether user input is enabled for the device.

Used in: GOPTIONS statement

Default: NOUSERINPUT

Restrictions: GIFANIM driver only; not supported by all browsers

Syntax

USERINPUT | NOUSERINPUT

USERINPUT

enables user input

NOUSERINPUT

disables user input

Details

When user input is enabled, processing of the animation is suspended until a carriage return, mouse click, or some other application-dependent event occurs. The user input feature works with the delay time setting so that processing continues when user input occurs or the delay time has elapsed, whichever comes first.

VORIGIN

Sets the vertical offset from the lower-left corner of the display area to the lower-left corner of the graph.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Detail window

Restriction: not supported by Java or ActiveX

See also: HORIGIN

Syntax

VORIGIN=*vertical-offset* <IN | CM | PT>

***vertical-offset* <IN | CM | PT>**

must be a nonnegative number and may be followed by a unit specification, either IN for inches (default), or CM for centimeters, or PT for points. If you do not specify VORIGIN, a default offset is searched for in this order:

- 1 the bottom margin specification on an OPTIONS BOTTOMMARGIN setting
- 2 VORIGIN setting in the device catalog.

Details

The display area is defined by the XMAX and YMAX device parameters. By default, the origin of the graphics output area is the lower-left corner of the display area; the graphics output is offset from the lower-left corner of the display area by the values of HORIGIN and VORIGIN. VORIGIN + VSIZE cannot exceed YMAX. See “About the Graphics Output Area” on page 34 for details.

VPOS

Sets the number of rows in the graphics output area.

Used in: GOPTIONS statement

Default: device dependent: the value of the LROWS or PROWS device parameter

Restriction: not supported by Java or ActiveX

See also: HPOS, LROWS, PROWS

Syntax

VPOS=*rows*

rows

specifies the number of rows in the graphics output area, which is equivalent to the number of hardware characters that can be displayed vertically. Specifying VPOS=0 causes the device driver to use the default hardware character cell height for the device.

Details

The VPOS= graphics option overrides the values of the LROWS or PROWS device parameters and temporarily sets the number of columns in the graphics output area. VPOS= does not affect the height of the graphics output area but merely divides it into rows. Therefore, you can use VPOS= to control cell height.

The values specified in the HPOS= and VPOS= graphics options determine the size of a character cell for the graphics output area and consequently the size of many graphics elements, such as hardware text. The larger the size of the HPOS= and VPOS= values, the smaller the size of each character cell.

See “Procedure Output and the Graphics Output Area” on page 34 for more information.

VSIZE

Sets the vertical size of the graphics output area.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Detail window

Restriction: partially supported by Java or ActiveX

See also: HSIZE, YMAX

Syntax

VSIZE=*vertical-size* <IN | CM | PT>

vertical-size <IN | CM | PT>

specifies the height of the graphics output area; *vertical-size* must be a positive number and may be followed by a unit specification, either IN for inches (default), or CM for centimeters, or PT for points. If you do not specify VSIZE=, a default size is searched for in this order:

- 1 the vertical size is calculated as

$YMAX - BOTTOMMARGIN - TOPMARGIN$

Note that `BOTTOMMARGIN` and `TOPMARGIN` are used in the `OPTIONS` statement.

- 2 `VSIZE` setting in the device catalog.

V6COMP

Allows programs that are run in the current version of SAS to run with selected Version 6 defaults.

Used in: `GOPTIONS` statement

Default: `NOV6COMP`

Restriction: partially supported by Java or ActiveX

Syntax

`V6COMP` | `NOV6COMP`

V6COMP

causes SAS/GRAPH programs to use these Version 6 behaviors:

- By default, patterns are hatched patterns, not solid, and the default outline color matches the pattern color.
- By default, the `GCHART` and `GLOT` procedures do not draw a frame around the axis area.

NOV6COMP

causes SAS/GRAPH programs to use all the features of the current SAS version.

Details

`V6COMP` performs the necessary conversions so that, for selected defaults, you get the same results in the current SAS version that you did in Version 6.

Note: `V6COMP` does not convert Version 6 catalogs to catalogs with the current SAS catalog format. \triangle

XMAX

Specifies the width of the addressable graphics display area; affects the horizontal resolution of the device and the horizontal dimension of the graphics output area.

Used in: `GOPTIONS` statement; `GDEVICE` procedure; `GDEVICE` Detail window

Restriction: Ignored by default display drivers, universal printing drivers, Java, and ActiveX

See also: `HSIZE`, `PAPERSIZE`, `XPIXELS`

Syntax

XMAX=*width* <IN | CM | PT>

width

is a positive number that may be followed by a unit specification, either IN for inches (default), or CM for centimeters, or PT for points. If you do not specify XMAX, a default width is searched for in this order:

- 1 the *width* specification on an OPTIONS PAPERSIZE setting
- 2 XMAX in the device entry catalog.

If XMAX=0, default behavior is used. If both XMAX and PAPERSIZE have been specified on GOPTIONS, the last request is used.

Details

Like the XPIXELS device parameter, XMAX controls the width of the display area, but the width is in inches, centimeters, or points rather than pixels. Typically, you might use XMAX to change the width of the display area for a hardcopy device.

SAS/GRAPH uses the value of XMAX in calculating the horizontal resolution of the device:

$$x\text{-resolution} = \text{XPIXELS} / \text{XMAX}$$

However, changing XMAX does not necessarily change the resolution:

- If you use the GOPTIONS statement to change only the value of XMAX= and do not change XPIXELS=, SAS/GRAPH retains the default resolution of the device and recalculates XPIXELS, temporarily changing the width.
- If you specify values for both XMAX= and XPIXELS=, SAS/GRAPH recalculates the resolution of the device using both of the specified values. The new resolution may or may not be different. For example, both of these pairs of values produce the same resolution, 300dpi:

$$\text{XPIXELS}=1500 \text{ and } \text{XMAX}=5$$

$$\text{XPIXELS}=1800 \text{ and } \text{XMAX}=6$$

XMAX also affects the value of HSIZE, which controls the horizontal dimension of the graphics output area.

- If you change the value of XMAX and do not change HSIZE=, SAS/GRAPH calculates a new value for HSIZE=, using this formula:

$$\text{HSIZE} = \text{XMAX} - \text{margins}$$

Note: The *margins* quantity, here, is not a device parameter. It represents the value of the left margin plus the right margin. The left margin is the value of HORIGIN. The right margin is whatever is left over when you subtract HSIZE and HORIGIN from XMAX. The value of *margins* is always based on the original XMAX and HSIZE values that are stored in the device entry. Δ

- If you specify values for both XMAX= and HSIZE=, SAS/GRAPH uses the specified values plus the value of device parameter HORIGIN. Anything left over is added to the right margin. For example, if XMAX=6IN and HSIZE=4IN and HORIGIN=.5IN, the right margin will be 1.5in. If HSIZE= is larger than XMAX=, HSIZE= is ignored.

To permanently change the value of the XMAX device parameter in the device entry, use the GDEVICE procedure. This can change the resolution.

To temporarily change the size of the display and the resolution of the device for the current graph or for the duration of your SAS session, use XMAX= and XPIXELS= in the GOPTIONS statement.

To reset the value of XMAX to the default, specify XMAX=0. To return to the default resolution for the device, specify both XMAX=0 and XPIXELS=0.

See “Procedure Output and the Graphics Output Area” on page 34 for more information.

XPIXELS

Specifies the width of the addressable display area in pixels and in conjunction with XMAX determines the horizontal resolution for the device.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Detail window

Default: device dependent

See also: XMAX

Restriction: Ignored by default display drivers and universal printing drivers; partially supported by Java or ActiveX

Syntax

XPIXELS=*width-in-pixels*

width-in-pixels

is a positive integer up to eight digits long (0...99999999).

Details

Like the XMAX device parameter, XPIXELS controls the width of the display area, but the width is in pixels rather than inches, centimeters, or points. Typically, you might use XPIXELS to change the width of the display area for an image format device.

Note: This option overrides the OutputWidth style attribute in the graph styles. For more information on graph styles, refer to the TEMPLATE procedure documentation in *SAS Output Delivery System: User's Guide*. \triangle

The value of XPIXELS is used in calculating the resolution of the device:

$$x\text{-resolution} = \text{XPIXELS} / \text{XMAX}$$

However, changing XPIXELS does not necessarily change the device resolution:

- \square If you use the GOPTIONS statement to change only the value of XPIXELS= and do not change XMAX=, SAS/GRAPH retains the default resolution of the device and recalculates XMAX, temporarily changing the width of the display. If HSIZE= is also not specified, SAS/GRAPH uses the new XMAX value to calculate a new HSIZE value, using this formula:

$$\text{HSIZE} = \text{XMAX} - \text{margins}$$

Note: *Margins* are not device parameters, but represent the value of HORIGIN (the left margin) plus the right margin. The right margin is whatever is left over when you subtract HSIZE and HORIGIN from XMAX. The values of *margins* is always based on the original XMAX and HSIZE values that are stored in the device entry. △

If HSIZE= is specified and its value is larger than XMAX, HSIZE= is ignored.

- If you use the GDEVICE procedure to permanently change the value of the XPIXELS device parameter in the device entry, SAS/GRAPH automatically recalculates the resolution of the device is using the value of XMAX device parameter.
- If you change the values of both XMAX= and XPIXELS=, SAS/GRAPH recalculates the resolution of the device using both of the specified values.

Note: When SAS/GRAPH recalculates the resolution, the resolution does not necessarily change. For example, both of these pairs of values produce the same resolution, 300dpi:

```
XPIXELS=1500 and XMAX=5
XPIXELS=1800 and XMAX=6
```

△

To reset the value of XPIXELS to the default, specify XPIXELS=0. To return to the default resolution for the device, specify both XPIXELS=0 and XMAX=0.

YMAX

Specifies the height of the addressable graphics display area; affects the vertical resolution of the device and the vertical dimension of the graphics output area.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Detail window

Restriction: ignored by default display drivers and universal printing drivers; not supported by Java or ActiveX

See also: PAPERSIZE, VSIZE, YPIXELS

Syntax

YMAX=*height* <IN | CM | PT>

height

is a positive number that may be followed by a unit specification, either IN for inches (default), or CM for centimeters, or PT for points. If you do not specify YMAX, a default height is searched for in this order:

- 1 the *height* specification on an OPTIONS PAPERSIZE setting
- 2 YMAX in the device entry catalog.

If YMAX=0, default behavior is used. If both YMAX and PAPERSIZE have been specified on GOPTIONS, the last request is used.

Details

See “XMAX” on page 362.

YPIXELS

Specifies the height of the addressable display area in pixels and in conjunction with YMAX determines the horizontal resolution for the device.

Used in: GOPTIONS statement; GDEVICE procedure; GDEVICE Detail window

Default: device dependent

See also: YMAX

Restriction: ignored by default display drivers and universal printing drivers; partially supported by Java or ActiveX

Syntax

YPIXELS=*height-in-pixels*

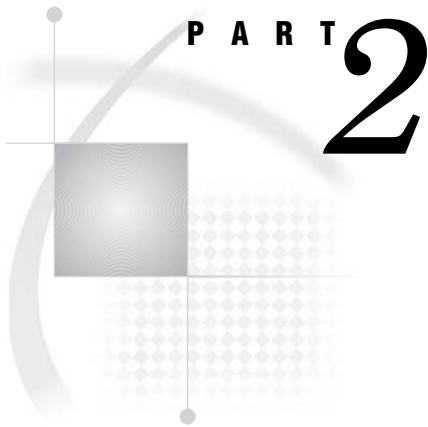
height-in-pixels

is a positive integer up to eight digits long (0...99999999).

Details

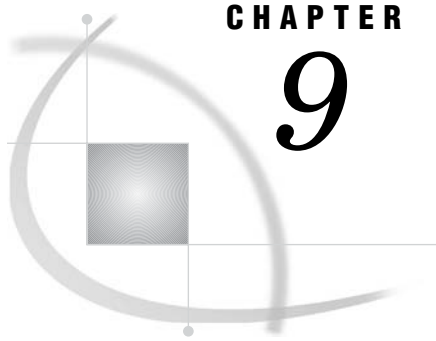
See “XPIXELS” on page 364.

Note: This option overrides the OutputHeight style attribute in the graph styles. For more information on graph styles, refer to the TEMPLATE procedure documentation in *SAS Output Delivery System: User’s Guide*. \triangle



Bringing SAS/GRAPH Output to the Web

- Chapter 9* **Introducing SAS/GRAPH Output for the Web** 369
- Chapter 10* **Creating Interactive Output for ActiveX** 387
- Chapter 11* **Creating Interactive Output for Java** 397
- Chapter 12* **Attributes and Parameters for Java and ActiveX** 421
- Chapter 13* **Generating Static Graphics** 439
- Chapter 14* **Generating Web Animation with GIFANIM** 457
- Chapter 15* **Generating Interactive Metagraphics Output** 469
- Chapter 16* **Managing Web Output with ODS** 487
- Chapter 17* **Generating Web Output with the Annotate Facility** 499
- Chapter 18* **Creating Interactive Treeview Diagrams** 503
- Chapter 19* **Creating Interactive Constellation Diagrams** 513
- Chapter 20* **Creating Critical Success Factor Diagrams** 527
- Chapter 21* **Macro Arguments for the DS2CONST, DS2TREE, DS2CSF, and META2HTM Macros** 535
- Chapter 22* **Enhancing Web Output** 567
- Chapter 23* **Troubleshooting Web Output** 579



CHAPTER

9

Introducing SAS/GRAPH Output for the Web

<i>Which Device Driver or Macro Do I Use?</i>	369
<i>Types of Web Presentations Available</i>	370
<i>Presentations That Use The ActiveX Control</i>	370
<i>Presentations That Use Java Applets</i>	371
<i>Graph, Map, and Contour Applets</i>	372
<i>Treeview Applet</i>	372
<i>Constellation Applet</i>	373
<i>Rangeview Applet</i>	374
<i>Metaview Applet</i>	375
<i>Presentations that Use Static Images</i>	376
<i>ACTXIMG Presentations</i>	377
<i>JAVAIMG Presentations</i>	377
<i>GIF, JPEG, and PNG Presentations</i>	377
<i>Animated GIF Presentations</i>	378
<i>Selecting a Type of Web Presentation</i>	378
<i>How is the graphical output produced?</i>	378
<i>What features are supported for each type of presentation?</i>	379
<i>What does your audience need to view the presentation?</i>	380
<i>Recommendations</i>	381
<i>Generating Web Presentations</i>	382
<i>Using ODS with a SAS/GRAPH Procedure</i>	382
<i>Using DS2TREE, DS2CONST, and DS2CSF Macros</i>	383
<i>Using META2HTM with a SAS/GRAPH Procedure</i>	384
<i>Changing the Location of Online Help for Java and ActiveX</i>	385

Which Device Driver or Macro Do I Use?

Generating a web presentation that includes graphics requires that you use a device driver or macro that generates web output. Determining which device driver or macro to use requires that you consider issues such as

- What type of graph do I need?
- What procedure, if any, generates the graph that I need?
- In which operating environments do I need to generate the presentation?
- In which operating environments do I need to deliver the presentation?
- Will my audience need to install additional software to view the presentation?
- What interactive features do I want in my presentation?

The following topics describe the types of web presentations that are available, help you decide which type you need, and tell you how to generate the presentation and

deliver it to your audience. The primary purpose of these topics is to help you determine which device driver or macro you need to use.

- “Types of Web Presentations Available” on page 370 describes each type of web presentation, their features, and which device driver or macro you need to use to create that type of presentation.
- “Selecting a Type of Web Presentation” on page 378 guides you through the process of determining which device driver or macro to use. If the type of presentation you need to generate can be generated with multiple device drivers, then additional factors determine which driver to use.
- “Generating Web Presentations” on page 382 summarizes the methods by which each type of web presentation is created.

Types of Web Presentations Available

Delivering information via the web frequently requires a web presentation that includes not only tables but graphics as well. SAS/GRAPH provides three basic ways to display presentations that include graphics. Presentations can be displayed

by an ActiveX control

The ActiveX control displays the output of SAS/GRAPH procedures. It enables such features as pop-up data tips, drill-down links, and interactive menus. The ActiveX control also enables you to use Output Delivery System (ODS) styles. For more information, see “Presentations That Use The ActiveX Control” on page 370.

by a Java applet

Java applets display the output of SAS/GRAPH procedures and macros. Depending on the applet, it may enable such features as data tips, drill-down links, or interactive features available through a pop-up menu. For more information, see “Presentations That Use Java Applets” on page 371.

as a static graph

You can also generate graphs that do not have any interactive features but do have interactive capabilities such as data tips or drill-down links. Static graphs can be generated as GIF, JPEG, or PNG files. For more information, see “Presentations that Use Static Images” on page 376.

For additional information about SAS/GRAPH output for the Web, including samples, refer to

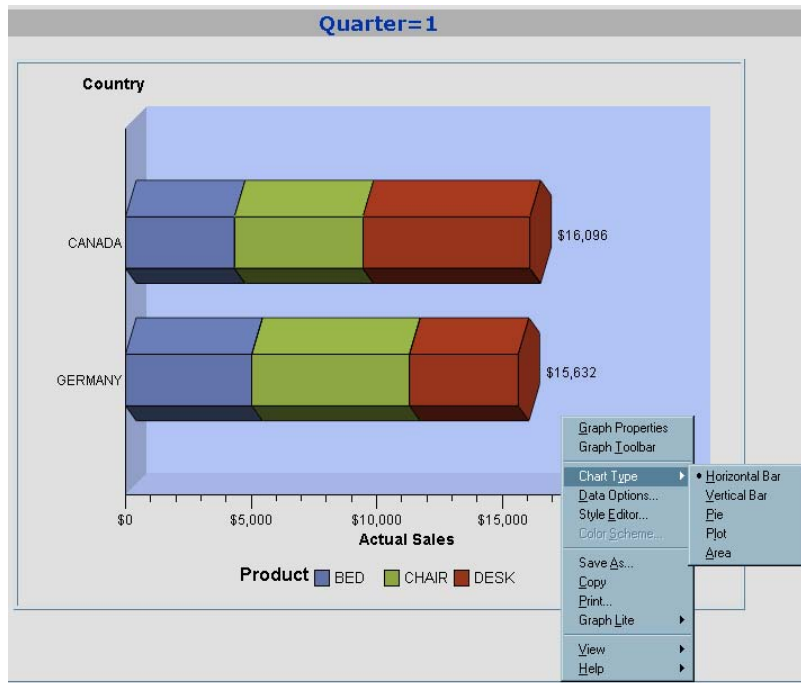
<http://support.sas.com/rnd/datavisualization>

Presentations That Use The ActiveX Control

The SAS/GRAPH ActiveX control displays the output of SAS/GRAPH procedures and enables extensive interactive features via a pop-up menu. The pop-up menus enable you to rotate, and zoom, and to control the properties of graphs such as its colors, legends, and axes.

You can use ODS styles with presentations created for the ActiveX control, and you can also enable pop-up data tips and drill-down links.

Display 9.1 on page 371 shows output from the GCHART procedure as displayed by the ActiveX control. (You can open the pop-up menu for the ActiveX control by positioning your cursor over the graph and pressing the right mouse button.)

Display 9.1 Sample ActiveX Presentation

The ActiveX control can be viewed only in the Windows operating environment with Microsoft's Internet Explorer on a PC with the ActiveX control installed.

The ActiveX control displays output from the G3D, GAREABAR, GBARLINE, GCHART, GCONTOUR, GPLOT, GMAP, and GRADAR procedures.

To create a graph to be displayed by ActiveX, specify `DEVICE=ACTIVEX` on your `GOPTIONS` statement. See "Using ODS with a SAS/GRAPH Procedure" on page 382 and Chapter 10, "Creating Interactive Output for ActiveX," on page 387 for more information.

Presentations That Use Java Applets

If you want to deliver your presentation to more operating environments than just Windows, you can use one of the following Java applets:

Graph, Map, and Contour applets

These applets display the output of SAS/GRAPH procedures and offer many interactive features. The Graph and Map applets also enable you to use ODS styles.

Treeview, Constellation, and Rangeview applets

These applets generate hierarchical treeview diagrams, constellation diagrams, and critical success factor diagrams, respectively, and are generated with the `DS2TREE`, `DS2CONST`, and `DS2CSF` macros.

Metaview applet

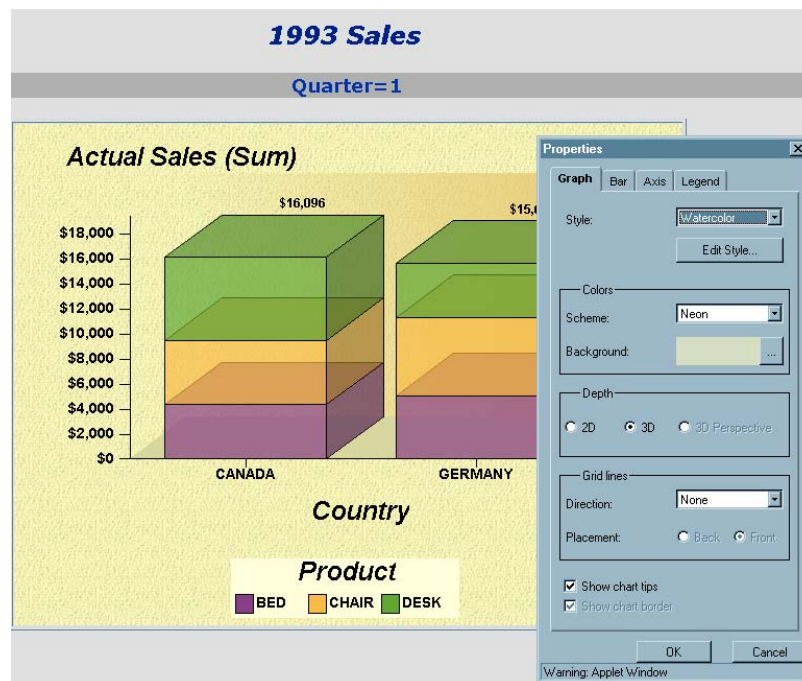
The Metaview applet displays the output of SAS/GRAPH procedures, and it enables pop-up data tips, drill-down links, and zooming.

Graph, Map, and Contour Applets

Like the ActiveX control, the Graph, Map, and Contour applets display the output of SAS/GRAPH procedures and enable extensive interactive features. The Graph and Map applets support ODS styles. The Graph, Map, and Contour applets enable data tips and drill-down links, and they provide pop-up menus which enable the user to pan, rotate, and zoom the graph, and to change properties such as the graph's colors, legends, and axes.

Display 9.2 on page 372 shows PROC GCHART output displayed by the Java Graph applet with a Properties dialog box. You can open the pop-up menu for these applets by positioning your cursor over the graph and pressing the right mouse button.

Display 9.2 Sample Java Presentation



These applets display the output of the following SAS/GRAPH procedures:

Graph Applet G3D Scatter Plots, GCHART, GPLOT

Contour Applet G3D Surface Plots, GCONTOUR

Map Applet GMAP

To create a graph to be displayed by one of these applets, specify `DEVICE=JAVA` on your `GOPTIONS` statement. For more information, see "Using ODS with a SAS/GRAPH Procedure" on page 382 and Chapter 11, "Creating Interactive Output for Java," on page 397.

Treewiew Applet

This applet displays a treeview diagram, which shows the parent-child relationships in a tree structure. In a treeview diagram, each child node has exactly one parent, and each parent node has zero or more children. In other words, the relationships in a treeview diagram are one-to-many. A treeview diagram is ideal for displaying such data as organizational charts or the hierarchical relationships of the pages of a Web site.

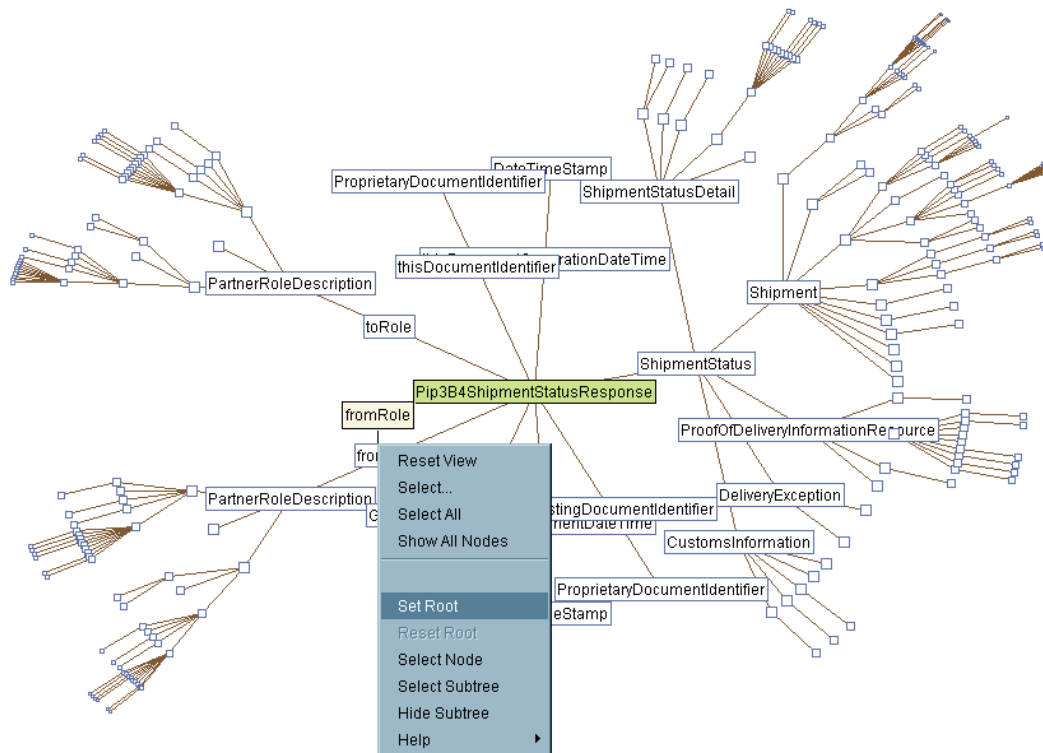
By default, the Treeview applet zooms in on the portion of the tree that is in the center of the display, as if you were looking through a fish-eye lens. Nodes in the center of the display are spread apart and shown with more detail, including node labels. Nodes near the periphery of the display are compressed and shown with less detail. Initially, the Treeview applet places the root node in the center of the display. You can click and drag the diagram to change the portion of the diagram that is in the center of the display.

The Treeview applet supports a pop-up menu that enables you to search for nodes, select or hide subtrees, and so on. You can add hotspots that link to Web pages when the user clicks on a node.

For example, Display 9.3 on page 373 shows a treeview diagram (with the pop-up menu opened) displaying the structure of an XML Document Type Definition.

To generate a treeview diagram, use the DS2TREE macro. For more information, see Chapter 18, “Creating Interactive Treeview Diagrams,” on page 503.

Display 9.3 Sample Treeview Diagram



Constellation Applet

The Constellation applet displays a general node-link diagram. Each node can be linked to one or more other nodes. Unlike the Treeview applet, the Constellation applet does not require a hierarchical relationship between the nodes. (Although it can be used to display hierarchical relationships, the Constellation applet does not automatically place the root node at the center of the display.)

The Constellation applet supports node and link properties, which determine the color and size of the nodes and the color and thickness of the link joining the nodes. These properties indicate the relative strength of the relationship between the nodes.

Like the Treeview applet, by default, the Constellation applet zooms in on the portion of the diagram that is in the center of the display, as if you were looking through a

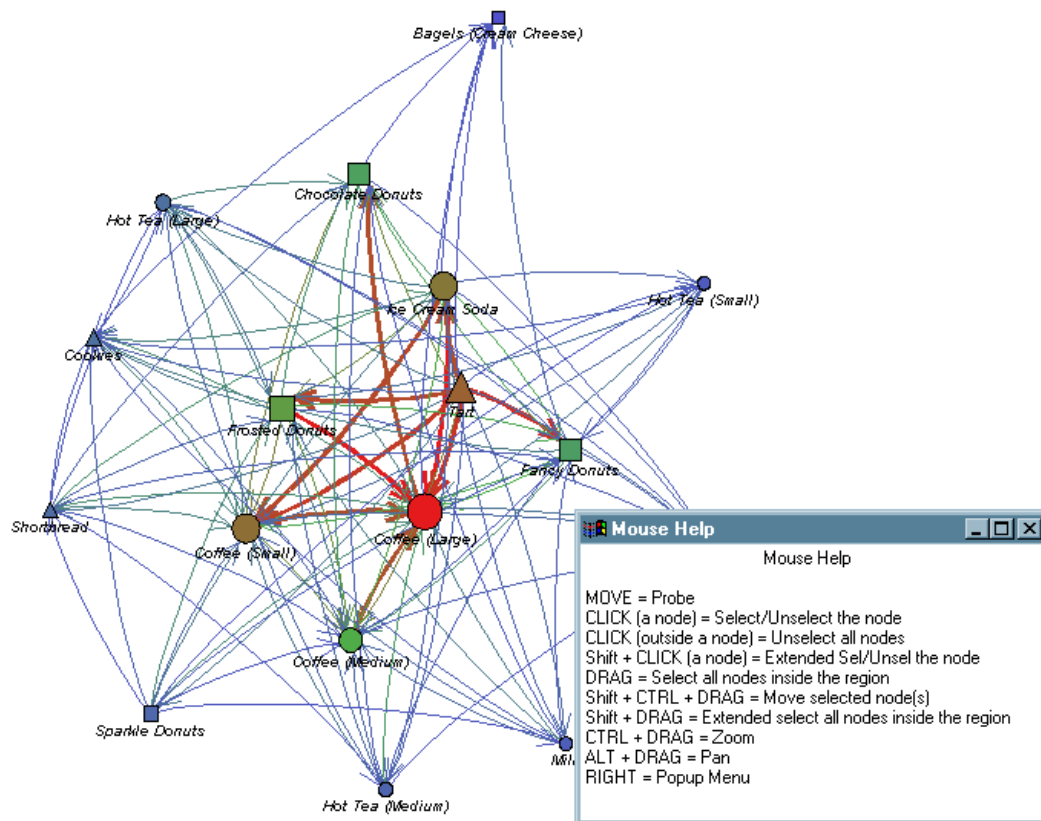
fish-eye lens. Nodes in the center of the display are spread apart and shown with more detail, including node labels. Nodes near the periphery of the display are compressed and shown with less detail. You can click and drag the diagram to change the portion of the diagram that is in the center of the display.

The Constellation applet has a pop-up menu that supports several functions such as highlighting specific links and searching for specific nodes. You can add hotspots that link to Web pages when the user clicks on a node.

Display 9.4 on page 374 shows a constellation diagram (with the Mouse Help menu displayed).

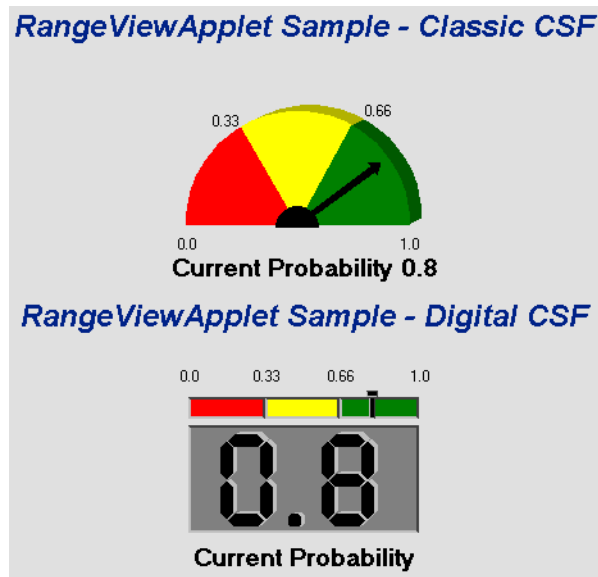
To generate the Constellation applet, use the DS2CONST macro. For more information, see Chapter 19, “Creating Interactive Constellation Diagrams,” on page 513.

Display 9.4 Sample Constellation Diagram



Rangeview Applet

The Rangeview applet displays critical success factor diagrams. Critical success factor diagrams display the value of a variable in a SAS data set in relation to a range of values. You can generate the Rangeview applet with the DS2CSF macro. Display 9.5 on page 375 shows two forms of output from the macro—one an analog dial and the other a digital display.

Display 9.5 Sample Rangeview Applet

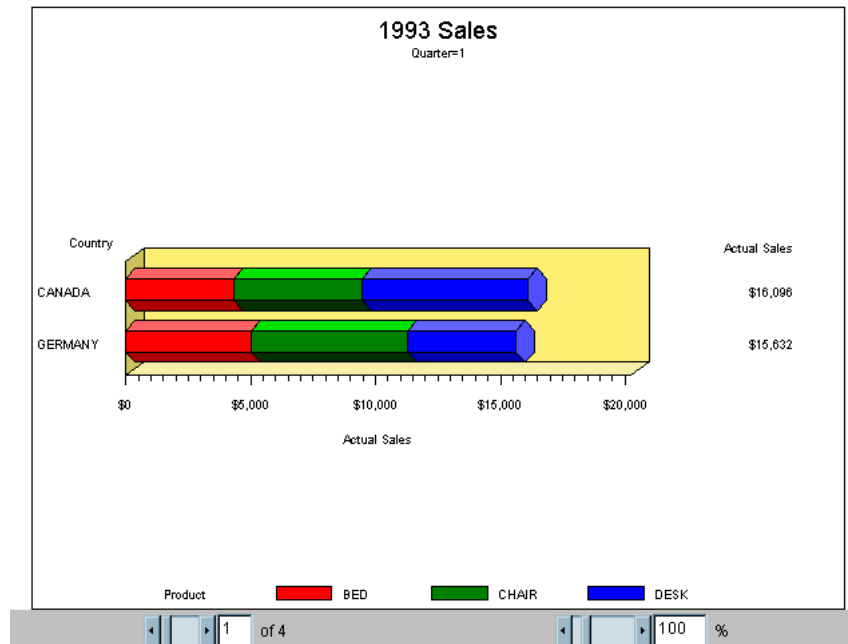
The DS2CSF macro generates an HTML file that invokes the Rangeview applet, and passes to the applet the information to be displayed on the dial.

For more information, see Chapter 20, “Creating Critical Success Factor Diagrams,” on page 527.

Metaview Applet

The Metaview applet displays the output of SAS/GRAPH procedures and enables interactive features that are not available with static images such as GIFs or JPEGs. It enables zooming and scrolling and supports pop-up menus with customized user-selectable links. When you generate a graph with the Metaview applet, you can specify background colors and text fonts, and enable drill-down links to HTML files, metagraphics files, and sets of metacodes.

Display 9.6 on page 376 shows the slider control on the bottom-left that the Metaview applet provides to enable a user to select which diagram to display.

Display 9.6 Sample Metaview Applet

The Metaview applet displays output from the G3D, GANNO, GBARLINE, GCHART, GCONTOUR, GPLOT, GMAP, GPRINT, GRADAR, GREPLAY, and GSLIDE procedures. To create a graph to be displayed by the Metaview applet, specify `DEVICE=JAVAMETA` on your `GOPTIONS` statement.

For additional information, see Chapter 15, “Generating Interactive Metagraphics Output,” on page 469.

Presentations that Use Static Images

If you do not need any interactive features in your presentations, then you can specify one of the following device drivers to generate a presentation that uses a GIF, JPEG, or PNG file.

ACTXIMG or JAVAIMG

create a web presentation that uses a static PNG image instead of an interactive applet. The images are identical to the images generated with the `ACTIVEX` and `JAVA` device drivers.

GIF, JPEG, or PNG

create web presentations that use static GIF, JPEG, or PNG images. These images are identical to the images generated by the server. (See “Resolving Differences Between Client and Server Graphs” on page 584.)

GIFANIM

generates a series of images that are displayed in sequence from a single GIF file.

To generate a web presentation that uses one of these drivers, specify the driver name with the `DEVICE=` option in your `GOPTIONS` statement. All of these device drivers generate output from `SAS/GRAPH` procedures.

For more information, refer to the following topics:

- “ACTXIMG Presentations” on page 377
- “JAVAIMG Presentations” on page 377

- “GIF, JPEG, and PNG Presentations” on page 377
- “Animated GIF Presentations” on page 378
- “Using ODS with a SAS/GRAPH Procedure” on page 382
- Chapter 13, “Generating Static Graphics,” on page 439.

ACTXIMG Presentations

You can use ODS and the ACTXIMG device driver to create a presentation that uses a PNG file that is identical in appearance to the image produced with the ACTIVEX device driver.

A presentation generated with the ACTXIMG driver supports data tips and drill-down links for GCHART, GBARLINE, and GPLOT (except for high-low plots) output. You can also use ODS styles with the ACTXIMG driver.

To render your output (create the PNG file), the ActiveX control must be installed on the PC where your SAS session is running. Because of this requirement, ACTXIMG presentations can be generated only on PCs. When you specify the ACTXIMG device driver, the output is rendered when your web presentation is generated, and the user does not need to have the ActiveX control installed to view it.

Note: The ACTXIMG device cannot be used with the ODS PDF, PCL, PS, or PRINTER destinations on 64-bit machines. SAS uses the JAVAIMG device instead. △

You can use ODS and the ACTXIMG device driver to generate presentations with the same procedures that are supported by the ACTIVEX driver: G3D, GAREABAR, GBARLINE, GCHART, GCONTOUR, GPLOT, GMAP, and GRADAR.

JAVAIMG Presentations

You can use ODS and the JAVAIMG device driver to create a presentation that uses a PNG file that is identical in appearance to the image produced with the JAVA device driver. You can use ODS styles to change the appearance of presentations generated with the JAVAIMG device driver.

Note: The Contour applet does not support ODS styles. △

The appropriate Java applet (Graph, Map, or Contour applet) is required to render your output (create the PNG file). The appropriate Java applet must be installed on the machine where your SAS session is running. When you specify the JAVAIMG device driver, the output is rendered when your web presentation is generated, and the user does not need to have any Java applet files installed to view it.

You can use ODS and the JAVAIMG device driver to generate presentations with the same procedures that are supported by the JAVA driver: G3D, GCHART, GCONTOUR, GPLOT, and GMAP.

GIF, JPEG, and PNG Presentations

Web presentations generated with the GIF, JPEG, or PNG device drivers use image files that are identical in appearance to the server (GRSEG) images. (See “Resolving Differences Between Client and Server Graphs” on page 584.) You can add pop-up data tips that are displayed when the cursor is over a portion of the image, and you can add hotspots that link to other Web pages.

You can use ODS and the GIF, JPEG, or PNG device drivers to generate presentations to display output from the G3D, GANNO, GBARLINE, GCHART, GCONTOUR, GPLOT, GMAP, GPRINT, GRADAR, GREPLAY, and GSLIDE procedures.

To create a web presentation that uses an image identical to the server image, specify `DEVICE=GIF`, `JPEG`, or `PNG` in your `GOPTIONS` statement.

Animated GIF Presentations

An animated presentation is a series of static images that are displayed automatically one after the other. Specify `DEVICE=GIFANIM` in your `GOPTIONS` statement to generate a web presentation that displays a series of images from a single GIF file. You can control the rate at which the successive images are presented.

You can generate animated GIF presentations from the `G3D`, `GANNO`, `GBARLINE`, `GCHART`, `GCONTOUR`, `GPlot`, `GMAP`, `GPRINT`, `GRADAR`, `GREPLAY`, and `GSLIDE` procedures.

For more information, see Chapter 14, “Generating Web Animation with `GIFANIM`,” on page 457.

Selecting a Type of Web Presentation

The type of web presentation that you choose to generate depends on several factors such as the type of graphs you need, the style of your presentation, the operating environment in which you want to generate your presentation, and the operating environments in which you plan to deliver your web presentation.

To determine which type of web presentation you need, consider the following questions:

How is your graphical output produced?

The structure of your data and the information that you need to generate from this data determine the type of graph that you need to produce. The type of graph that you need determines which procedure or macro you need to use to produce your graph. Which procedure or macro, if any, you need to use may determine which device drivers you can use.

What features are supported for each type of presentation?

Each type of web presentation enables different features such as data tips, drill-down links, and pop-up menus. Whether you need extensive interactive capabilities or just data tips can determine which device driver you need to use. Also, some device drivers support ODS styles, which may be important in your presentation.

What do you need to deliver the presentation?

Which device or macro you use to generate your web presentation determines whether the presentation can be viewed on multiple platforms and whether it requires any software except a supported browser.

How is the graphical output produced?

Which type of graph you need to produce is determined by the structure of your data and the information that you need to convey to your audience. For example, treeview diagrams and bar charts convey very different types of information. If you need to create a web presentation that includes graphics that are produced by one of the `SAS/GRAPH` procedures, then you need to use one of the device drivers that supports that procedure. Assuming that you know which type of graph you need, then you can determine which device drivers or macros you can use.

Table 9.1 on page 379 lists the procedures that are supported by each device driver and the diagrams that are produced by each macro.

Note: To generate a web presentation using the ACTXIMG device driver, the ActiveX control must be installed on the PC on which your SAS session is running. Δ

Table 9.1 How is the graphical output produced?

Driver or Macro	How Output is Produced
ACTXIMG	G3D, GAREABAR, GBARLINE, GCHART, GCONTOUR, GPLOT, GMAP, GRADAR
JAVAIMG	G3D, GCHART, GCONTOUR, GPLOT, GMAP
ACTIVEX	G3D, GAREABAR, GBARLINE, GCHART, GCONTOUR, GPLOT, GMAP, GRADAR
JAVA	G3D, GCHART, GCONTOUR, GMAP, GPLOT
GIF, JPEG, PNG	G3D, GANNO, GBARLINE, GCHART, GCONTOUR, GPLOT, GMAP, GPRINT, GRADAR, GREPLAY, GSLIDE
GIFANIM	G3D, GANNO, GBARLINE, GCHART, GCONTOUR, GPLOT, GMAP, GPRINT, GRADAR, GREPLAY, GSLIDE
JAVAMETA	G3D, GANNO, GBARLINE, GCHART, GCONTOUR, GPLOT, GMAP, GPRINT, GRADAR, GREPLAY, GSLIDE
DS2TREE, DS2CONST, DS2CSF	These macros create treeview diagrams, constellation diagrams, or critical success factor diagrams, respectively, without involving a SAS/GRAPH procedure.

For example, if you need a radar chart, you can use the ACTXIMG, ACTIVEX, or JAVAMETA driver (as well as other drivers). Which device driver you choose depends on what additional features (such as interactive capabilities) you need and on how you plan to deliver your web presentation.

If you need to graph hierarchical relationships, consider using the DS2TREE macro to generate a treeview diagram. If you need to show relationships that are not hierarchical or if you need to show the relative affinity of the relationships, then consider using the DS2CONST macro to generate a constellation diagram. If you need a critical success factor diagram, then you will need to use the DS2CSF macro.

What features are supported for each type of presentation?

The following table shows, for each type of Web presentation, what features are available to a viewer when viewing the presentation in a browser. You can see from the table that presentations that involve a Web executable, such as Java applets or the ActiveX control, enable interactive manipulation via pop-up menus. Presentations that use GIF, JPEG, and PNG files provide static images with no interactivity besides pop-up data tips and drill-down links.

After you have determined which device drivers or macros you can use, you then need to determine which extra features you need in your web presentation. For example, you may not want or need to give your audience the ability to subset the graph's data or change the graph from a bar chart to a pie chart.

The following table shows which features are supported for each device driver or macro.

Table 9.2 What features are supported for each type of presentation?

Driver or Macro	Features Supported
ACTXIMG	ODS styles, pop-up data tips and drill-down links (for selected output), static graphics with no interactivity
JAVAIMG	ODS styles, static graphics with no interactivity
ACTIVEX	ODS styles, pop-up data tips, drill-down links, interactivity via pop-up menus
JAVA	ODS styles, pop-up data tips, drill-down links, interactivity via pop-up menus
GIF, JPEG, PNG	Pop-up data tips, drill-down links, static graphics with no interactivity
GIFANIM	Slide show of static images with no interactivity
JAVAMETA	Pop-up data tips, drill-down links, some interactivity such as zooming and slide shows
DS2TREE, DS2CONST	Pop-up data tips, drill-down links, interactivity via pop-up menus
DS2CSF	Single drill-down link for the graph

Data tips and drill-down links for ACTXIMG are supported for output from GCHART, GPLOT (except for high-low plots), GBARLINE, and GRADAR.

The pop-up menus available with the JAVA and ACTIVEX device drivers typically enable your audience to change many aspects of the graph such as changing chart types, subsetting data, changing the variable used as the response variable, turning data tips on or off, or changing the colors used the graph. Static graphs do not offer any of these interactive features. Web presentations that use the JAVAMETA driver may enable a zoom control, and page selection and slide show controls for presentations that include multiple images.

What does your audience need to view the presentation?

To view your web presentation, your audience must view the presentation through one of the supported browsers. For a list of supported browsers, refer to the SAS Web site Install Center at

<http://support.sas.com/documentation/installcenter>

Select the *System Requirements* link for the appropriate operating system environment and search for the section on viewing HTML pages created for Java and ActiveX.

It is recommended that graphs be displayed on a device that has at least 16-bit color (that is, more than 8-bit, 256 colors).

Depending on how the presentation is generated, there may be additional requirements. The following table shows, for each type of Web presentation, what is required on a viewer's machine besides a supported browser.

Table 9.3 What does your audience need to view the presentation besides the browser?

Driver or Macro	Additional Requirements
ACTXIMG	None
JAVAIMG	None
ACTIVEX	The presentation must be viewed on a Windows system with the SAS ActiveX control installed locally.
JAVA	The Java applet files must be installed locally or on a server accessible by the client machine, and Java 1.4 plug-in must be installed on each client machine. On Windows systems, the user is prompted to install the plug-in if it is not already installed. On other systems, the plug-in can be installed from the Sun Microsystems site (http://www.sun.com) or from one of the SAS Third Party Software Components CDs.
GIF, JPEG, PNG	None
GIFANIM	None
JAVAMETA	The Java applet files must be installed locally or on a server accessible by the client machine. The Java plug-in is not required on the client machine; the Metaview applet works with the Java Virtual Machine that is built into the supported browsers.
DS2TREE, DS2CONST, DS2CSF	The Java applet files must be installed locally or on a server accessible by the client machine, and Java 1.4 plug-in must be installed on each client machine. On Windows systems, the user is prompted to install the plug-in if it is not already installed. On other systems, the plug-in can be installed from the Sun Microsystems site (http://www.sun.com) or from one of the SAS Third Party Software Components CDs.

Presentations generated with the ACTIVEX driver can be viewed only on Windows PCs, and the ActiveX control must be installed locally on each PC.

Presentations generated with the JAVAMETA driver can be viewed in any supported browser and offer limited interactivity, but do not require that a Java plug-in be installed.

Recommendations

If you will be delivering your presentation on Windows only, you want to use ODS styles, and you want your audience to be able to interact with the graph, then you can use the ACTIVEX device driver. If you will be delivering your presentation to other operating environments, but you still want to use ODS styles and extensive interactive features, then you can use the JAVA device driver. However, the ACTIVEX and JAVA device drivers require that your audience install the ActiveX control and Java plug-in, respectively.

If you want the look of the ACTIVEX or JAVA driver, but do not need the interactive capability or do not want to require that your audience install the ActiveX control or the Java plug-in, then use the ACTXIMG or JAVAIMG device drivers.

If you need data tips, drill-down capability, limited interactivity such as zoom, and want your graphs to look like the GRSEG produced by the server, but you do not want to require that your audience install the Java plug-in or the ActiveX control, then you can use the JAVAMETA device driver.

If you need only data tips and drill-down capability and prefer that your graphs look like the GRSEG produced by the server (see “Resolving Differences Between Client and Server Graphs” on page 584), then you can use the GIF, JPEG, or PNG device driver.

Generating Web Presentations

As shown in Figure 9.1 on page 382, there are three basic methods in which you can generate a web presentation:

- using ODS with a SAS/GRAPH procedure.
- using the META2HTM macro with a SAS/GRAPH procedure.
- using the DS2TREE, DS2CONST, or DS2CSF macro.

Figure 9.1 Methods for Creating Web Presentations

ODS	META2HTM	DS2TREE DS2CONST
SAS/GRAPH Procedures		DS2CSF
SAS Data Set		

Using ODS with a SAS/GRAPH Procedure

The recommended method for getting procedure output on the Web is with ODS. By using ODS in a program with one or more SAS/GRAPH procedures, you can create an HTML file and its associated SAS/GRAPH (or tabular) output.

At a minimum, to use ODS with SAS/GRAPH you must do the following:

- 1 Use a GOPTIONS statement to specify a device driver with the `DEVICE=device-driver` option, where *device-driver* is one of the following:
 - ACTIVEX or ACTXIMG
 - JAVA or JAVAIMG
 - GIF, JPEG, or PNG
 - GIFANIM
 - JAVAMETA

Note: When you specify `DEVICE=JAVAIMG` on z/OS, you must specify the SAS system option `FILESYSTEM=HFS`. HFS file space is needed to run the Java JRE 1.4. △

- 2 Close the ODS LISTING destination. If you do not close this destination, SAS/GRAPH creates a duplicate copy of your graph in your current directory.
- 3 Open an HTML output file using an ODS statement such as `ODS HTML` or `ODS MARKUP`. At a minimum, you must use the `FILE=` (alias `BODY=`) option to

specify a body file. For device drivers that generate image output files, use the PATH= option to ensure that all output files are stored in the same location.

- 4 Run a graphics procedure. Be sure to include RUN and QUIT statements.
- 5 Close the HTML destination.
- 6 Open the LISTING destination if needed.

Figure 9.2 on page 383 shows the basic structure of a SAS program for generating Web output with ODS and a SAS/GRAPH procedure.

Figure 9.2 Using ODS with SAS/GRAPH Procedures

GOPTIONS	goptions device= <i>device-driver</i> ;
DATA step	DATA mydata; ;
ODS HTML	ods listing close; ods html file="output-file.htm " path="url-or-fileref " gpath="file-or-url-or-fileref " style=banker;
SAS procedure	proc gchart data=mydata; vbar x / sumvar=y; run; quit;
ODS HTML CLOSE	ods html close; ods listing

When you use ODS, you can use ODS styles to control the appearance of a Web presentation created from SAS procedures with the following device drivers: ACTIVEEX, JAVA, ACTXIMG, and JAVAIMG. For more information on ODS styles, see “Using ODS Styles” on page 488.

Using DS2TREE, DS2CONST, and DS2CSF Macros

The following macros generate a Web presentation from a SAS data set:

- DS2TREE generates treeview diagrams
- DS2CONST generates constellation diagrams
- DS2CSF generates critical success factor diagrams.

To use these macros, simply define your data, then call one of these macros using the appropriate options. For these macros, you do not use ODS or call a SAS/GRAPH procedure. For additional information, refer to Chapter 18, “Creating Interactive

Treeview Diagrams,” on page 503, Chapter 19, “Creating Interactive Constellation Diagrams,” on page 513, and Chapter 20, “Creating Critical Success Factor Diagrams,” on page 527.

Using META2HTM with a SAS/GRAPH Procedure

The META2HTM macro offers an alternative to ODS for creating a Web presentation from a SAS/GRAPH procedure and the JAVAMETA device driver. At a minimum, to use the META2HTM macro you must do the following:

- 1 Use a GOPTIONS statement to specify the JAVAMETA device driver.
- 2 Define the fileref _WEBOUT. When you specify DEVICE=JAVAMETA, the output of a SAS/GRAPH procedure is directed to the file specified by _WEBOUT.

```
filename _webout "output-file.htm";
```

- 3 Call the META2HTM macro with the appropriate options to capture the metagraphics information.
- 4 Run a graphics procedure.
- 5 Call the META2HTM macro with the appropriate options to finish capturing the metagraphics information.

Figure 9.3 on page 384 shows the structure of a simple program that generate a Javameta applet to display the output of the GCHART procedure.

Figure 9.3 Using the META2HTM Macro

GOPTIONS	goptions device= <i>javameta</i> ;
DATA step	DATA mydata; ;
FILENAME	FILENAME _webout " <i>output_file.htm</i> ";
META2HTML on	%META2HTM (capture= on, htmlref=_webout, openmode=replace, archive=metafile.zip);
SAS procedure	proc gchart data=mydata; vbar x / sumvar=y; run; quit;
META2HTM off	%META2HTM (capture=off, htmlref=_webout, openmode=append);

The META2HTM macro directs the procedure to create its output as metacodes, and generates an HTML page that invokes the Metaview Applet to display the metacodes. For more information, see “Using the META2HTM Macro” on page 471.

Changing the Location of Online Help for Java and ActiveX

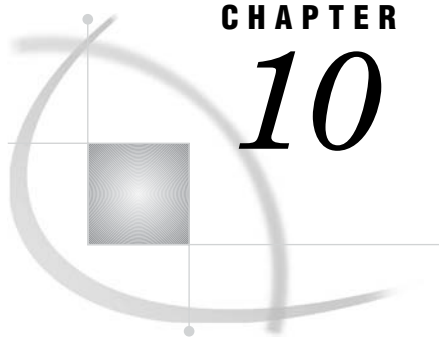
Online help is available for the ActiveX control and the Java applets. The online help explains the use of the interactive features that are available for each control or applet. To display the online help, position your cursor over the graph, press the right mouse button to open the pop-up menu, and select Help.

By default, the pop-up menus access help files on the SAS Website. If your audience does not have Internet access or if you want to customize the help files for your specific needs, you may want to change the location that is accessed by the Help selection. For the ActiveX control and the Graph, Map, and Contour applets, you can download the help files to a local machine and use the HELPLLOCATION parameter to point to these local files. For the Constellation and Treeview applets, there is currently no way to change the help location.

You can copy the help files for the ActiveX Control from the SAS Client-Side Components CD Volume 1. The help files for the Java applets are on the SAS Mid-Tier Components CD. You can download and modify these files as needed for your audience. Save the files in a location that is accessible to your audience. Then, in your SAS program, specify the applet parameter HELPLLOCATION to point to the location of the help. For example:

```
ods html file="ncpop.htm"  
      parameters=("HELPLLOCATION"="http://www.b.com/help/");
```

For more information on specifying applet parameters, see “Specifying Applet Parameters Using the ODS PARAMETERS= Statement” on page 477.



CHAPTER

10

Creating Interactive Output for ActiveX

<i>Overview</i>	387
<i>When to Use the ACTIVEX Device Driver</i>	388
<i>Installing the ActiveX Control</i>	389
<i>Manually Installing the ActiveX Control</i>	389
<i>Configuring Your Program to Prompt Users to Install the ActiveX Control</i>	389
<i>Prompting for Installation of the ActiveX Control</i>	390
<i>Uninstalling the ActiveX Control</i>	390
<i>Generating Output for ActiveX</i>	391
<i>About Languages and Special Fonts in ActiveX</i>	392
<i>Configuring Drill-Down Links with ACTIVEX</i>	392
<i>ActiveX Examples</i>	393
<i>Embedding ActiveX Graphs in Word Files</i>	393
<i>Generating an Interactive Contour Plot in ActiveX</i>	394
<i>Creating Graphs Interactively</i>	395
<i>Creating Graphs</i>	395
<i>Inserting the ActiveX Control into Microsoft Word Documents</i>	395

Overview

In the Windows operating environment, in the Internet Explorer Web browser, the SAS/GRAPH Control for ActiveX provides a high degree of graphical interactivity. Interactive features include the ability to change graph types (a two-dimensional bar chart can be changed to a three-dimensional bar chart, for example), display data values at the point of the cursor, rotate and zoom, reassign variable roles, and modify axes, legends, colors, and text fonts.

If your Web audience has SAS installed locally, the control runs automatically when the HTML output file is displayed on a Web browser that uses ActiveX device drivers. Members of your Web audience that do not have the SAS system installed locally, and who have not already installed the ActiveX Control, can be prompted to install the control at display time, as described in “Installing the ActiveX Control” on page 389.

You can enhance your ActiveX presentations by

- adding drill-down links (see “Configuring Drill-Down Links with ACTIVEX” on page 392)
- specifying graph styles in ODS (see “Using ODS Styles” on page 488)
- configuring interactive features (see “Specifying Parameters and Attributes for Java and ActiveX” on page 421)
- enhancing HTML output “Overview of ODS Enhancements for Web Output” on page 487.

In addition to HTML output, you can use the ActiveX Control to display interactive graphs in Microsoft Word, in Object Linked Embedded (OLE) documents, and in applications written in Visual Basic, C++, and JavaScript.

The following table lists the procedures and statements whose output can be displayed in the SAS/GRAPH Control for ActiveX.

Table 10.1 Procedures and Statements that Generate Output for the ActiveX Control

Procedure	Statements
GCHART	HBAR, HBAR3D, VBAR, VBAR3D, PIE, PIE3D, DONUT
GPLOT	PLOT, BUBBLE, BUBBLE2, PLOT2
GCONTOUR	PLOT
GMAP	CHORO, BLOCK, PRISM
G3D	PLOT, SCATTER
GBARLINE	BAR, PLOT
GAREABAR	HBAR, HBAR3D, VBAR, VBAR3D
GRADAR	CHART

Note: The ActiveX Control does not enable 8-bit grayscale images. Be sure to specify 24-bit images for backgrounds and chart elements. \triangle

Note: Note that using PROC GMAP to generate a highly detailed map might create a large HTML output file, which might cause problems on certain Web browsers. If this is the case, you can use PROC GREDUCE to remove some of the complexity and produce a more usable map. \triangle

When to Use the ACTIVEX Device Driver

If your Web audience uses the Windows operating environment and the Internet Explorer Web browser, the ActiveX Control might be preferable over a Java applet from a performance standpoint.

In general, the interactive features of the ActiveX Control are comparable to those that are provided in Java through the Java applets. Some features differ, as you can see in the comparison table that is presented in the “Parameter Reference for Java and ActiveX” on page 424. Also, the JAVA device driver does not display output generated with the GAREABAR, GBARLINE, or GRADAR procedures.

Unlike the JAVA device driver, you can use the ACTIVEX device driver to embed interactive graphics in Microsoft Word documents. Simply use the ODS RTF statement, as shown in “Embedding ActiveX Graphs in Word Files” on page 393. You can also copy the ActiveX window out of Internet Explorer and paste that material into a Microsoft Word document.

If you want to use an ODS style to enhance the appearance of your graph, but you do not need interactivity, then use the ACTXIMG device driver, as described in “Developing Web Presentations with the JAVAIMG and ACTXIMG Device Drivers” on page 442. You can use the ACTXIMG device driver only on Windows systems. Although you do not need it when viewing the output, to produce the output file you must install it on computers that use ActiveX device drivers.

Note that you can generate output for the ActiveX Control even if you are not working in the Windows operating environment. For example, you can generate HTML output for ActiveX in the Unix operating environment, even though you cannot run Internet Explorer in that environment. Moving the output file into Windows will display the output as if it was generated in that operating environment.

When you use the ACTIVEEX device driver with an ODS statement that does not enable interactive output, the output is automatically changed to the PNG image that is generated with the ACTXIMG device driver. For example, the ODS PDF statement generates output for the Adobe Acrobat Reader in Portable Document Format (PDF). This format does not enable embedded ActiveX applications. Specifying the ACTIVEEX device driver with the ODS PDF statement generates a PDF output file that contains a static image in PNG format.

The ACTXIMG device driver can produce an image map in the HTML to enable data tips and drill-down functionality from the image.

Installing the ActiveX Control

You can install the SAS/GRAPH Control for ActiveX manually, as described in “Manually Installing the ActiveX Control” on page 389, or your presentation can prompt the Web user through the installation process, as described in “Configuring Your Program to Prompt Users to Install the ActiveX Control” on page 389 and “Prompting for Installation of the ActiveX Control” on page 390.

Manually Installing the ActiveX Control

Follow these steps to manually install the ActiveX Control on a Web browser that uses ActiveX device drivers.

- 1 Obtain the installation program, in one of the following ways:
 - Installing SAS/GRAPH software.
 - Copying the program from the SAS Client-Side CD-ROM, which is provided with SAS/GRAPH software.
 - Downloading the program from the SAS Web site, at

`http://www.sas.com/apps/demosdownloads/setupintro.jsp`

and selecting the SAS/GRAPH Software link.

- 2 Run the installation program (`sasgraph.exe`). You will have an opportunity to change the default storage location, which is as follows:

`C:\Program Files\SAS Institute\Shared Files\Graph\Vx`

Where *x* is the version number. Installation requires seven megabytes of disk space.

Configuring Your Program to Prompt Users to Install the ActiveX Control

You can set up your SAS/GRAPH program so that the resulting presentation will prompt users to install the ActiveX Control. Note that no files are installed without permission. Users can refuse installation by refusing the licensing agreement at the

beginning of the installation process. Also note that the installation program will not run if the control has already been installed.

To be able to access the installation program, Web users must be able to access its storage location. You might need to copy the program to another location to ensure availability.

If you are using ODS to generate your HTML output file, then you can reference the installation program when you open the HTML output file. For example:

```
ods html body="myGraph.html"
  codebase="http://www.ourco.com/sasweb/graph/sasgraph.exe#version=9,1,0,304";
```

If the program is not stored on a Web ActiveX device driver, then you can use a file specification as the value of the CODEBASE attribute. For example:

```
ods html body="myGraph.html"
  (codebase="/grsrc/sasgraph.exe#version=9,1,0,304");
```

Prompting for Installation of the ActiveX Control

You can edit an existing ACTIVEX presentation so that it will prompt users to install the SAS/GRAPH Control for ActiveX. If the control is already present, then the component setup program will not run.

Follow these steps to add the installation capability to your ACTIVEX presentation:

- 1 In a text editor, open the initial HTML file of your Web presentation.
- 2 In the OBJECT tag, insert the CODEBASE= attribute. The attribute references the location of the installation program. The following CODEBASE value references a public directory:

```
CODEBASE="file://grsrc/sasgraph.exe"
```

If the installation program is stored on a Web ActiveX device driver, use an HTTP reference:

```
CODEBASE="http://www.ourco.com/sasweb/graph/sasgraph.exe#version=9,1,0,304"
```

- 3 Save the HTML file and close the editor.

Now displaying the HTML file gives users who need it the option of installing the control in the default location on their local computers.

Note that if you want to install the control in a non-default location, you need to install the control manually, as described in “Manually Installing the ActiveX Control” on page 389.

Uninstalling the ActiveX Control

If you installed the ActiveX Control as a stand-alone component, then you can use the following steps to uninstall the ActiveX Control.

- 1 Open the Control Panel window by selecting

Start ► Settings ► Control Panel

- 2 Select

Add/Remove Programs

- 3 Select

SAS Graph Component

- 4 Select the `Add/Remove` button.

Because several different products can install the ActiveX Control, you might need to repeat the install process.

Generating Output for ActiveX

The SAS/GRAPH Control for ActiveX displays interactive charts, maps, and plots. The following table lists the various ways that you can deliver ActiveX output to your audience.

Table 10.2 Primary Delivery Choices for ActiveX Output

Application	ODS Statement	Output File
Internet Explorer	ODS HTML	HTML
Microsoft Word	ODS RTF	Rich text format
Adobe Acrobat Reader	ODS PDF	Portable document format
Ghostview, etc.	ODS PSL	PostScript Format

Table 10.1 on page 388 lists the SAS/GRAPH procedures that generate output for ActiveX.

Follow these steps to generate a default Web presentation that runs the SAS/GRAPH Control for ActiveX.

- 1 Close the ODS listing destination to conserve resources:

```
ods listing close;
```

- 2 Open an output file in ODS and optionally specify an ODS style:

```
ods html file="your_file.htm"
style="banker";
```

- 3 Specify the ACTIVEEX device driver and set other graphics options:

```
goptions reset=all
device=activex
border;
```

For information on graph styles, see “Using ODS Styles” on page 488.

- 4 Specify a data set and run a procedure or procedures that are used by the ACTIVEEX device driver (see Table 10.1 on page 388):

```
proc gchart data=sashelp.class;
vbar height / group=age;
run;
quit;
```

- 5 Close the HTML output file and reopen the ODS listing destination:

```
ods html close;
ods listing;
```

The preceding program structure assumes that your Web audience has installed the ActiveX Control in advance. For information on prompting new users to start the installation process, see “Configuring Your Program to Prompt Users to Install the

ActiveX Control” on page 389. For further troubleshooting information, see “Troubleshooting Web Output” on page 579. For information on enhancing the default Web presentation, see “Configuring Drill-Down Links with ACTIVEX” on page 392 and “Overview of ODS Enhancements for Web Output” on page 487.

About Languages and Special Fonts in ActiveX

For international audiences, the ActiveX Control has a graphical user interface that can appear in the following languages: Chinese (simplified), Danish, English, French, German, Hebrew, Hungarian, Italian, Japanese, Korean, Polish, Russian, and Spanish. To display a translated graphical user interface, in general, Web-based ActiveX device drivers must use a language-specific operating environment and Web browser. This requires the all-languages version of the JRE. For further information, contact your SAS support consultant.

In the LABEL and SYMBOL statements, the Java applets enable the following markers: B, C (up triangle), M (club), N (heart), O (spade), P (diamond), U (square), and V (star). For a full list of the marker font that applies to these letters, see Figure 5.5 on page 87. Also enabled are the following symbols: D (diamond), H (circle), L (up triangle). For a full list of the special font, see Figure 5.8 on page 89.

Configuring Drill-Down Links with ACTIVEX

ActiveX parameters are used to implement drill-down functionality and to configure interactive features. The purpose and syntax of these parameters are defined in “Parameter Reference for Java and ActiveX” on page 424.

In the ODS HTML statement, ActiveX parameters are specified with the PARAMETERS= option, as described in “Specifying Parameters and Attributes for Java and ActiveX” on page 421.

The SAS/GRAPH Control for ActiveX enables three drill-down modes for charts and maps. Drill-down functionality is not enabled for contour plots. The three drill-down modes are URL, HTML, and Script. These modes are implemented in ActiveX in the same way that they are implemented in Java. For information on implementing these drill-down modes, see “Configuring Drill-Down Links for Java and ActiveX” on page 400. To convert the Java examples to ActiveX, simply change the GOPTIONS statement from DEVICE=JAVA to DEVICE=ACTIVEX.

The following table lists the procedures and statements whose output can be used in ActiveX presentations with drill-down functionality.

Table 10.3 Statements Enabled for Drill-Down Functionality in ActiveX

Procedure	Statements
GCHART	HBAR, HBAR3D, VBAR, VBAR3D, PIE, PIE3D, DONUT
GPLOT	PLOT, BUBBLE, BUBBLE2, PLOT2

GMAP

CHORO, BLOCK, PRISM

G3D

PLOT, SCATTER

ActiveX Examples

Embedding ActiveX Graphs in Word Files

The following example demonstrates how the ODS RTF statement can be combined with the ACTIVEX device driver to generate interactive graphics inside Microsoft Word files. The example also shows how user-defined formats can be used to enhance the appearance of graph labels and the headings of PROC PRINT tabular output.

The following example is available in the SAS Sample library under the name GWBAXRTF.

```
ods rtf file='c:\Public\graph\9.1\gwbaxrtf.rtf' style=torn;
options dev=activex;
PROC FORMAT;
  value agefmt    0-10='0 to 10'
                 11-12='Pre-Teen'
                 13-15='Mid-Teen'
                 16='late-Teen';
  value $sexfmt   'm'='Male'
                 'f'='Female';
  value weightfmt 0-99.9='0-99 lbs'
                 100-119='100-119 lbs'
                 120-149='120-149 lbs'
                 150-999='150 lbs or more';
run;
data class; set sashelp.class;
  format age agefmt. sex $sexfmt. weight weightfmt.;
  label sex='Gender';
  label height='Average Height (inches)';
  input name sex age height weight;
datalines;
Alfred m 14 69 112.5
Barbara f 13 65.3 98
Henry m 14 63.5 102.5
Jane f 12 59.8 84.5
John m 12 59 99.5
Joyce f 11 51.3 50.5
Mary f 15 66.5 112
Philip m 16 72 150
Robert m 12 64.8 128
Susan f 11 57.5 85
William m 15 66.5 112
;
run;
Title "RTF Output with ActiveX Control";
Title2 "Physical Statistics";
proc gchart data=class;
```

```

    hbar weight / sumvar=height type=mean group=sex discrete;
run;
quit;
Title2 "Formatted Classroom Data";
Proc print data=class; id name; run;
ods rtf close;

```

Generating an Interactive Contour Plot in ActiveX

The following example displays a contour plot of water depth on a small island known as an atoll. The ActiveX Control lets you manipulate many of the aspects of the plot using the menu that is displayed with the right mouse button.

Of note in this example is the way that PROC G3GRID is used to generate a rectangular grid of points.

This example, including the full data set, is available in the SAS Sample Library under the name GWBAXCON.

```

ods html file="your_web_path/your_HTML_file.htm";

goptions      device=activex
              xpixels=500      ypixels=350
              border           cback=white
              gunit=pct        htext=3;

data atoll;
    input vdist hdist depth;
datalines;
10   0.25   2.77
20   0.25   2.77
30   0.25   2.77
40   0.25   2.77
50   0.25   2.77
60   0.25   2.77
70   0.25   2.77
80   0.25   2.77
90   0.25   2.77
100  0.25   2.77
/* for the full data set, see GWBAXCON */
62   133.75  5.00
64   135     5.00
64   136.25  5.00
60   138.75  5.08
62   138.75  5.08
;
run;

/* Prepare the data to be a rectangular grid */
/* of points. */

proc g3grid data=atoll out=atollgrid;
    grid vdist*hdist=depth / naxis1=50 naxis2=50;
run;

```



```

title 'Pacific Atoll';

axis1 order=(0 to 150 by 25)   c=cx002288 width=3
      minor=(n=4)             label=('Meters');
axis2 order=(0 to 100 by 25)  c=cx002288 width=3
      minor=(n=4)             label=('Meters');

legend1 frame
      label=(position=top j=c 'Depth (in meters)')
      shape=line(7);

proc gcontour data=atollgrid;
  plot vdist*hdist=depth /
  levels = 1 2 2.5 3 3.5 4 4.5 5 7 9
  clevels = CXFF0000 CXF07275 CXEC9592
           CXE9A2B2 CXE5BFC6 CXBFA0CF
           CX9981D8 CX7362E1 CX4D43EA
           CX0000FF
  legend=legend1
  haxis=axis1
  vaxis=axis2
  des='Atoll';
run;
quit;

```

Creating Graphs Interactively

If you have Enterprise Guide 2.0 with HotFix 11 or higher installed, you can interactively create graphs within any Windows OLE application, such as Word or Excel, without installing SAS and the SAS/GRAPH procedures. You can drag-and-drop SAS data sets or import data from Microsoft Excel or Access. You can even create your own original data in a table format and pick a standard chart type to view that data.

Creating Graphs

Follow these steps to create graphs interactively.

- 1 Install Enterprise Guide 2.0 with HotFix 11 or higher.
- 2 Insert the control for this functionality into your Windows application or embed it in a Web page.
- 3 Assign the data and select a chart type. You can use SAS, Excel, or Access data.
- 4 Use any of the available default styles or customize them. If this is a Windows application, you can then either save or print the document from the application.

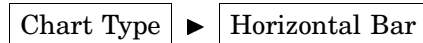
Inserting the ActiveX Control into Microsoft Word Documents

You can easily incorporate this functionality into a Microsoft Word document. Follow these steps to insert it into the document in the same way that you would insert any other object.

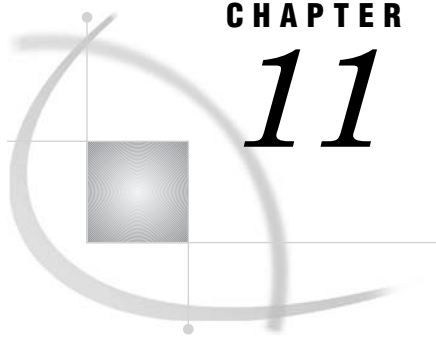
- 1 In Microsoft Word or Excel, position your cursor where you want your graph, then select



- 2 Select **SAS/GRAPH V9** in the Object window. The control for this functionality is inserted into your Word document. Although you see a graph, the control does not yet contain any real data.
- 3 You can add data to the control in any of the following ways.
 - Drag a SAS, Excel, or Access data file from Windows Explorer and drop it into the control.
 - Open a data file by selecting **Data Options** from the pop-up menu for the control. In the Data Options window, select **Open Data** and use the Open window to navigate to your data.
 - Create new data by selecting **Data Options** from the pop-up menu for the control. Select **Open Data** in the Data Options window. The control contains default data that enables it to display a graph. To change one of the default variable names, select the column title and enter a new name. To change one of the default variable values, select the cell and enter a new value.
- 4 Change the orientation of the bar by selecting



- from the pop-up menu.
- 5 Change the graph style by selecting **Style Editor** from the pop-up menu. In the Style Editor window you can change the style or color scheme that applies to the graph, or both. You can also isolate changes to chart elements, lines, text, or any combination of these. Experiment with the Style Editor. For example, change the style that is assigned in the Style name field.
 - 6 To save the current graph as an image, select **Save As** from the pop-up menu.
 - 7 To print the current graph, select **Print** from the pop-up menu.



CHAPTER

11

Creating Interactive Output for Java

<i>Overview</i>	397
<i>When to Use the JAVA Device Driver</i>	398
<i>Generating Output for Java</i>	398
<i>About Languages and Special Fonts in Java</i>	400
<i>Configuring Drill-Down Links for Java and ActiveX</i>	400
<i>Setting the Drill-Down Mode</i>	401
<i>Configuring the Local Drill-Down Mode</i>	401
<i>Understanding Default Behavior in Local Mode</i>	402
<i>Customizing Drill-Down Levels</i>	405
<i>Configuring the Script Drill-Down Mode</i>	407
<i>Working with the Array of Arguments</i>	408
<i>Generating a JavaScript</i>	409
<i>Configuring the URL Drill-Down Mode</i>	409
<i>Configuring the HTML Drill-Down Mode</i>	410
<i>Variables as Substitution Strings</i>	410
<i>Drill-Down Tags as Substitution Strings</i>	411
<i>Understanding Variable Roles</i>	411
<i>Using Drill-Down Tags</i>	412
<i>Configuring the Drill-down Response</i>	413
<i>Removing Blank Spaces from Data Values</i>	414
<i>Disabling the Drill-Down Functionality</i>	414
<i>Examples of Interactive Java Output</i>	415
<i>Local Drill-Down Mode</i>	415
<i>Script Drill-Down Mode</i>	416
<i>URL Drill-Down Mode</i>	417
<i>HTML Drill-Down Mode</i>	419

Overview

The JAVA device driver generates interactive presentations that run in the Graph, Map, and Contour applets. These applets enable users to interact with the output of the GCHART, GPLOT, G3D, GMAP, and GCONTOUR procedures.

The Java applets enable the Web user to display data values, to change the type of the graph, to pan, rotate, and zoom, and to change colors, fonts, axes, legends, and variable roles.

You can enhance Java graphs by setting applet parameters and specifying Output Delivery System (ODS) options. Applet parameters let you configure drill-down and override default values in the user interface. Information on parameters is provided in Chapter 12, “Attributes and Parameters for Java and ActiveX,” on page 421.

You can use ODS to enhance the appearance of Java charts using graph styles, as described in “Using ODS Styles” on page 488. You can also use ODS to generate other HTML enhancements, as described in “Overview of ODS Enhancements for Web Output” on page 487.

To generate a Web presentation that runs the Graph, Map, or Contour applet, you generally specify the JAVA device driver in a GOPTIONS statement, open an output file in ODS, specify an ODS style to set the appearance of the output, generate one or more graphs, and close the HTML output file, as described in “Generating Output for Java” on page 398.

You can generate the same graphs as static pictures using DEVICE=JAVAIMG. This does not require that the Web user install the applets or JRE. For details, see “ACTXIMG and JAVAIMG Device Drivers” on page 440.

When to Use the JAVA Device Driver

The JAVA device driver generates output for the Graph, Map, and Contour applets. These applets provided unprecedented levels of interactivity in all popular Web browsers.

If you need to generate interactive output in the Windows operating environment with the procedures GAREABAR, GBARLINE, or GRADAR, then use the ACTIVEX device driver, as described in Chapter 10, “Creating Interactive Output for ActiveX,” on page 387. ActiveX output can also appear in Microsoft Word documents or other OLE applications.

If you want to use an ODS style but do not need interactivity, then use the JAVAIMG device driver, as described in “Developing Web Presentations with the JAVAIMG and ACTXIMG Device Drivers” on page 442.

Generating Output for Java

To develop a SAS/GRAPH program that generates output for the Graph Applet or Map Applet, follow these steps:

- 1 Reset graphics options and specify a device driver.

```
goptions reset=all;
goptions device=java;
```

- 2 To conserve resources, close the default ODS output destination.

```
ods listing close;
```

- 3 Open an output file by specifying an ODS statement and a fully qualified path. Use the STYLE= option to specify an ODS style (see “Using ODS Styles” on page 488). Use the PARAMETERS= option to configure the applet (see “Specifying Parameters and Attributes for Java and ActiveX” on page 421). Use other ODS options to enhance the HTML (see “Overview of ODS Enhancements for Web Output” on page 487).

```
ods html
  file="/dept/web-server1/sales/q393/eastregion.html"
  style=gears
  parameters=("tips"="none")
  headtext="Georgia Peaches, Inc."
  nogtitle;
```

To run an applet, your audience must be able access the appropriate Java archive files. Two archives are referenced by default: one is the Java plug-in from Sun Microsystems, and the other is the SAS Java archive.

In the ODS output file, the location of the Java plug-in from Sun Microsystems is specified in the CODEBASE *attribute* of the OBJECT tag. If you need to change this default value, then use the ATTRIBUTES= option of the ODS statement, as described in “Specifying Parameters and Attributes for Java and ActiveX” on page 421. On Windows systems, the user is prompted to install the plug-in if it is not already installed. On other systems, the plug-in can be installed from the Sun Microsystems site (<http://www.sun.com>) or from one of the SAS Third Party Software Components CDs.

The location of the SAS Java archive is specified in the CODEBASE *parameter* in the body of the APPLET tag. The default CODEBASE is specified by the APPLETLOC= system option. If the default value of this system option specifies a widely-accessible URL, then you need not change this value. If you need to specify a different location, then you can change the value of the system option. Another alternative is to override the APPLETLOC= system option by specifying a value for the ODS statement option CODEBASE=, as described in “Specifying Parameters and Attributes for Java and ActiveX” on page 421.

Note: When specifying a location for the SAS Java archive, you can use an HTTP address, or you can use a UNC path, such as //sasjava, with forward slashes instead of backward slashes. \triangle

4 Provide data using a LIBNAME statement or a DATA step.

```
data regsales;
  length Region State $ 8;
  format Sales dollar8.;
  input Region State Sales;
  datalines;
West CA 13636
West OR 18988
West WA 14523
Central IL 18038
Central IN 13611
Central OH 11084
Central MI 19660
South FL 14541
South GA 19022
;
```

5 Generate your initial graphics output.

```
title1 'Regional Sales';
proc gchart data=regsales;
  vbar3d state / sumvar=sales
run;
quit;
```

The procedure does not require any additional statements to generate output that runs in an applet.

6 Close the HTML output file and reopen the ODS listing destination.

```
ods html close;
ods listing;
```

Running your program starts the applet and displays the initial graph.

If the browser display differs from what you would see in SAS without ODS, then ensure that your SAS/GRAPH procedure is fully enabled in the applet. Refer to Appendix 1, “Summary of ActiveX and Java Support,” on page 1507 for details.

Note: Using PROC GMAP to generate a highly detailed map might create a large HTML output file, which might cause problems on certain browsers. If this is the case, you can run PROC GREduce to remove some of the complexity and produce a more usable map. \triangle

For further information on troubleshooting Web output, see “Troubleshooting Web Output” on page 579.

About Languages and Special Fonts in Java

For international audiences, the Java applets have graphical user interfaces that can appear in the following languages: Chinese (simplified), Czech, Danish, English, French, German, Hebrew, Hungarian, Italian, Japanese, Korean, Norwegian, Polish, Russian, Spanish, and Swedish. To display a translated graphical user interface, in general, Web-based Java device drivers must use a language-specific operating environment and Web browser. This requires the all-languages version of the JRE. For further information, contact your SAS support consultant.

In the LABEL and SYMBOL statements, the Java applets enable the following markers: B, C (up triangle), M (club), N (heart), O (spade), P (diamond), U (square), and V (star). For a full list of the marker font that applies to these letters, see Figure 5.5 on page 87. Also enabled are the following symbols: D (diamond), H (circle), L (up triangle). For a full list of the special font, see Figure 5.8 on page 89.

Configuring Drill-Down Links for Java and ActiveX

The JAVA and ACTIVEX device drivers enable the following drill-down modes for presentations that run in the Graph applet, Map applet, and in the ActiveX Control (see Chapter 10, “Creating Interactive Output for ActiveX,” on page 387).

Local mode (Graph applet only)

responds to drill-down actions by dynamically generating and displaying new graphs. The data in the initial graph is subset based on the graph element that was selected in the drill-down action. In the drill-down graph, you can select again to generate another graph, as long as the data can still be subset, or as long as you have configured your own levels of drill-down. To configure a graph at a given level of drill-down, you specify the applet parameter DDLEVEL n . The value of this parameter determines the graph type, data subset, variable roles, and colors. Local is the default drill-down mode for the Graph applet. For details, see “Configuring the Local Drill-Down Mode” on page 401.

Script mode

calls a JavaScript method that you specify in your SAS/GRAPH program, and passes to that method information on the selected graph element or map region. The JavaScript method determines the browser’s response to the drill-down action. This drill-down mode is the default for the Map Applet and the ActiveX Control. See “Configuring the Script Drill-Down Mode” on page 407.

Note: Although scripting of applets using JavaScript is available, support is not provided if you customize the samples in this document or in the Sample Library. \triangle

URL mode

displays URLs that are provided by link variables. The link variables are identified to the graphics procedure with the HTML= option. The drill-down functionality of the URL mode is similar to the drill-down functionality that is provided by the GIF, JPEG, and PNG device drivers, with the addition of the improved rendering and the availability of graph styles in ODS. See “Configuring the URL Drill-Down Mode” on page 409.

HTML mode

generates drill-down URLs based on a substitution pattern that you specify in your SAS/GRAPH program. The Graph Applet, Map Applet, or the ActiveX Control completes the URL by inserting the specified data from the graph element that was selected in the drill-down action. See “Configuring the HTML Drill-Down Mode” on page 410.

Any mode (Graph applet and ActiveX control)

attempts to implement each of the four drill-down modes in succession until a valid Web destination is found. The order of the attempts is Local (Graph applet only), Script, URL, and HTML.

The Graph applet has selectable drill-down modes. You can select these modes from the applet menu.

Note: If the HTML= option is specified in the procedure that generates the initial graph, then the URL drill-down mode is automatically set. \triangle

The drill-down modes are specified as applet parameters, in the PARAMETERS= option of an ODS statement. Definitions of applet parameters are provided in “Parameter Reference for Java and ActiveX” on page 424.

Setting the Drill-Down Mode

To enable a given drill-down mode, specify a value for the applet parameter DRILLDOWNMODE. Like all other applet parameters, the DRILLDOWNMODE parameter is specified in an ODS statement. Use the following syntax to set the DRILLDOWNMODE parameter in the ODS statement:

ODS HTML

```
PARAMETERS=("DRILLDOWNMODE"="LOCAL" | "HTML" | "SCRIPT" |
            "URL" | "ANY");
```

The Local drill-down mode is enabled by default.

If the graphics procedure that generates the initial graph specifies the HTML= option, then the value of the DRILLDOWNMODE parameter is automatically set to URL. Any different mode that is specified in ODS is overridden.

Configuring the Local Drill-Down Mode

For the Graph applet only, you can specify the applet parameter DDLEVEL n to configure the Local drill-down mode. At any drill-down level you can specify the graph type, colors, and variable roles. Variable roles define how variables are applied to the axes of the graph, as described in “Understanding Variable Roles” on page 411.

An example of Local drill-down configuration would be to specify that the second-level drill-down graph is a pie chart, with non-default group and subgroup variables.

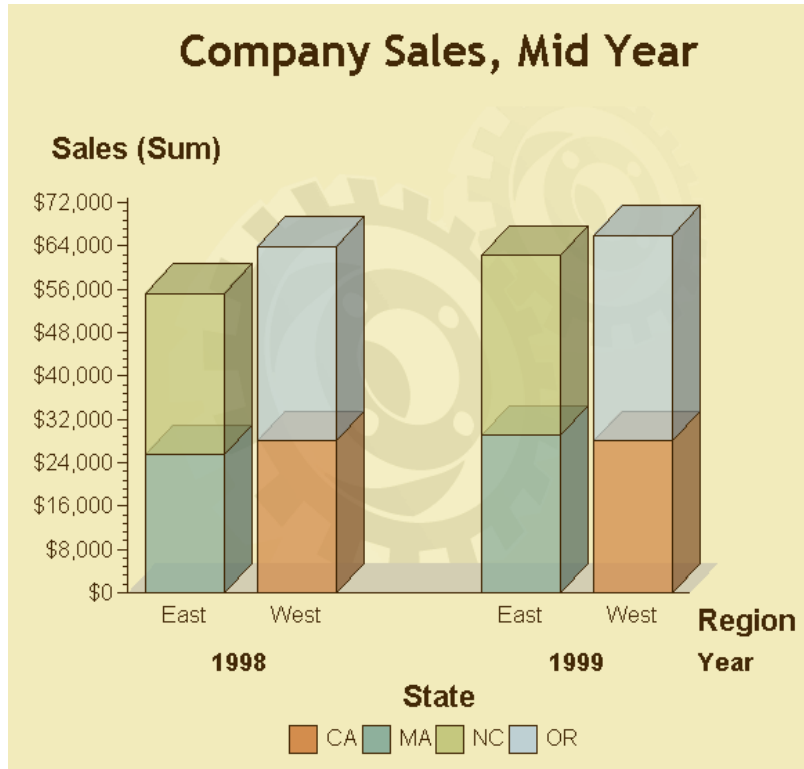
Understanding Default Behavior in Local Mode

To understand how you can configure the Local drill-down mode, it is best to learn how the Graph applet generates drill-down graphs by default.

The following code generates the graph shown in Display 11.1 on page 403.

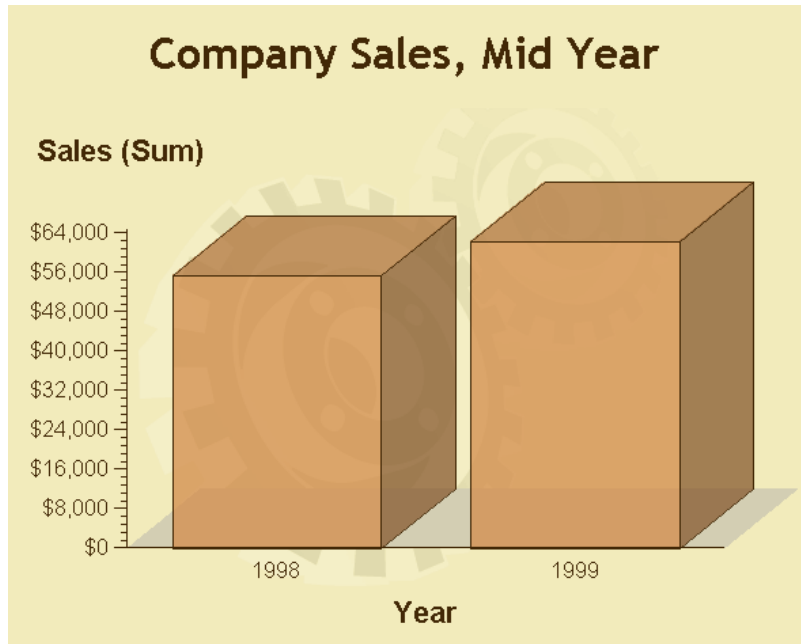
```
filename odsout 'C:\vbarweb.htm';
data sales;
  length Region $ 4 State $ 2;
  format Sales dollar8.;
  input Region State Sales Year Qtr;
  datalines;
West CA 13636 1999 1
West OR 18988 1999 1
/* see ''Local Drill-Down Mode'' on page 415 for the full data set */
East NC 12184 1998 2
East MA 12760 1998 2 ;
goptions reset=all device=java;
ods listing close;
ods html file=odsout
  style=gears;
title1 'Company Sales, Mid Year';
proc gchart data=sales;
  vbar3d region / sumvar=sales
  group=year subgroup=state;
run; quit;
ods html close;
ods listing;
```


Display 11.1 Graph Applet, Top-Level Graph



In a Web browser, selecting a bar in the graph causes the Graph applet to generate a new three-dimensional vertical bar chart. By default, the Graph applet retains the type and style of the initial graph in all the drill-down graphs in that presentation. In this example, all of the drill-down graphs will be three-dimensional vertical bar charts using the ODS graph style GEARS.

Selecting the bar on the far left in the initial graph generates the first-level graph shown in Display 11.2 on page 404.

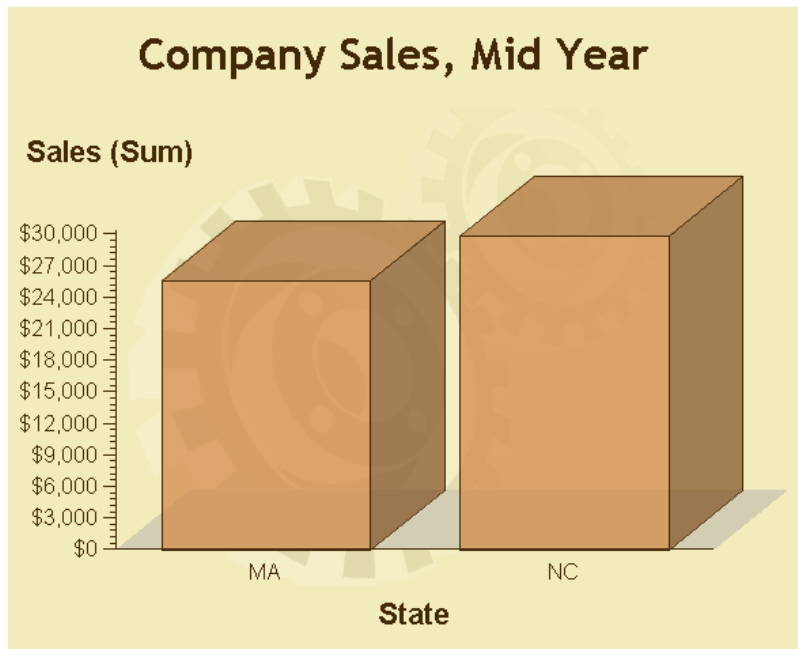
Display 11.2 Graph Applet, Local Drill-Down Mode, Level 1

The first-level drill-down graph retains the dependent variable SALES. The subset of data that the Graph Applet uses to generate the drill-down graph is defined by the drill-down action. The bar on the far left of the initial graph depicts part of the east region, so the drill-down graph shows east region sales only.

To differentiate the east region data, the Graph Applet makes the variable YEAR the independent variable. In the initial graph, the YEAR variable was a subgroup variable.

In the drill-down graph, the variable STATE is now the sole subgroup variable.

Subsetting repeats if you click on an element in the first-level drill-down graph. For example, selecting the bar on the far left (labeled 1998) displays the following second-level drill-down graph.

Display 11.3 Graph Applet, Local Drill-Down Mode, Level 2

The second-level drill-down graph shows 1998 sales figures for the east region states MA and NC. The dependent axis is unchanged and the STATE variable becomes the sole variable that is represented on the independent axis.

The second-level drill-down graph is the last that can be generated out of this data set, because no other variables can appear as independent variables.

The preceding example shows how the Graph applet generates drill-down graphs in the default configuration of the Local drill-down mode. The Graph applet retains the dependent variable and subsets the data based on the drill-down selection. At each level of drill-down, the applet promotes a new variable into the independent role. This succession can recur until the data cannot be subset any further. The succession uses all independent variables first, followed by all group and subgroup variables. Variables that are assigned to multiple roles are used in the order in which they appear in the data set.

Now that you see how the Graph applet generates Local-mode drill-win graphs by default, you can move on to configuring the drill-down graphs on your own, as described in “Customizing Drill-Down Levels” on page 405.

To see the SAS/GRAPH code that was used to generate this example, see “Local Drill-Down Mode” on page 415. You can use that example to experiment with different drill-down configurations.

Customizing Drill-Down Levels

The DDLEVEL n parameter lets you configure the graphs that are generated at specified drill-down levels. The DDLEVEL n parameter is specified as follows in the ODS statement:

```
ODS HTML
```

```
PARAMETERS=(“DDLEVEL $n$ ”=“string”);
```

n

represents the number of the drill-down level that is being configured.

string

specifies the graph type, the variable roles in the new graph, the color of the elements in the new graph, and the variable that is to be subset to create the elements in the new graph.

The syntax of the *string* argument is as follows:

```
{CHART} {chart_type} {tag_1} {variable_1...} {...tag_n} {variable_n} | {subset_tag_1...}
<{...subset_tag_n}>
```

```
{CHART} {chart_type}
```

identifies the type or style of the graph. This tag is case-sensitive: it must always be specified in uppercase. The values of the tag (chart types) are not case-sensitive. To use the same chart type as the preceding drill-down level, do not specify the CHART tag. Available chart types are as follows:

HBAR

generates a two-dimensional horizontal bar chart.

HBAR3D

generates a three-dimensional horizontal bar chart.

VBAR

generates a two-dimensional vertical bar chart.

VBAR3D

generates a three-dimensional vertical bar chart.

PIE

generates a two-dimensional pie chart.

PIE3D

generates a three-dimensional pie chart.

SCATTER

generates a scatter plot that is similar in appearance to Figure 46.13 on page 1319.

LINE

generates a line or needle plot that is similar in appearance to Figure 7.17 on page 192.

BOX

generates a box plot that is similar in appearance to Figure 7.15 on page 188.

HILO

generates a high-low plot that is similar in appearance to Figure 7.16 on page 190.

```
{tag_1} {variable_1...} {...tag_n} {variable_n}
```

associates drill-down tags with data set variables, to specify roles for variables in the new graph, and to optionally determine the color of the elements in the new graph. For definitions of the drill-down tags, see “Using Drill-Down Tags” on page 412.

```
{subset_tag_1...} <{...subset_tag_n}>
```

specifies one or more variable roles from the original graph whose values will be used to subset the data in the preceding graph. For example, if you specify G_GROUPV, then the data that will be used to draw the new graph will consist only of data that is associated with the group variable in the preceding graph. For example, if the group variable in the preceding graph was REGION, and if the value of that variable in the selected graph element was East, then the drill-down

graph would be drawn only with the observations where the REGION variable had the value of East.

At least one of the following tags must be specified as the subset variable: G_INDEPV, G_GROUPV, G_SUBGRV, or G_DEPTHV. For definitions of these tags, see “Using Drill-Down Tags” on page 412.

Specifying multiple subset variables means that two or more values must match the value in the selected graph element for that observation to be used in the new graph. For example, assume that you specify {G_INDEPV}{G_SUBGRV} as the subset variables, and that the selected graph element has an independent variable of YEAR and a subgroup variable of STATE. Also assume that the values for these variables in the selected graph element were 2000 and NC. The observations that would be used in the drill-down graph would include those with YEAR=2000 and STATE=NC.

The following example shows how the DDLEVEL n parameter can be used to specify the default behavior for the first drill-down level.

```
ods html file=odsout
  parameters=("drilldownmode"="local"
             "ddlevel1"="{chart}{vbar3d}
                       {g_dep}{sales}
                       {g_indep}{year} |
                       {g_indepv}" );
```

As the example shows, the value of the DDLEVEL n parameter is divided into two halves, which are separated by a vertical bar character. The drill-down graph is configured in the syntax that appears before the vertical bar character (|). After the vertical bar, drill-down tags specify how the data from the previous level of drill-down is to be subset for use in the current drill-down graph.

In the preceding example, the first drill-down level (DDLEVEL1) is configured as a three-dimensional vertical bar chart. The dependent variable is SALES and the independent variable is YEAR. The G_INDEPV tag specifies that the data is to be subset based on the independent variable in the previous graph that was selected by the Web user. In our example, the independent variable in the initial graph is REGION. If the Web user selects a graph element that describes the WEST region, then the data will be subset such that the drill-down graph will contain only those observations for which the value of the REGION variable is WEST.

If you do not specify a role for a variable, then that variable does not appear in the drill-down graph. If you do not specify variables for the G_DEP and G_INDEP tags, then the Graph Applet uses the independent and dependent variables of the graph in the preceding drill-down level.

You can explicitly remove a variable role (such as group or subgroup) from the drill-down graph by specifying a \$ character as the drill-down value, as in

```
{G_GROUP} {$}
```

Web users can make this change in the Graph Applet menus by selecting the None option from the list of variables that can be applied to a given variable role. Note that you cannot assign a \$ to the G_INDEP and G_DEP variables, because they must always be present in the drill-down graph.

Configuring the Script Drill-Down Mode

You can use the parameters DRILLDOWNMODE, DRILLFUNC, PATTERNSTRIP, and DRILLTARGET to configure the Script drill-down mode for the Graph Applet, Map Applet, or ActiveX Control. The Script drill-down mode enables you to execute a

JavaScript callback method in response to drill-down actions. You use PUT statements to write the callback method into the HTML output file. Some experience with JavaScript is therefore required.

The syntax used to implement the Script drill-down mode is specified in the ODS statement as follows:

ODS HTML

```
PARAMETERS=("DRILLDOWNMODE"="SCRIPT" "DRILLFUNC"="method");
```

The applet parameter DRILLDOWNMODE (see “Setting the Drill-Down Mode” on page 401) establishes the Script drill-down mode. The DRILLFUNC parameter specifies the name of the JavaScript callback method that will be executed in response to drill-down actions.

In response to a drill-down action, the applet or control generates an array of arguments that is to be passed into the callback method. The array contains all of the data that is associated with the selected graph element. The callback method can make use of any of the data in the array as it generates its output. As the callback method terminates, it may return an object. The applet or control ignores this object.

To invoke the callback method, the applet or control issues `netscape.javascript.JSObject.call()`, in the following form:

```
PUBLIC OBJECT CALL(String method-name, Object argument-array-name[])
```

The *method-name* argument is the name of the callback method that you define in JavaScript in your program. The applet or control supplies the *argument-array-name*.

Working with the Array of Arguments

Understanding the structure of the array of arguments is important for you to be able to access those elements in your callback method. The elements in the array represent all of the variables and values that are represented by the graph element that was selected in the drill-down action. The data is labeled in the array using drill-down tags. The tags identify variable roles or labels and values. For details, see “Using Drill-Down Tags” on page 412 and “Understanding Variable Roles” on page 411.

The first element in the array of arguments is the name of the applet or control. The second element in the array is the name of a file. The name of that file is derived from the variable roles in the graph at the preceding drill-down level, using the following substitution string:

```
{&G_INDEPV, f}  
{&G_GROUPV, f}  
{&G_SUBGRV, f}.html
```

The filename is a concatenation of the formatted values of the independent, group, and subgroup variables in the graph at the preceding drill-down level.

Note: The file name and file type are provided as a convenience. If you use this file name and file type, then it is up to you to create the actual file and to provide its content. \triangle

The remaining elements in the array consist of drill-down tags, and the data that is associated with those tags in the graph element that was selected in the drill-down action. Each variable is represented by triplet pairs of arguments in the array, in the following sequence:

```
tag variable_name  
tagV variable_value  
tagV,F formatted_value
```

Thus, each data value in the selected graph element is represented by six arguments in the array.

For example, assume that the graph shown in Display 11.1 on page 403 is configured for Script drill-down mode. Selecting the east region sales figures for the state of North Carolina generates the following array:

```
[appletName East1998NC.html
G_DEP Sales G_DEPV 10000 G_DEPV,F $10,000
G_INDEP Region G_INDEPV East G_INDEPV,F East
G_GROUP Year G_GROUPV 1998 G_GROUPV,F 1998
G_SUBGR State G_SUBGRV NC G_SUBGRV,F NC]
```

The output filename is East1998NC.html. The remaining triplet pairs would capture the roles and values of the variables that make up the selected graph element. Note that all variable names are case-sensitive as they appear in the array. For example, the value Region is capitalized. This would be the case only if the variable name was defined as Region in the DATA step.

Generating a JavaScript

To implement Script drill-down mode, you use PUT statements in a DATA step to write a JavaScript callback method into the HTML output file. To see an example that shows you how to use PUT statements to generate JavaScript, see “Script Drill-Down Mode” on page 416.

For information on writing JavaScript, refer to the many JavaScript tutorials that are available on the Internet.

Configuring the URL Drill-Down Mode

You can use the parameters DRILLDOWNMODE and DRILLTARGET to configure the URL drill-down mode for the Graph Applet, Map Applet, or ActiveX Control.

The URL drill-down mode uses the HTML= option to name a link variable that provides drill-down URLs. This mode is implemented in a manner that is similar to the type of drill-down that is available for the GIF, JPEG, and PNG device drivers, except that in this case, the applet or control associates drill-down URLs with graph elements without using an image map.

To configure the URL drill-down mode, you need to:

- 1 Specify the device driver. Choose JAVA or ACTIVEX. Set any other global options.

```
goptions reset=global device=java;
```

- 2 Close the ODS listing destination to conserve resources.

```
ods listing close;
```

- 3 Open an HTML output file in ODS and specify URL drill-down mode.

```
ods html file='C:\web\UrlDrill.htm' style=watercolor;
```

Note that you need not specify the applet parameter DRILLDOWN=URL. This drill-down mode is invoked by default when you specify the statement option HTML= (as shown in step 5).

- 4 Specify drill-down URLs by adding a link variable to your data set. See “Adding Links with the HTML= and HTML_LEGEND= Options” on page 574.
- 5 Specify a SAS/GRAPH procedure to generate the graph. Specify the statement option HTML= to identify the link variable. Note that you cannot use the HTML_LEGEND= option for this purpose with these device drivers.

```
proc gchart data=regsales;
  vbar3d region / sumvar=sales
  patternid=midpoint
  html=rpt;
```

- 6 Close the HTML output file and reopen the ODS listing destination.

```
ods html close;
ods listing;
```

To see an example program that implements the URL drill-down mode, see “URL Drill-Down Mode” on page 417.

Configuring the HTML Drill-Down Mode

You can use the parameters DRILLDOWNMODE, DRILLPATTERN, PATTERNSTRIP, and DRILLTARGET to configure the HTML drill-down mode for the Graph applet, Map applet, and the ActiveX Control.

In the HTML drill-down mode, the applet or control responds to drill-down actions by constructing a uniform resource locator (URL) using the data in the selected graph element. The applet then passes the URL to the Web browser for display.

The applet parameter DRILLDOWNMODE (see “Setting the Drill-Down Mode” on page 401) establishes the HTML drill-down mode. The PATTERNSTRIP parameter (see “Removing Blank Spaces from Data Values” on page 414) can be used to selectively remove blank spaces from data values before those values are applied to the URL. The DRILLTARGET parameter (see “Configuring the Drill-down Response” on page 413) enables you to specify where you want the drill-down graph to appear in the browser.

The DRILLPATTERN parameter is specified as follows in the ODS statement:

ODS HTML

```
PARAMETERS=("DRILLDOWNMODE"="HTML"
  "DRILLPATTERN"="URL-with-substitution-strings");
```

An example of this statement might look like this:

```
ods html file=statepop.htm
  parameters= ("DRILLDOWNMODE"="HTML"
    "DRILLPATTERN"='http://www.state.{&statename}.us');
```

In the preceding example, the value of the data set variable STATENAME completes the drill-down URL.

When ODS is configured as shown above, the applet or control dynamically generates URLs in response to drill-down actions. The applet or control replaces the substitution strings with data values from the graph element that was selected in the drill-down action. The *URL-with-substitution-strings* can include multiple substitution strings. Substitution strings can include combinations of variable names, variable roles or labels, and drill-down tags. For details, see “Variables as Substitution Strings” on page 410 and “Drill-Down Tags as Substitution Strings” on page 411. All substitution strings are enclosed in brackets ({ and }) and begin with an ampersand character (&).

Variables as Substitution Strings

When you specify a variable name as a substitution string in the HTML drill-down mode, the applet or control replaces the entire string with the value of the variable as it is specified in the selected graph element. The syntax of the substitution string is as follows:


```
{&variable_name}
```

Because JavaScript is case-sensitive, the name of the variable must be specified exactly as it is specified in the data set.

An example of a variable name substitution string might look like this:

```
http://ourweb.com/uspop/{&statename}/poptable.htm
```

The substitution string above could be used in a Web presentation that begins with a map of the United States. In response to a drill-down action in HTML mode, the value of the STATENAME variable for the selected state would be substituted into the URL. The resulting URL would point to a Web page that contains a table of population information for the selected state.

Drill-Down Tags as Substitution Strings

In the HTML drill-down mode, you can specify variable roles or labels as substitution strings, using drill-down tags, as described in “Understanding Variable Roles” on page 411 and “Using Drill-Down Tags” on page 412. The syntax of these substitution strings is as follows:

```
{&drill-down-tag}
```

The *drill-down-tag* specifies a variable role or label in the initial graph. The applet or control replaces the substitution string by deriving a variable name from the role or label, and applying the value of that variable to the URL. The value is taken from the data that is associated with the selected graph element.

For example, assume that a Web presentation was configured for HTML drill-down mode using the following value of the DRILLPATTERN parameter:

```
http://ourweb.com/regstaff/{&G_INDEPV}/stafflist.htm
```

Now assume that a Web user selects an element in the graph. If the independent variable for that graph was the variable REGION, and if the value of the REGION variable in the selected graph element was East, then the applet would display the following URL:

```
http://ourweb.com/regstaff/East/stafflist.htm
```

The default value for the DRILLPATTERN parameter is as follows:

```
{&G_INDEPV,f}{&G_GROUPV,f}{&G_SUBGRV,f}.html
```

The URL that results from this specification points to an HTML file in the same directory as the top-level HTML file. The name of the file consists of a concatenation of formatted values for the first independent, group, and subgroup variables that are defined in the data set.

To see an example program that uses the HTML drill-down mode, see “HTML Drill-Down Mode” on page 419.

Understanding Variable Roles

The assignment of roles to variables determines the appearance of the resulting graph. The assignment of roles takes place in the SAS/GRAPH statement that generates the graph. One variable is always assigned the role of independent variable, and another is always assigned the role of dependent variable. Once the initial graph has been displayed in the applet or control, Web users can change the variable roles using menu options.

Variable roles are used to configure the Local, HTML, and Script drill-down modes. The roles are assigned with parameters, using the PARAMETERS= option on the ODS

statement. In the specification of a parameter, the assignment of roles is done with drill-down tags (see “Using Drill-Down Tags” on page 412).

Using Drill-Down Tags

You can use the following tags to specify drill-down behavior for the Graph applet, Map applet, or ActiveX control. The following table defines the drill-down tags and explains the types of graphs to which the tags can be applied.

Table 11.1 Drill-Down Tags Used by the Graph Applet, Map Applet, and ActiveX Control

Tag Name	Tells the Applet to...	Definition of the Value That Follows the Tag	Applied in
G_COLOR	Use new colors for the graph elements	Name of the new color variable	Scatter plots
G_COLORV	Use the color variable from the preceding level	None	Scatter plots
G_DEP	Use a new dependent variable	Name of the new dependent variable	All charts
G_DEPV	Use the dependent variable from the previous level	None	All charts
G_DEPTH	Use a new depth variable	Name of the new depth variable	Vertical bar charts and scatter plots
G_DEPTHV	Use the depth variable that was used in the previous level	None	Vertical bar charts and scatter plots
G_GROUP	Use a new group variable	Name of the new group variable	Bar charts
G_GROUPV	Use the group variable that was used in the previous level	None	Bar charts
G_INDEP	Use a new independent variable	Name of the new independent variable	Charts and maps
G_INDEPV	Use the independent variable that was used in the previous level	None	Charts and maps
G_LABEL	Use a new label	Name of the new label (mapID) variable	Maps
G_LABELV	Use the same label that was used in the previous level	None	Maps

Tag Name	Tells the Applet to...	Definition of the Value That Follows the Tag	Applied in
G_SUBGR	Use a new subgroup variable	Name of the new subgroup variable	Bar charts and scatter plots
G_SUBGRV	Use the same subgroup variable that was used in the previous level	None	Bar charts and scatter plots

When you specify a variable name after a tag, that name must be specified exactly the way it appears in the data set, because variable names are case-sensitive in JavaScript. To find out how a variable was defined in the data set, use the CONTENTS procedure.

You can explicitly remove a tag by specifying a \$ for its value, which is the same as setting it to None using the menu of the applet or control. This removes from the graph the data and axis label that would otherwise be included in the graph. Note that you cannot set the values of the G_INDEP and G_DEP tags to None because they are always represented in the graph.

For Script drill-down mode only, you can specify that data values are to be formatted or not formatted. By default, the values of the variables are not formatted. If the characters , **f** are appended to the end of the tag, then those values will be presented in formatted form. For example, the following tag specifies that the values of the independent variable **cost** are to appear in formatted form:

```
{g_indep,f}{cost}
```

The format is applied using the FORMAT statement in the DATA step or graphics procedure that generated the data for the graph. Formatted values are specified in the statement that generated the original graph. For example, assume that the DOLLAR5.2 format was specified for the **cost** variable. If the value of the **cost** variable is 10, its unformatted value would be 10, and its formatted value would be \$10.00. Formatted values are used for axis labels, legends, and data tips that are displayed when the mouse is positioned over a graph element.

You may also append , **n** to tags that reference variables whose values are URLs. Normally, the substitution string is URL-encoded for browsers that do not support embedded white space in URL strings. Use , **n** to prevent this encoding. Note that using , **n** is not the same as using the applet parameter PATTERNSTRIP, which removes blank spaces from data values before those values are applied to substitution strings.

No intervening white space should be added between the primary tag and the appended , **f** or , **n** characters.

Configuring the Drill-down Response

In the HTML and URL drill-down modes, you can specify the parameter DRILLTARGET to specify where you want the Web browser to display drill-down graphs. By default, the applet or control displays drill-down graphs in a new Web browser window.

Specify the DRILLTARGET parameter as follows using the PARAMETERS= option in the ODS statement:

```
ODS HTML
```

```
PARAMETERS=("DRILLTARGET"=
  "_BLANK" | "_SELF" | "_PARENT" | "_TOP" | any_named_target)
```

_BLANK

displays the drill-down graph in a newly opened, unnamed browser window.

_SELF

displays the drill-down graph in the same frame or window as the initial graph. This is the default behavior in most browsers.

_PARENT

displays the drill-down graph in the parent frame in a frameset. If no frames are defined, this value is the same as **_SELF**.

_TOP

displays the drill-down graph in the full browser window, thereby replacing any frames that were defined in that window.

any_named_target

displays the drill-down graph in the appropriately named frame or browser window.

Note: If a Netscape browser is configured to launch with a blank page (using **_BLANK**) in a new browser window, that page will not receive the access authorizations that existed in the previous window. If the Netscape setting is changed to launch Netscape with the "Last page visited," then the page does receive the access authorizations. △

Removing Blank Spaces from Data Values

The drill-down modes Script (see "Configuring the Script Drill-Down Mode" on page 407) and HTML (see "Configuring the HTML Drill-Down Mode" on page 410) make use of substitution strings to generate a response to drill-down actions. The substitution strings are replaced with data values. Blank spaces in those data values can produce unexpected results. To remove blank spaces from data values when those values are to be used in a substitution string, specify the **PATTERNSTRIP** parameter as follows in the ODS statement:

```
ODS HTML FILE=fileref-or-external-file
  PARAMETERS=("DRILLDOWNMODE"="SCRIPT | URL"
    "PATTERNSTRIP"="NONE | YES | COMPRESS");
```

NONE

is the default value. Any blank spaces in the data value are inserted into the substitution string.

YES

strips all blank spaces from the end of the data value, but retains blank spaces elsewhere.

COMPRESS

removes all blank spaces from the data value, wherever they occur.

Disabling the Drill-Down Functionality

For the Graph applet, you can specify the **DISABLEDRILLDOWN** parameter to disable the drill-down functionality. Specify the **DRILLDOWNMODE** parameter as follows in the ODS statement:

```
ODS HTML
  PARAMETERS=("DISABLEDRILLDOWN"="TRUE");
```

Specifying this parameter disables the default Local drill-down mode.

Examples of Interactive Java Output

Local Drill-Down Mode

The following example generates an HTML output file that runs the Graph applet. The applet automatically generates and displays drill-down graphs based on the element in the graph that was selected with a click of the left mouse button. In the example, note how variable roles are assigned in the VBAR3D statement.

This example is available in the Sample Library under the name GWBJALOC. For further information, see “Configuring the Local Drill-Down Mode” on page 401.

```
filename odsout 'your-path-and-filename.htm';

data sales;
  length Region $ 4 State $ 2;
  format Sales dollar8.;
  input Region State Sales Year Qtr;
  datalines;
West CA 13636 1999 1
West OR 18988 1999 1
West CA 14523 1999 2
West OR 18988 1999 2
East MA 18038 1999 1
East NC 13611 1999 1
East MA 11084 1999 2
East NC 19660 1999 2
West CA 12536 1998 1
West OR 17888 1998 1
West CA 15623 1998 2
West OR 17963 1998 2
East NC 17638 1998 1
East MA 12811 1998 1
East NC 12184 1998 2
East MA 12760 1998 2 ;

goptions reset=all device=java;

ods listing close;

ods html file=odsout
  style=gears;

title1 'Company Sales, Mid Year';

proc gchart data=sales;
  vbar3d region / sumvar=sales
  group=year subgroup=state;
run; quit;
```

```
ods html close;
ods listing;
```

Script Drill-Down Mode

The following example shows you how to implement the Script drill-down mode in the Graph applet or Map applet. The program generates a map of the United States that responds to drill-down actions by displaying abbreviated state names. In the example, note how PUT statements are used to insert JavaScript into the ODS output file.

This example is available in the Sample Library under the name GWBSCDRL.

For further information, see “Configuring the Script Drill-Down Mode” on page 407.

```
/* Change the next two lines to run this program. */
filename odsout 'html-output-file' ;
libname maps 'map-data-library';

/* Create a data set that contains the US states. */
proc sql;
create table work.mydata as
select unique state from maps.us;
quit;

/* Add state abbreviations to the new data set. */
data work.mydata;
length Statename $2;
set work.mydata;
Statename=trim(left(uppercase(fipstate(state))));
run;

/* Specify the JAVA driver and the graph size. */
goptions reset=all device=java
          xpixels=325 ypixels=225 ;

/* Specify the HTML output file, the script */
/* drill-down mode, and the callback method. */
ods html file=odsout
          parameters=("DRILLDOWNMODE"="Script"
                    "DRILLFUNC"="MapDrill");

/* Specify a map title and generate the map. */
title1 "State Abbreviations";
proc gmap map=maps.us data=work.mydata all;
  id state;
  choro statename / nolegend;
run;
quit;

/* Close the HTML destination and */
/* open the listing destination. */
ods html close;
ods listing;

/* Create the MapDrill script that is specified on */
```

```

/* the ODS HTML statement's DRILLFUNC parameter. */
/* Write the script to the same file that contains */
/* the HTML output from the GMAP procedure. */
data _null_ ;
file odsout mod; /* Modify the file rather than replacing it. */
put ' ' ;
put '<SCRIPT LANGUAGE="JavaScript">' ;
put 'function MapDrill( appletref )' ;
put '{' ;
put ' ' ;
put '/* Open an alert box to show the abbreviated state name. */ ' ;
put 'for(i = 2; i < MapDrill.arguments.length; i += 2)' ;
put ' {' ;
put '     if (MapDrill.arguments[i] == "G_DEPV,f" )';
put '         alert(MapDrill.arguments[i+1]);' ;
put ' }' ;
put ' ' ;
put '}' ;
put '</SCRIPT>';
run ;

```

URL Drill-Down Mode

The following example demonstrates URL drill-down mode. This example is available in the SAS sample library under the name `GWBURLDR`. For further information, see “Configuring the URL Drill-Down Mode” on page 409.

```

/* Enter the web-output-path. */
filename urldrill 'web-output-path';
filename sales 'web-output-path\sales.html';
filename central 'web-output-path\central.html';
filename south 'web-output-path\south.html';
filename west 'web-output-path\west.html';

/* Close the ODS listing destination to conserve resources. */
ods listing close;

/* Specify the device driver. */
goptions reset=global device=java;

/* Create the data set REGSALES. */
data regsales;
    length Region State $ 8;
    format Sales dollar8.;
    input Region State Sales;

/* Initialize the link variable. */
    length rpt $40;

/* Assign values to the link variable. */
if Region='Central' then
    rpt='href="central.html"';
else if Region='South' then

```

```

        rpt='href="south.html"';
    else if Region='West' then
        rpt='href="west.html"';

datalines;
West CA 13636
West OR 18988
West WA 14523
Central IL 18038
Central IN 13611
Central OH 11084
Central MI 19660
South FL 14541
South GA 19022
;

/* Open the HTML output file and specify the URL drill-down mode. */
ods html body=sales
    path=urldrill
    style=money
    parameters=("drilldown"="url");

/* Create a chart that uses the link variable. */
title1 'Company Sales';
proc gchart data=regsales;
    vbar3d region / sumvar=sales
    patternid=midpoint
    html=rpt;
run;
quit;

/* Create an HTML file for central sales drill-down. */
ods html body=central path=urldrill style=money;

/* Generate the first drill-down report. */
title1 'Central Sales';
proc print data=regsales noobs;
    var state sales;
    where region='Central';
run;

/* Create an HTML file for southern sales drill-down. */
ods html body=south path=urldrill style=money;

/* Generate the second drill-down report. */
title1 'Southern Sales';
proc print data=regsales noobs;
    var state sales;
    where region='South';
run;

/* Create an HTML file for western sales drill-down. */
ods html body=west path=urldrill style=money;

```



```

/* Generate the third drill-down report. */
title1 'Western Sales';
proc print data=regsales noobs;
    var state sales;
    where region='West';
run;
quit;

/* Close the HTML destination and open the listing destination. */
ods html close;
ods listing;

```

HTML Drill-Down Mode

The following example generates an HTML output file that displays the Map applet. The applet is configured for HTML drill-down mode, where drill-down URLs are dynamically generated based on the data in the graph element that was selected in the drill-down action. In the example, note how the value of the STATENAME variable is used to complete drill-down URLs.

This sample is available in the SAS Sample Library under the name GWBJAMAP. For further information, see “Configuring the HTML Drill-Down Mode” on page 410.

```

/* Close the ODS listing destination to conserve resources. */
ods listing close;

/* Specify a path and name for the HTML output file. */
ods html
    file="your_web_path/your_HTML_file.htm"
    parameters=("DRILLDOWNMODE"="HTML")
    parameters=("DRILLPATTERN"='http://www.state.{&statename}.us')
    parameters=("BACKCOLOR"="FFFFFF");

/* Specify the Java driver and set up customizations. */
goptions reset=all      device=java      cback=white
        border          gunit=pct       htext=3
        xpixels=500     ypixels=350

/* Specify colors for map regions. */
colors=(
cxa5c09d,cxff358f,cx0431f8,cxfffff0,cxd3a4ef,cxff8287,
cxd3a4ef,cxffc2d3,cx0431f8,cxffffc4,cx00edcf,cxcd7384,
cxf0eded,cx999797,cxa5c09d,cx008080,cxfabc46,cxff358f,
cx3f769a,cxff8600,cx45ab90,cxca4db0,cxf6d3a5,cx274776,
cxff72b0,cxb0c1f4,cx7dff88,cx4a97ed,cxed5662,cxffff81,
cx922e83,cxa5c09d,cxff358f,cx0431f8,cxfffff0,cxd3a4ef,
cxff8287,cx5d55b3,cxffc2d3,cx0431f8,cxffffc4,cx00edcf,
cxcd7384,cxf0eded,cxca4db0,cx5d55b3,cx008080,cxfabc46,
cxd3a4ef,cx3f769a,cx5d55b3 );

/* Create data for the graph. */
proc sql;
    create table work.mydata as
    select unique state from maps.us;

```

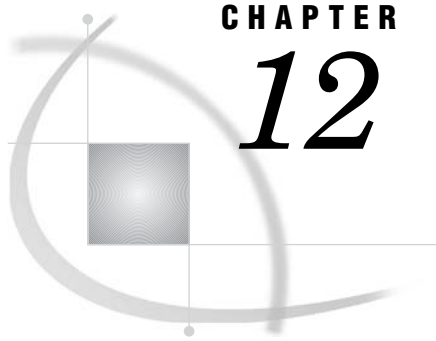
```
quit;
run;
data work.mydata;
  length statename $1020;
  set work.mydata;

/* Place the state name in the data set. */
  statename=trim(left(lowercase(fipstate(state))));
run;

title1
  "Click on a state to go to that state's home page";

/* Generate the graph. */
proc gmap map=maps.us
  data=work.mydata all;
  id state;
  choro statename / levels=1 discrete
    outline=black
    nolegend
    des='US Government Web Sites'
    name='usgov';
  run; quit;

/* Close the HTML output file and open the listing destination. */
ods html close;
ods listing;
```



CHAPTER

12

Attributes and Parameters for Java and ActiveX

<i>Specifying Parameters and Attributes for Java and ActiveX</i>	421
<i>Specifying the Location of Control and Applet Files (CODEBASE= and ARCHIVE= Options)</i>	422
<i>Specifying the Location of the ActiveX Control</i>	422
<i>Specifying the Location of the Java Applets</i>	423
<i>Specifying the CODEBASE= URL</i>	423
<i>Specifying the Location of the Java Plug-In (CODEBASE= Attribute)</i>	424
<i>Parameter Reference for Java and ActiveX</i>	424
<i>Parameter Definitions</i>	427

Specifying Parameters and Attributes for Java and ActiveX

You can specify attributes and parameters in ODS to override default values in Java and ActiveX. No attributes or parameters are required. SAS provides workable defaults in most cases.

Attributes can be any HTML name/value pair that is valid inside the initial (opening) OBJECT tag. Parameters are values that appear in the body of the OBJECT tag, to configure the appearance or functionality of a Java applet or the ActiveX control.

Attributes and parameters are specified as options of one of the available ODS statements, such as ODS HTML:

ODS HTML

```
<ATTRIBUTES=('attr-name'='attr-value')>
<PARAMETERS=('param-name'='param-value')>
<other-options>;
```

The preceding syntax applies to all applicable ODS statements, such as HTMLCSS, MARKUP, PDF, PS, and RTF.

You can specify more than one name/value pair (separated by blank spaces) inside the parenthesis of an ATTRIBUTES= or PARAMETERS= option. You can also specify multiple ATTRIBUTES= and PARAMETERS= options in a single ODS statement. These options can be specified in any order in the ODS statement.

Most examples in the following topics specify parameters:

- “Examples of Interactive Java Output” on page 415
- “ActiveX Examples” on page 393

For information on other ODS statement options, see the *SAS Output Delivery System: User's Guide*.

In HTML output that runs an applet or a control, all values of the ATTRIBUTES= option appear in the opening OBJECT tag. For example, a SAS/GRAPH program can specify the WIDTH attribute:

```
ods html file='C:\sashtml\piechart.htm'
  attributes=('width'=720');
```

In the HTML output file, the WIDTH attribute appears inside the beginning OBJECT tag:

```
<OBJECT height=480 width=720>
/* parameters and graph info here */
</OBJECT>
```

Valid attribute names are those that are enabled for the OBJECT tag in HTML. Valid attributes must also be specified as required by Java or ActiveX device drivers that run in the operating environment.

All of the name/value pairs that are specified in the ODS statement option PARAMETERS= appear in the body of the OBJECT tag. For example, the BACKIMAGE parameter provides a path to an image that the applet or control displays behind the graphical output in the browser window.

Valid parameter values for the ActiveX control, Graph applet, Map applet, and Contour applet are defined in “Parameter Reference for Java and ActiveX” on page 424. Parameters for other applets, such as Metaview, are provided in the sections that apply to those applets, as in “Metaview Applet Parameters” on page 475.

Specifying the Location of Control and Applet Files (CODEBASE= and ARCHIVE= Options)

When you generate Web presentations with the Java and ActiveX device drivers, SAS generates HTML pages that automatically look for the Java archive files or the ActiveX control file in the default installation location. If you install the ActiveX control .exe file or the Java archive .jar files in a location other than the default or if you want to publish Output Delivery System (ODS) output containing the SAS/GRAPH control or the applets in a Web server, then you may need to specify the location of the .exe file or the .jar files when you generate your Web presentation.

You can use the CODEBASE= option to specify the location of the ActiveX control or the Java applets. You can use the ARCHIVE= option to specify the name of the Java archive file.

Note: The ActiveX control must be installed locally on each PC where the web presentation will be viewed. \triangle

Specifying the Location of the ActiveX Control

If you use the ActiveX device driver with ODS to generate output containing an ActiveX control, then specify the location and version of the .exe file with the CODEBASE= option on the ODS statement. Specify the directory and filename of the .exe file. (The default filename is **sasgraph.exe**.) The CODEBASE location may be specified as a path name or as a URL. (See “Specifying the CODEBASE= URL” on page 423 for more information.) If you have installed previous versions of the ActiveX control, then you also need to specify the version that you want to use. For example, if your .exe file is in **/sasweb/graph**) you would specify

```
ods html file="/path/to/mygraph.html"
  codebase="/sasweb/graph/sasgraph.exe#version=9,1,0,304";
```

If you use the DS2GRAF macro to generate output containing an ActiveX control, then specify the location and version of the `.exe` file with the CODEBASE= macro argument. Use the %STR function to enclose the argument value. For example:

```
%ds2graf(codebase=%str(http://web_server_name/sasweb/graph/sasgraph.
exe#version=9,1,0,304),htmlfile=your_path_and_filename.htm
...

```

Specifying the Location of the Java Applets

By default, the location of the SAS Java archive files is specified by the APPLETLOC= system option. This value is the default value of the CODEBASE= parameter. If the default location is accessible by users who will be viewing your Web presentation, and the SAS Java archive is installed at that location, then you do not need to change the value of the CODEBASE= parameter.

If you use the Java device driver with ODS to generate output containing a SAS/GRAPH applet, then specify the path to the `.jar` file with the CODEBASE= option on the ODS statement. Specify only the directory of the `.jar` file. The CODEBASE location may be specified as a path name or as a URL. (See “Specifying the CODEBASE= URL” on page 423 for more information.) For example, if your `.jar` file is in `/sasweb/graph`, you would specify

```
ods html file="/path/to/mygraph.html"
codebase="/sasweb/graph";

```

The ARCHIVE= option specifies the file name of the `.jar` file. You do not need to specify the ARCHIVE= option on the ODS statement unless you have renamed the `.jar` files.

For applets generated with macros, specify the CODEBASE= argument for the macro. For example:

```
%ds2const(codebase=http://your_path_to_archive,
htmlfile=your_path_and_filename.htm
...
);

```

For the DS2GRAF, DS2CSF, and META2HTM macros, you must also specify the ARCHIVE= macro argument. For example:

```
%meta2htm(htmlhref=_webout,openmode=replace,
codebase=http://web_server_name/sasweb/graph
archive=metafile.zip);

```

For the DS2TREE and DS2CONST macros, you do not need to specify the ARCHIVE= argument unless you have renamed the `.jar` files.

Specifying the CODEBASE= URL

If the value that you specify for CODEBASE= is a URL, it can be a full URL (for example, `http://your_server/sasweb/graph`), or it can be relative to your Web server (`/sasweb/graph`). If you are publishing HTML only on Web servers where the control or the applets are installed in a common location, it is generally recommended that you use the shorter, relative URL. A relative URL will allow you to move the HTML to any Web server without modifying the HTML (assuming the control or the applets are installed on that server). If you are creating HTML to be viewed directly via a `file:` URL, sent

by e-mail or copied to Web servers without the control or the applets installed, then you should use a full URL to point to the applet `.jar` files at a known location.

Specifying the Location of the Java Plug-In (CODEBASE= Attribute)

The `CODEBASE=` attribute on the ODS statement specifies the location of the Java plug-in from Sun Microsystems. By default, SAS points to the Web site of the Java plug-in from Sun Microsystems. If necessary, you can change the location of the Java plug-in by specifying the `CODEBASE=` attribute on the ODS statement. For example:

```
ods html file='c:\myfile.htm'
      attributes=('codebase='http://www.ourco.com/ourPlugin/j2re--1_4_1--windows-i586.exe');
```

On Windows systems, the user is prompted to install the plug-in if it is not already installed. On other systems, the plug-in can be installed from the Sun Microsystems site (<http://www.sun.com>) or from one of the SAS Third Party Software Components CDs.

Parameter Reference for Java and ActiveX

The following table lists the parameters that you can specify in programs that use the `JAVA` and `ACTIVEX` device drivers. Output from the `JAVA` device driver runs in the Graph applet, Map applet, or Contour applet. Output from the `ACTIVEX` device driver runs in the SAS/GRAPH Control for ActiveX.

For information on parameters for other applets, see the sections that apply to those applets, such as “Metaview Applet Parameters” on page 475.

Parameter definitions appear after the following table.

Table 12.1 Parameters Enabled for Java and ActiveX

Parameter	ActiveX	Graph Applet	Map Applet	Contour Applet
AMBIENT on page 427	♦			♦
BACKCOLOR on page 427				♦
BACKDROP COLOR on page 427				♦
BACKIMAGE on page 427	♦	♦	♦	♦
CLIPTIPS on page 427				♦
COLORNAMELIST on page 427				♦
COLORNAMES on page 427		♦		♦
COLORSCHEME on page 428	♦	♦		
DDLEVELn on page 428	♦	♦	♦	
DIRECT on page 428	♦			♦
DRAWIMAGE on page 428		♦	♦	♦
DRAWMISSING on page 428				♦
DRAWSIDES on page 428				♦
DRILLDOWNFUNCTION on page 428	♦	♦	♦	

Parameter	ActiveX	Graph Applet	Map Applet	Contour Applet
DRILLDOWNMODE on page 429	◆	◆	◆	
DRILLPATTERN on page 429	◆	◆	◆	
DRILLTARGET on page 429	◆	◆	◆	
DUPLICATEVALUES on page 429				◆
FILLPOLYGONEDGES on page 430				◆
FREQNAME on page 430		◆		
G_COLOR on page 430	◆	◆	◆	
G_COLORV on page 430	◆	◆	◆	
G_DEP on page 430	◆	◆	◆	
G_DEPTH on page 430	◆	◆	◆	
G_DEPTHV on page 430	◆	◆	◆	
G_DEPV on page 430	◆	◆	◆	
G_GROUP on page 431	◆	◆	◆	
G_GROUPV on page 431	◆	◆	◆	
G_INDEP on page 431	◆	◆	◆	
G_INDEPV on page 431	◆	◆	◆	
G_LABEL on page 431			◆	
G_LABELV on page 431			◆	
G_SUBGR on page 431	◆	◆	◆	
G_SUBGRV on page 431	◆	◆	◆	
GRADIENTBACKGROUND on page 431	◆	◆	◆	◆
GRADIENTENDCOLOR on page 431	◆	◆	◆	◆
GRADIENTSTARTCOLOR on page 431	◆	◆	◆	◆
HELPLLOCATION on page 432	◆	◆	◆	◆
HONORASPECT on page 432				◆
IMAGEPOSX on page 432		◆	◆	◆
IMAGEPOSY on page 432		◆	◆	◆
LEGENDFIT on page 432				◆
LEGENDFONT on page 432				◆
LEGENDFONTSIZE on page 432				◆
LEGENDHEIGHTPERCENT on page 432				◆
LEGENDPERCENT on page 432				◆
LEVELOFDETAIL on page 432				◆
LEGENDWIDTHPERCENT on page 432				◆

Parameter	ActiveX	Graph Applet	Map Applet	Contour Applet
LIGHTING on page 433				♦
LOADFUNC on page 433		♦		
LOCALE on page 433		♦	♦	♦
LODCOUNT on page 433				♦
MENUREMOVE on page 433		♦		
MINLEGENDFONTSIZE on page 433				♦
MISSINGCOLOR on page 434				♦
NAME on page 434		♦	♦	♦
NAVIGATERENDERMODE on page 434			♦	♦
NOJSOBJECT on page 434		♦		
OUTLINECOLOR on page 434	♦	♦		♦
OUTLINES on page 434				♦
OVERFLOWCOLOR on page 434	♦			♦
PATTERNSTRIP on page 434	♦	♦	♦	
PROJECTION on page 434				♦
PROJECTIONRATIO on page 434				♦
RENDERMODE on page 435				♦
RENDEROPTIMIZE on page 435				♦
RENDERQUALITY on page 435			♦	♦
SHOWBACKDROP on page 435				♦
SHOWLEGEND on page 436				♦
SIMPLEDEPTHSORT on page 436				♦
SIMPLETHRESHOLD on page 436		♦		
STACKED on page 436				♦
STACKPERCENT on page 436				♦
SURFACESIDECOLOR on page 436				♦
TIPBACKCOLOR on page 436				♦
TIPBORDERCOLOR on page 436				♦
TIPS on page 437	♦	♦	♦	♦
TIPMODE on page 437	♦	♦	♦	♦
TIPSTEMSIZE on page 437				♦
TIPTEXTCOLOR on page 437				♦
UNDERFLOWCOLOR on page 437				♦
USERFMTn on page 437				♦
VIEW2D on page 437	♦	♦		♦

Parameter	ActiveX	Graph Applet	Map Applet	Contour Applet
VIEWPOINT on page 438				◆
XBINS on page 437				◆
YBINS on page 437				◆

Parameter Definitions

AMBIENT=*light-level*

specifies the intensity of non-directional ambient light in relation to direct light. Valid values range from 0.0 to 1.0. The default value is 0.4. The sum of direct light (see the DIRECT parameter) and ambient light can never exceed 1.0. Direct light is given priority. If you specify a sum of these two values that is greater than one, the ambient value will be reduced so that the sum of the two values equals one. This parameter is valid in the ActiveX control and for the Contour applet.

BACKCOLOR=*color*

specifies the background color of the applet. The default value is the default window color of the operating system. This parameter is valid only in the Contour applet.

BACKDROP COLOR=*color*

specifies the color of all walls in the applet, including the floor. The default value is white. This parameter is valid only in the Contour applet.

BACKIMAGE=*image-URL*

specifies the URL of the image that is applied to the background of the applet image area. By default, no image is used and the background is drawn in a single solid color. The way that the image will be applied to the background is specified with the DRAWIMAGE parameter. For the ActiveX control, the background image must be in GIF, JPEG, or BMP format. For the Graph, Map, or Contour applet, the URL must be absolute and not relative.

CLIPTIPS=TRUE | FALSE

indicates whether data tips should be clipped. The default value of TRUE does not display data tips when the cursor is outside of the plot area. A value of FALSE displays data tips when the cursor is outside of the plot area. The data tips window hugs the boundary and displays the value of the element that is closest to the cursor along that edge of the plot. This parameter is valid only in the Contour applet.

COLORNAMELIST=*string*

specifies which of two named color lists has priority when searching for named colors. The default is to search the list of HTML 3.2 colors first, followed by the SAS name list. Specifying SAS as the string reverses this priority, giving SAS names higher priority. This parameter is valid only in the Contour applet.

COLORNAMES=*name1=value1,name2=value2, ... nameN=valueN*

specifies the color names and associated 6-digit hexadecimal RGB values that will be displayed in the **Standard Colors** list box in the Color Edit dialog box. In the parameter value, no white space is allowed. The color name can be any valid string, and is displayed as specified in the list box. This parameter is valid in the Graph, Map, and Contour applets.

COLORSCHEME=*scheme-name*

specifies the name of the color scheme that is applied to the graph. By default, no color scheme is applied to the graph. This parameter is valid in the ActiveX control and the Graph applet.

DDLEVEL*n*=*configuration-string*

configures the drill-down graph that is generated at the drill-down level that is specified by the letter *n*. The drill-down graph is configured using drill-down tags such as G_INDEPV. For details, see “Using Drill-Down Tags” on page 412. This parameter is valid in the ActiveX control and in the Graph and Map applets.

DIRECT=*light-level*

specifies the intensity of direct light (from a light source) in relation to the ambient (non-directional) light. Valid values range from 0.0 to 1.0. The default value is 0.6. The sum of direct light and ambient light (see the AMBIENT parameter) cannot exceed 1.0. Direct light is given priority. If you specify a sum of these two values that is greater than one, the level of ambient light will be reduced so that the sum of the two values equals one. This parameter is valid in the ActiveX control and the Contour applet.

DRAWIMAGE=*background-image-application*

specifies how the image specified in the BACKIMAGE parameter is applied to the background of the applet window. This parameter is valid for the Graph, Map, and Contour applets. Here are the valid values:

CENTER

centers a single instance of the image in the background, without resizing the image.

POSITION

places a single instance of the image at the location supplied by the IMAGEPOSX and IMAGEPOSY parameters, without resizing. If these parameters are not specified, then the image is positioned at the top left corner of the applet window.

SCALE

fills the entire background of the applet window with a single instance of the specified image, which is resized as necessary.

TILE

fills the entire background of the applet window using multiple instances of the specified image, without resizing that image. The images are arranged in rows and columns.

DRAWMISSING=TRUE | FALSE

specifies whether missing values should be drawn. By default, missing values are not drawn. Missing values are drawn only when this parameter is set to TRUE and the **styles** menu option is set to Block, Smooth, or Surface. This parameter is valid only in the Contour applet.

DRAWSIDES=TRUE | FALSE

specifies that sides should be drawn when the value of the STACKED parameter is TRUE and when the **styles** menu option is set to Surface, Areas, or LinesAndAreas. The default value is FALSE. To override this parameter, you can specify an ODS style definition. This parameter is valid in the Contour applet.

DRILLDOWNFUNCTION=*function-name***DRILLFUNC**=*function-name*

specifies the name of the JavaScript function that is called in Script drill-down mode. This parameter is valid in the ActiveX control and in the Graph and Map applets.

DRILLDOWNMODE=HTML | LOCAL | SCRIPT | URL

specifies the drill-down mode. This parameter is valid in the ActiveX control and in the Graph and Map applets. Here are the valid values:

HTML

uses a substitution string to dynamically generate a URL based on the selected chart elements, then passes the URL to the browser.

Local mode (Graph applet only)

constructs and displays a new graph based on the data in the previous level of a drill-down graph.

Script mode

invokes the JavaScript function specified in the **DRILLDOWNFUNCTION** parameter, and passes into the function data from the selected graph element.

URL mode

provides static drill-down, using an image map in the HTML file. The image map is generated using the **IMAGEMAP=** and **HTML=** options in **SAS/GRAPH**.

The default drill-down mode is Local for the Graph applet. The Map applet and the ActiveX control do not enable user-selectable drill-down modes.

DRILLPATTERN=*substitution-string*

specifies how to construct the drill-down URL when the drill-down mode is HTML. The substitution string is constructed with drill-down tags, which are expressed in parameters such as **G_DEPV**, as described in “Using Drill-Down Tags” on page 412. This parameter is valid in the ActiveX control and in the Graph and Map applets.

DRILLTARGET=*target*

specifies where the drill-down destination is displayed in the browser. The default target is **_BLANK**, which is an HTML reserved word that displays the drill-down destination in a new browser window. The target can be specified as another reserved target name or as the name of a window or frame in your Web presentation. This parameter is valid in the ActiveX control and in the Graph and Map applets.

DUPLICATEVALUES=*string*

determines how the applet will handle data values for grid positions that already have a data value. This parameter is valid in the Contour applet. Specify one of the following values:

COUNT

stores at each grid location the number of values found for that location.

FIRST

stores the first value found.

LAST

stores the last value found.

MAX

stores the maximum value found.

MEAN

stores the mean (average) of all values found. This is the default value.

MIN

stores the maximum value found.

NMISS

stores the number of missing values found.

RANGE

stores the range of values found. The range is computed as the maximum value minus the minimum value.

SUM

stores the sum of all values found.

FILLPOLYGONEDGES=ALWAYS | NEVER | OS/2

specifies whether to adjust rendering to fix a temporary vendor rendering defect. If you use Netscape on OS/2, then polygon edges do not always fill in correctly. If the value of this parameter equals the `os.name` Java system property, then the Contour applet sets the default value of this parameter to OS/2, which lets `drawPolygon` correctly fill in (render) the polygon edges, yet this extra drawing effort slows performance. If you set this parameter to the value of the parameter of the name of the operating system returned in `os.name`, then the adjusted rendering is performed when the applet runs on that operating system because the applet notifies the Java console. When you set the value to ALWAYS, the adjusted rendering is always performed, regardless of the operating system on which the applet is running. Similarly, if you set the value to NEVER, the adjusted rendering is never performed on any operating system. This parameter is valid only in the Contour applet.

FREQNAME=*variable-name*

specifies a name for a new variable that contains the frequency count when a frequency chart is produced. By default, the name assigned to this variable is "Frequency". This parameter may be overridden if you specify an ODS style definition. This parameter is valid in the Graph applet.

G_COLOR=*variable-name*

specifies a new color variable for the current drill-down level. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_COLORV=*variable-name*

specifies that the current color variable is the same variable that was used to configure the previous drill-down level. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_DEP=*variable-name*

specifies a new dependent variable for the current drill-down level. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_DEPV=*variable-name*

specifies that the drill-down graph at the specified drill-down level is to use the same dependent variable that was used in the previous drill-down level. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_DEPTH=*variable-name*

specifies a new depth variable for the current drill-down level. Drill-down graphs that use this variable can be vertical bar charts or scatter plots. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_DEPTHV=*variable-name*

specifies that the depth variable for the current drill-down level is the same depth variable that was used in the previous drill-down level. Drill-down graphs that use this variable can be vertical bar charts or scatter plots. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_GROUP=*variable-name*

specifies a new group variable for the current drill-down level. Drill-down graphs that use this variable can be bar charts. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_GROUPV=*variable-name*

specifies that this group variable should be the same group variable that was used at the previous drill-down level. Drill-down graphs that use this variable can be bar charts. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_LABEL=*variable-name*

specifies a new label variable for the current drill-down level. Drill-down graphs that use this variable can be maps. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_LABELV=*variable-name*

specifies that this label variable should be the same label variable that was used at the previous drill-down level. Drill-down graphs that use this variable can be maps. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_INDEP=*variable-name*

specifies a new independent variable for the current drill-down level. Drill-down graphs that use this variable can be charts and maps. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_INDEPV=*variable-name*

specifies that an independent variable at the current drill-down level is the same variable that was used at the previous drill-down level. Drill-down graphs that use this variable can be charts and maps. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_SUBGR=*variable-name*

specifies a new subgroup variable for the current drill-down level. Drill-down graphs that use this variable can be bar charts and scatter plots. This parameter is valid in the ActiveX control and in the Graph and Map applets.

G_SUBGRV=*variable-name*

specifies that a subgroup variable at this drill-down level is the same subgroup variable that was used at the previous drill-down level. Drill-down graphs that use this variable can be bar charts and scatter plots. This parameter is valid in the ActiveX control and in the Graph and Map applets.

GRADIENTBACKGROUND=TRUE | FALSE | VERTICAL | HORIZONTAL

specifies that the background of the window is or is not using a color gradient. To override this parameter, you can specify an ODS style definition. This parameter is valid in the ActiveX control and in the Graph, Map, and Contour applets. TRUE and FALSE are valid only for the Graph and Map applets. VERTICAL and HORIZONTAL specify the orientation of the color gradient and are only valid for the Contour applet. This parameter is ignored in the Contour applet if you specify the BACKIMAGE parameter. Use GRADIENTSTARTCOLOR and GRADIENTENDCOLOR to define the colors used to draw the background.

GRADIENTENDCOLOR=*color*

GRADIENTSTARTCOLOR=*color*

specify the start color and the end color when two colors are blended in a gradient across a wall, background, or graph element. The color can be an HTML 3.2 color name or a 6-digit hexadecimal RGB value. This parameter may be overridden if

you specify an ODS style definition. This parameter is valid in the ActiveX control and in the Graph, Map, and Contour applets.

HELPLLOCATION=URL

specifies a non-default location for application-specific help that replaces the default help. The default help location is the SAS Web site. This parameter is valid in the ActiveX control and in the Graph, Map, and Contour applets.

HONORASPECT=TRUE | FALSE

specifies whether the aspect of the data being displayed is or is not honored. The default value FALSE scales the shortest axis (x or y). This parameter is valid in the Contour applet. Note that certain annotations, such as pies, may display differently in the applet than in SAS when the value is FALSE.

IMAGEPOSX=horizontal-pixels

IMAGEPOSY=vertical-pixels

specify the location of the upper-left corner of the background image that is named in the BACKIMAGE parameter. These parameters are ignored unless the value of the DRAWIMAGE parameter is POSITION. Positive pixel values are measured from the top-left corner of the applet window. Negative pixel values are measured from the bottom-right corner of the applet window. These parameters are valid in the ActiveX control and in the Graph, Map, and Contour applets.

LEGENDFIT=TRUE | FALSE

specifies the default size of the font to be used in the legend. Only positive values are valid. This parameter is valid only in the Contour applet.

LEGENDFONT=font

specifies which font to use in the legend. Except for the case, the font name must match the name of a Java font available in the browser. This parameter is valid only in the Contour applet.

LEGENDFONTSIZE=font-size

specifies whether the legend should fit within the height of the contour plot area. By default the legend occupies as much of the applet height as is feasible. If TRUE, the height of the legend is restricted to the height of the contour plot within the legend. When you set this parameter, any value specified for LEGENDHEIGHTPERCENT is ignored. This parameter is valid only in the Contour applet.

LEGENDHEIGHTPERCENT=percentage

restricts the height of the legend to a specified percentage of the height of the Contour applet. A vertical margin is always maintained. Valid values are greater than 0 and less than 100 percent, with the default value being 20. This parameter is valid only in the Contour applet.

LEGENDPERCENT=percentage

specifies how much of the Contour applet space (width) to use as the legend area. Valid values are 0 to 80 percent. The default value is 20. This parameter is valid only in the Contour applet.

LEGENDWIDTHPERCENT=percentage

restricts the width of the legend to a specified percentage of the width of the Contour applet. A horizontal margin is always maintained. Valid values are greater than 0 and up to 80 percent, which the default value being 20. This parameter is valid only in the Contour applet.

LEVELOFDETAIL=TRUE | FALSE

specifies whether the level-of-detail processing should be used when drawing plots. The default value is TRUE, which allows level-of-detail processing. See also the LODCOUNT parameter. This parameter is valid only in the Contour applet.

LIGHTING=HEADLIGHT | OVERHEAD | NORTHEAST | SOUTHEAST

specifies the position of the light source relative to the position of the graph. The default value is HEADLIGHT, which directs two light sources at the graph from the front-center of the screen. This parameter is valid in the Contour applet.

LOADFUNC=*Java-method*

specifies the name of a JavaScript method in the HTML output file that loads values and specifications. This parameter is valid in the Graph applet. This parameter should not be specified if you are using ODS.

LOCALE=*xx_yy*<*_variant*>

specifies the language and country to use when displaying locale-sensitive text. This parameter is valid in the Graph, Map and Contour applets. Here are the values for this parameter, which are java.util locale specifiers:

xx

represents the required two-digit ISO-639 language code, as defined at <http://www.loc.gov/standards/iso639-2/>.

yy

represents the required two-digit ISO-3166 country code, as defined at <http://www.niso.org/standards/resources/3166.html>.

<*_variant*>

represents the optional variant code, which depends on the browser and operating environment. If a variant is specified, the initial underscore character is required.

LODCOUNT=*number-of-cell(s)*

specifies the number of cells to use as the level-of-detail threshold. The default value is 2000. When the number of cells involved in drawing a plot in the applet exceeds this value and level-of-detail processing is on, then some cells are ignored when rendering the plot representation. See also the LEVELOFDETAIL parameter. This parameter is valid only in the Contour applet.

MENUREMOVE=*menu-item(s)*

disables items in the Graph applet menu. Here is the syntax of *menu-item(s)*:

menu1-item<*.menu2-item*... *.menuN-item*, *menu-item2*, ...*menu-itemN*>

In the *menu-item(s)* value, periods (“.”) separate menu levels in menu paths. In menu paths, the menu item that is disabled is the last item in the path. Commas separate menu items and menu paths in a series. Menu items are specified using the text that is displayed by the applet, with blank spaces removed. For example, the menu item Graph Properties would be specified as GRAPHPROPERTIES. To apply the MENUREMOVE parameter, first generate the graph without the MENUREMOVE parameter. Then note the menu paths of the items that you wish to disable. This parameter is valid in the Graph applet.

MINLEGENDFONTSIZE=*font*

specifies the minimum font to be used when attempting to fit the legend in the available applet area. Only positive integers are valid values. This parameter is valid only in the Contour applet.

MISSINGCOLOR=*color*

specifies an HTML 3.2 color name or 6-digit hexadecimal RGB value that is to be used to draw missing values. The default color is black. This parameter is valid in the Contour applet.

NAME=*applet-name*

specifies the name for this instance of the applet. Use this parameter only if you have more than one instance of the APPLET tag in your HTML file, and if you have included your own scripts or DHTML that communicates with or acts on a particular instance of the applet. This parameter may be overridden if you specify an ODS style definition. This parameter is valid in the Graph, Map, and Contour applets.

NAVIGATERENDERMODE=NONE | POINT | SOLID | WIREFRAME

specifies how to render the graph during pan, rotate, and zoom. The default value is WIREFRAME. This parameter is valid when the RENDERQUALITY parameter is set to CUSTOM. This parameter may be overridden if you specify an ODS style definition. This parameter is valid in the Contour applet.

NOJSOBJECT

specifies that no JavaScript callback options can be created or used within the applet. This parameter may be overridden if you specify an ODS style definition. This parameter is valid in the Graph applet.

OUTLINECOLOR=*color*

specifies the HTML 3.2 color name or 6-digit hexadecimal RGB value for the outlines of graph elements. This parameter is valid in the ActiveX control and in the Graph and Contour applets.

OUTLINES=TRUE | FALSE

specifies whether outlines should be drawn for the current contour style. Outlines are drawn when this parameter is TRUE and the **Styles** menu option is set to Area, Block, or Surface. This parameter is valid only in the Contour applet.

OVERFLOWCOLOR=*color*

specifies an HTML 3.2 color name or a 6-digit hexadecimal RGB color for colors that are assigned to data values that exceed the maximum range of colors that have been defined in the color palette. The default value is CYAN. This parameter is valid in the ActiveX control and in the Contour applet.

PATTERNSTRIP=TRUE | FALSE

removes preceding and trailing white space from drill-down substitution patterns before the substituted text is added into a dynamically generated drill-down URL. The default value is FALSE. This parameter is valid in the ActiveX control and in the Graph and Map applets.

PROJECTION=ORTHOGRAPHIC | PERSPECTIVE

specifies the type of projection that is used to draw contours. The default value is ORTHOGRAPHIC. This parameter is valid in the Contour applet.

PROJECTIONRATIO=*plot-size-ratio*

specifies the ratio of the plot area (applet size minus legend reserve) to the longest dimension of the plot. For example, specifying a value of 2.0 means that the area that contains the contour plot is twice the size of the longest plot dimension. This guarantees that the plot will be surrounded by a space that measures half the length of the longest projection (not including axes). The default value is 1.5. Values must be greater than or equal to 1.0. This parameter is valid in the Contour applet.

RENDERMODE=*string*

specifies how to render the contours when you are not navigating (panning, rotating, or zooming) the Contour applet. This parameter is valid only in the Contour applet. In some cases, changing the representation can provide additional information about the image, such as more clearly displaying cell boundaries. Here are the valid values for the polygon representations that determine how the Contour applet image can be drawn:

POINT

draws polygons using only single-pixel points at the polygon vertices.

SOLID

draws filled polygons. This is the default value and the normal representation.

WIREFRAME

draws polygons using only lines to represent their edges.

RENDEROPTIMIZE=ALWAYS | NAVIGATION | NEVER | ONNAVIGATION

sets the default for rendering optimization for the Contour applet. This parameter is valid only in the Contour applet. To correctly render images, the applet must first sort the polygons that comprise the image. Some polygons require additional sorting steps to ensure that they are correctly drawn. In many cases, these additional steps are unnecessary because they only slow applet performance and do not add to image quality. This parameter lets you specify if and when the applet should attempt to optimize or reduce the number of sorting operations to be performed. The RENDEROPTIMIZE parameter is ignored unless you set the RENDERQUALITY parameter to CUSTOM. The default value depends on the value of the RENDERQUALITY parameter.

When the RENDERQUALITY parameter is set to BESTQUALITY, the default value for the RENDEROPTIMIZE parameter is NEVER.

When the RENDERQUALITY parameter is set to FASTERNAVIGATION, the default value for the RENDEROPTIMIZE parameter is ONNAVIGATION.

When the RENDERQUALITY parameter is set to BESTPERFORMANCE, the default value for the RENDEROPTIMIZE parameter is ALWAYS.

RENDERQUALITY=*value*

specifies how two available rendering algorithms, one slower and one faster, are applied to the graph. This parameter may be overridden if you specify an ODS style definition. This parameter is valid for the Map and Contour applets. Here are the valid values:

BESTPERFORMANCE | PERFORMANCE

always uses the faster, less complex rendering algorithm.

BESTQUALITY | QUALITY

always uses the slower, more complex rendering algorithm.

FASTERNAVIGATION | NAVIGATION

uses the faster, less complex rendering algorithm during pan, rotate, and zoom, and uses the more complex algorithm otherwise. This is the default value.

CUSTOM (Contour applet only)

lets the user select individual elements that control speed and quality directly, rather than as a group when rendering an image.

SHOWBACKDROP=TRUE | FALSE

specifies whether all walls (including the floor) should be displayed. This parameter overrides any ODS settings and is valid only in the Contour applet.

SHOWLEGEND=TRUE | FALSE

specifies whether the legend should be displayed. This parameter overrides any ODS settings and is valid only in the Contour applet.

SIMPLEDEPTHSORT=TRUE | FALSE

the default value TRUE indicates that the simpler polygon sorting algorithm is used when rendering the plot. This parameter is valid in the Contour applet.

SIMPLETHRESHOLD=*number-of-elements* | NEVER

specifies an integer for the threshold that is used to determine if the graph should be rendered using simple geometry. For bar charts, simple geometry means that graphical elements are represented as lines. For plots, simple geometry means that graphical elements are represented as plus signs (+).

If the graph contains a number of elements that is greater than the SIMPLETHRESHOLD value, simple geometry is used and the Shape menu is made unavailable. The default value is 500. You can also specify the value NEVER, in which case simple geometry is never used and the Shape menu is always available.

Note that if you select and display a subset of the graph, and if the number of elements in the resulting graph drops below the value of the SIMPLETHRESHOLD parameter, regular markers are drawn and the Shape menu is made available.

This parameter is valid in the Graph applet.

STACKED=TRUE | FALSE

specifies whether the contours should be displayed in stacked form, where height is added to the contour plot based on the contour level. This parameter takes effect only when the Style menu option is set to Areas or LinesAndAreas. The default value of this parameter is FALSE. See also the DRAWSIDES parameter. This parameter is valid in the Contour applet.

STACKPERCENT=*height-percentage*

specifies the maximum stacking height as a percentage of the longest axis. The default value is 30. This parameter is valid in the Contour applet.

SURFACESIDECOLOR=*color*

specifies the color of the sides of a contour plot when that plot uses multiple colors. The value of the parameter is ignored when drawing a surface plot in a single color. The default color is the color of the minimum data value. The value must be an HTML 3.2 color name or a 6-digit hexadecimal RGB value. This parameter is valid in the Contour applet.

TIPBACKCOLOR=*color*

specifies an HTML 3.2 color name or a 6-digit hexadecimal RGB value for the background of the data tips. The default value is YELLOW. This parameter is valid in the Contour applet.

TIPBORDERCOLOR=*color*

specifies an HTML 3.2 color name or a 6-digit hexadecimal RGB value for the border of the data tips. The default value is BLACK. This parameter is valid in the Contour applet.

TIPS=NONE | STATIONARY | TRUE | FALSE

specifies whether to display data tips. NONE and STATIONARY are valid values only for the Graph and Map applets, and TRUE and FALSE are valid only for the Contour applet. Specifying the default value of STATIONARY or TRUE enables displays data tips, and NONE and FALSE disables this. This parameter is valid in the ActiveX control and in the Graph, Map, and Contour applets.

TIPMODE=STANDARD | HTML | TABULAR

specifies which of two types of data tips are to be displayed. One set of data tips is specified with the TIPS parameter on page 437. The other set of data tips is specified with the HTML= statement option. Specify TIPMODE=HTML to display only the data tips that are indicated by the HTML= statement option. Specify TIPMODE=TABULAR to display only the data tips that are indicated by the value of the TIPS parameter. Specify TIPMODE=STANDARD to display both sets of data tips. The default value is STANDARD. To display data tips with the HTML= statement option, the syntax of that option is HTML="ALT='text' | variable-name". For further information on data tips, see "Adding Data Tips to Web Presentations" on page 568.

TIPSTEMSIZE=*line-length*

specifies the length in pixels of the line that connects the data tips to the graph element that makes use of that data. The default value is 20. This parameter is valid in the Contour applet.

TIPTEXTCOLOR=*color*

specifies an HTML 3.2 color name or a 6-digit hexadecimal RGB value for the text in the data tips. The default value is BLACK. This parameter is valid in the Contour applet.

UNDERFLOWCOLOR=*color*

specifies an HTML 3.2 color name or a 6-digit hexadecimal RGB value for the color that is assigned to data values that are smaller than the minimum range of colors that have been defined in the color palette. The default value is WHITE. This parameter is valid in the Contour applet.

USERFMT*n*=*string(s)*

defines the user format specification. The syntax is the same as that of the VALUE and PICTURE statements for PROC FORMAT. You can specify multiple USERFMT*n* parameters by replacing *n* with the appropriate number from 1 to *n*, where *n* is the number of format parameters to be defined. For example, to define a simple YESNO format, specify the parameter <PARAM NAME="USERFMT1" VALUE="VALUE YESNO 1='Yes' 2='No' ">. This parameter is valid only in the Contour applet.

VIEW2D=TRUE | FALSE

indicates whether the view point should be locked to two dimensions. The default value is TRUE for the Contour applet and FALSE for the Graph applet and ActiveX control. This parameter might be overridden if you specify an ODS style definition.

XBINS=*bin-number-or-values*

YBINS=*bin-number-or-values*

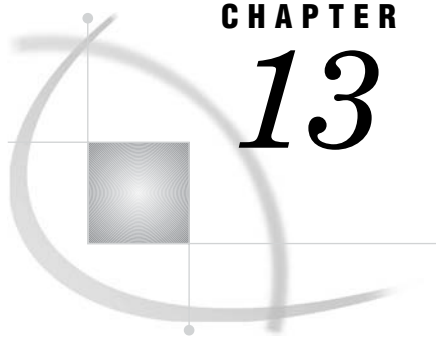
configures the bins uses to generate a contour plot. Specifying a single integer uses that number of bins. The single integer must be greater than 2. Specifying multiple values uses multiple bins with those values. Multiple values are real numbers that are separated by semicolons, as follows:

```
ods html file=filename.html
  parameters=( "XBINS"="-1;0;2.5;3.5;4"
               "YBINS"="1;2;3;4;5;6" );
```

These parameters are enabled in the Contour applet.

VIEWPOINT=2D | SE | SOUTHEAST

defines the initial viewpoint for the Contour applet. The value SE or SOUTHEAST set the initial viewpoint to Southeast, a three-dimensional viewpoint. The value 2D sets the value to be two-dimensional. The default value is 2D for PROC GCONTOUR output and SOUTHEAST for PROC G3D. Setting this parameter unlocks the 2D view (see VIEW2D). This parameter is valid only in the Contour applet.



CHAPTER

13

Generating Static Graphics

<i>What is a Static Graphic?</i>	439
<i>Creating a Static Graphic with ODS</i>	440
<i>ACTXIMG and JAVAIMG Device Drivers Compared to GIF, JPEG, and PNG Device Drivers</i>	440
<i>GIF, JPEG, and PNG Device Drivers</i>	440
<i>ACTXIMG and JAVAIMG Device Drivers</i>	440
<i>Output From Different Device Drivers Compared</i>	441
<i>Developing Web Presentations with the JAVAIMG and ACTXIMG Device Drivers</i>	442
<i>Using JAVAIMG in z/OS</i>	443
<i>When to Use the JAVAIMG or ACTXIMG Device Driver</i>	443
<i>Developing Web Presentations with the GIF, JPEG, and PNG Device Drivers</i>	443
<i>When to Use the GIF, JPEG, and PNG Device Drivers</i>	444
<i>Generating One or More GIF, JPEG, PNG Output Files Without ODS</i>	444
<i>Generating an HTML Output File with ODS and the GIF, PNG, or JPEG Device Driver</i>	445
<i>Naming Conventions Used for Image Output Files</i>	445
<i>Enhancing Web Presentations Generated with the GIF, JPEG, or PNG Device Driver</i>	446
<i>Generating Drill-Down Web Presentations with the GIF, JPEG, or PNG Device Driver</i>	447
<i>Sample Programs for Static Images</i>	447
<i>Using ODS with the ACTXIMG Device Driver</i>	447
<i>Generating GIF Output Using ODS</i>	450
<i>GIF Output with Hotspot Links</i>	452

What is a Static Graphic?

By static graphic, we mean a graphic in the form of a GIF, JPEG, or PNG file. Whereas a user can interact with the graphics presented by the SAS graph applets and the ActiveX Control (such as by hiding nodes or zooming in on a portion on the graph), the only thing a user can do with a static graphic is to look at it because its appearance is permanently fixed once it is created. To generate a static graphic, run a SAS procedure with a GOPTIONS statement with DEVICE= specified as one of the following:

ACTXIMG
 JAVAIMG
 GIF
 JPEG
 PNG

By default, GIF images are created with dimensions of 800 x 600 pixels. Use the following variants of the GIF driver to create different size GIF images:

GIF160 160 x 120

<i>GIF260</i>	260 x 195
<i>GIF373</i>	373 x 280
<i>GIF570</i>	570 x 430
<i>GIF733</i>	733 x 550

Creating a Static Graphic with ODS

You can use a GOPTIONS statement with a device type of GIF, JPEG, or PNG to create a static image file from one or more SAS/GRAPH procedures. SAS first creates a GRSEG entry in your Work catalog, and then creates an image file of the specified type that is identical to the GRSEG entry.

Use ODS with the following arguments to create the HTML file that embeds the image:

<i>FILE=</i>	The filename of the output HTML file (<i>BODY=</i> is a synonym for <i>FILE=</i>).
<i>PATH=</i>	The location (URL or fileref) of the HTML file and static graphic file.
<i>GPATH=</i>	The location of the image file that is created.
	<i>Note:</i> You must specify <i>GPATH</i> only if you specify <i>FILE=</i> as a <i>complete path</i> and file name, and you don't specify <i>PATH=</i> .
	If you specify <i>FILE=</i> as just a filename (and extension), and you specify <i>PATH=</i> , then both the HTML file and the image file are written to the same location (as specified by <i>PATH</i> .) \triangle
<i>STYLE=</i>	The style to be applied, if <i>DEVICE=ACTXIMG</i> or <i>DEVICE=JAVAIMG</i> . The <i>STYLE</i> argument is optional.

For samples, see “Sample Programs for Static Images” on page 447.

ACTXIMG and JAVAIMG Device Drivers Compared to GIF, JPEG, and PNG Device Drivers

GIF, JPEG, and PNG Device Drivers

When you specify GOPTIONS *DEVICE=GIF*, *DEVICE=JPEG*, or *DEVICE=PNG* with a SAS/GRAPH procedure, the image file that is created is identical in appearance to the corresponding GRSEG entry as it appears in the Graph window of SAS.

If you use the Output Delivery System (ODS), then you can add data tips that are displayed when the cursor is over a portion of the image. (See “Data Tips in GIF, JPEG, and PNG Files” on page 568.) You can also add hotspots to images to link to other images or to other URLs. (See “Links in GIF, JPEG, and PNG Files” on page 571.)

ACTXIMG and JAVAIMG Device Drivers

If you specify GOPTIONS *DEVICE=ACTXIMG* or *DEVICE=JAVAIMG*, then a PNG file is created by either a SAS/GRAPH control for ActiveX or by a Java applet and may

not be identical in appearance to the image in the GRSEG catalog, but can be enhanced with ODS stylesheet properties.

When you specify `DEVICE=ACTXIMG`, you can add links to the output of any SAS/GRAPH procedure that supports the `HTML=` or `HTML_LEGEND=` option. For further details, see “Links in GIF, JPEG, and PNG Files” on page 571 . You can also provide pop-up data tips to display when the cursor is over an image created with the `ACTXIMG` device driver (see “Data Tips in `ACTXIMG` and `JAVAIMG` Images” on page 568). When you specify `DEVICE=JAVAIMG`, SAS does not create an image map for hotspot links.

Output From Different Device Drivers Compared

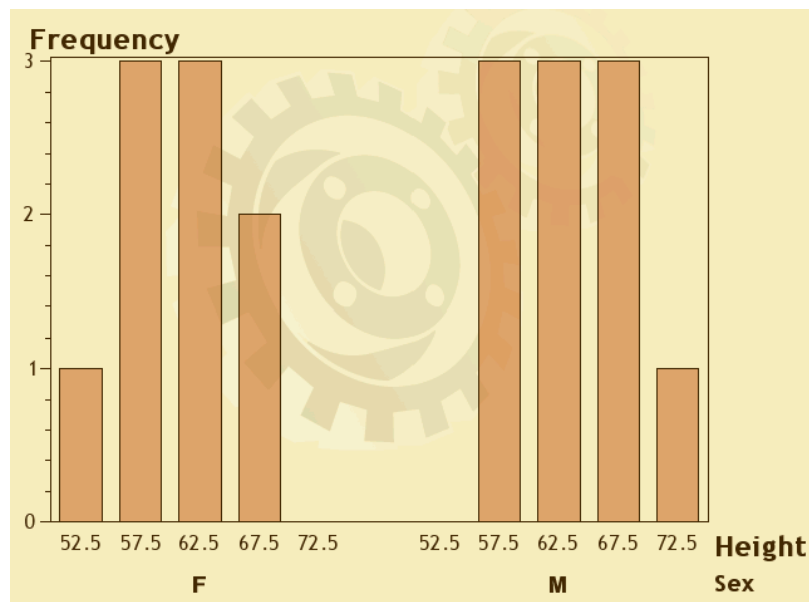
The following example uses the `JAVAIMG` device driver to generate the image. The resulting Web presentation has the visual impact of an interactive presentation, but with smaller files that require no Java access or ActiveX Control installation and are not interactive. In this example, the ODS style `GEARS` specifies a color-coordinated background and background image, along with coordinated text fonts and sizes in the axis labels. The code for this example, which is available in the SAS Sample Library under the name `GWBJAIMG`, is as follows:

```
ods listing close;
ods html file='temp.html' style=gears;
goptions device=javaimg;

proc gchart data=sashelp.class;
  vbar height / group=sex name='test' ;
run;
quit;

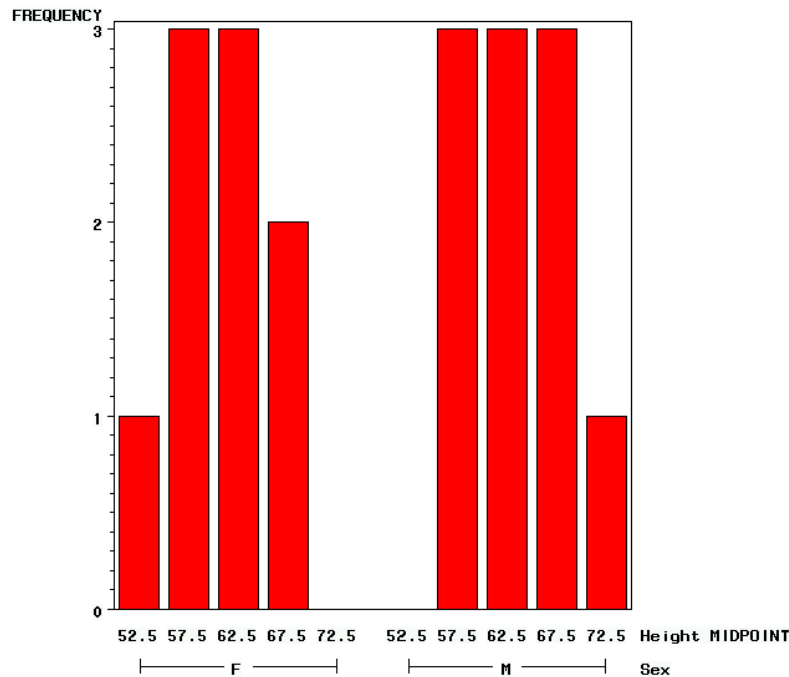
ods html close;
```

Display 13.1 A Bar Chart Using the `JAVAIMG` Device Driver



The following images shows the same bar chart created with the GIF device driver. You can generate this image by specifying GOPTIONS DEVICE=GIF. The graphs generated by specifying DEVICE=JPEG or DEVICE=PNG are very similar to this and differ mostly in the physical file size.

Display 13.2 A Bar Chart Using the GIF Device Driver



Developing Web Presentations with the JAVAIMG and ACTXIMG Device Drivers

The JAVAIMG and ACTXIMG device drivers enable you to generate Web presentations that display one or more graphs in PNG format. The resulting PNG files can be viewed in any browser—neither Java nor ActiveX is required to view them.

The PNG files are identical in appearance to the graphs created with DEVICE=JAVA or DEVICE=ACTIVEX as they are initially displayed in a browser by a SAS Java applet or SAS ActiveX Control respectively. However, unlike these latter graphs which are interactive and can be manipulated by a user viewing them in a browser, PNG files are static and their appearance can not be changed after they are created.

When you run a program that specifies the ACTXIMG device driver, the ActiveX Control runs in the background to generate the PNG image files. Your computer must therefore be running in the Windows 32-bit operating environment with the ActiveX Control installed in advance. For installation information, see “Installing the ActiveX Control” on page 389. SAS/GRAPH procedures that can be used with the ACTXIMG device driver are the same as those that can be used with the ActiveX Control, as listed in Table 10.1 on page 388. The procedures that can be used with the JAVAIMG device driver are listed in “Graph, Map, and Contour Applets” on page 372.

Use ODS to specify graph styles for charts and to format the HTML output file when you specify either DEVICE=JAVAIMG or ACTXIMG. For an example that uses the

ACTXIMG device driver, see “Using ODS with the ACTXIMG Device Driver” on page 447.

Note that using PROC GMAP to generate a highly detailed map might create a large HTML output file, which might cause problems on certain browsers. Running PROC GREduce may help to remove some of the complexity and produce a more usable map.

Using JAVAIMG in z/OS

If you are running SAS in the z/OS operating environment with DEVICE=JAVAIMG, then you must specify filesystem=HFS because HFS file space is needed to write the image files. You may also need to increase the amount of memory that is allotted for your session so that SAS can run Java in the background. The suggested region size is 200 megabytes. For a batch job, add either REGION=200M or REGION=204800K to the JOB card. For a TSO session, specify SIZE(204800). For more information, refer to your JCL reference manual.

When to Use the JAVAIMG or ACTXIMG Device Driver

The PNG images that are generated by the ACTXIMG and JAVAIMG device drivers can differ in appearance from those that are generated by the GIF, JPEG, and PNG drivers. (Although it also creates a PNG file, the PNG driver does not use the ActiveX or Java routines that the ACTXIMG and JAVAIMG device drivers use.) Colors, fonts, shading, and lines are visibly different. Specifying one of the graph styles in ODS results in further differences because colors, fonts, and images have been coordinated in advance.

If you don't need interactivity such as zoom, the JAVAIMG and ACTXIMG device drivers provide several advantages over the interactive presentations that are generated with JAVA and ACTIVEX. Because PNG image files are generated, the Web clients are not required to access the Java run-time environment or install the ActiveX Control to display the graphs. Also, Web performance improves because the PNG image files are smaller in size than the HTML files that run an applet or the ActiveX Control.

Note: The ACTXIMG device cannot be used with the ODS PDF, PCL, PS, or PRINTER destinations on 64-bit machines. SAS uses the JAVAIMG device instead. △

Note: When SAS is installed on a server, the ACTXIMG and JAVAIMG drivers are limited by the display capabilities of the server on which they run—for example, the number of colors that the server is capable of. Consequently, the PNG output might not look as good as what you get from the client-side drivers (JAVA and ACTIVEX). Thus, it is better to use JAVA/ACTIVEX if the server's display settings are less than optimal. △

Developing Web Presentations with the GIF, JPEG, and PNG Device Drivers

You can use the GIF, JPEG, and PNG drivers with ODS to generate an HTML file to display multiple images. For details, see “Generating an HTML Output File with ODS and the GIF, PNG, or JPEG Device Driver” on page 445. For information on using these drivers without ODS, see “Generating One or More GIF, JPEG, PNG Output Files Without ODS” on page 444.

Enhancements that are available to GIF, PNG, and JPEG Web presentations include formatting of the HTML output file using ODS, as described in “Overview of ODS Enhancements for Web Output” on page 487, and adding drill-down or pop-up data-tip functionality. Drill-down functionality can be enabled in two ways. The elements of the graph can be hotspots, or the elements of an Annotate data set can be hotspots. For details, see “Generating Drill-Down Web Presentations with the GIF, JPEG, or PNG Device Driver” on page 447.

When to Use the GIF, JPEG, and PNG Device Drivers

The GIF, JPEG, and PNG device drivers are best suited to Web presentations with interactivity that is limited to drill-down functionality and that is implemented in an automatically generated image map in the HTML output file. If you need more interactivity, or if you want to compute responses to drill-down actions when the graph is viewed, then generate a presentation that runs in a Java applet or in the ActiveX Control.

If you do not need drill-down functionality, use the ACTXIMG or JAVAIMG device driver to generate a Web presentation with the best available image quality. These device drivers use an applet or the ActiveX Control to generate PNG image output files. The images can exhibit the color blending, transparency, anti-aliasing, and shading that are available in the graph styles, as described in “Developing Web Presentations with the JAVAIMG and ACTXIMG Device Drivers” on page 442.

Generating One or More GIF, JPEG, PNG Output Files Without ODS

To generate just one GIF, JPEG, or PNG image file, specify a fileref, filename, and storage location in a FILENAME statement, as follows:

```
filename fileref "your_path/your_file.image_extension";
```

The fileref can be up to eight characters in length.

The following code shows how an actual FILENAME statement might look for one output image:

```
filename mygif1 "C:\mysas\images\barchart.gif";
```

To generate multiple images in a single program, specify a fileref for the path only, as follows:

```
filename fileref "your_path";
```

As shown in this example:

```
filename imageout "C:\mysas\images";
```

When you generate multiple image output files, SAS/GRAPH automatically generates the names of the image files, as described in “Naming Conventions Used for Image Output Files” on page 445.

After assigning a fileref, all you need to add to generate an image output file are values for the graphics options DEVICE= and GSFNAME=, as follows:

```
goptions device=device_driver
         gsfname=fileref;
```

The value of the GSFNAME= option is the name of your previously defined fileref, whether that fileref references a filename or a directory.

When you have specified a storage location and your `DEVICE=` and `GSFNAME=` graphics options, then you can run the procedure that generates the graph. The output will be stored in the specified format in the specified output location.

Generating an HTML Output File with ODS and the GIF, PNG, or JPEG Device Driver

Follow these steps to generate a complete Web presentation that consists of an HTML output file and one or more images:

- 1 To conserve resources, close the ODS listing destination (the Output window, which is open by default). Then reset graphics options as follows:

```
ods listing close;
goptions reset=all;
```

- 2 Enter your `DATA` step, if necessary.
- 3 Specify your ODS HTML statement, with the following options:

```
ods html
  path='C:/Public/graph' (url=none)/* HTML output directory */
  body='webgif1.htm'      /* HTML filename          */
  gpath='C:/Public/graph'; /* image file output location */
```

Specifying `URL=NONE` tells ODS to reference the image file simply by name without prefixing the full path (assuming that the image file is in the same directory as the HTML file).

Note: With the GIF, JPEG, or PNG device driver, footnotes and titles are stored in the image file by default. To move footnotes and titles out of the image file and into the HTML file, specify the ODS HTML options `NOGTITLE` or `NOGFOOTNOTE` or both. Δ

- 4 Specify your device driver:

```
goptions device=gif;
```

- 5 Run procedures to generate graphs. Each procedure ends with a `RUN` statement.
- 6 Close the HTML output file and reopen the ODS listing destination:

```
ods html close;
ods listing;
```

Reopening the listing destination establishes standard operating conditions for later programs that you run in the same SAS session.

Note: Using this technique, however, you can not create hotspots for links on your graphics or for data tips. Δ

Naming Conventions Used for Image Output Files

When you use the GIF, JPEG, or PNG device drivers to generate output for the Web using ODS, the graphs are saved as GRSEG catalog entries and as image format files. If you do not specify filenames, then SAS generates them based on the names of the catalog entries.

For example, you can use a procedure's `NAME=` option to assign a name of up to eight characters to the catalog entry. If you assign the name `MYGRAPH` to the catalog

entry, then SAS/GRAPH names the GIF image file MYGRAPH.GIF. If you do not use the NAME= option, then SAS/GRAPH names the entry with the first eight characters of the procedure name (for example, GCHART), in which case SAS/GRAPH names the GIF file GCHART.GIF. For more information on catalog entry names, see “Names and Descriptions of Catalog Entries” on page 55.

By default, SAS/GRAPH does not replace existing GRSEG entries when a procedure creates a new entry of the same name. Rather, it increments the duplicate name to make it unique. For example, if you use a procedure’s NAME= option to name an entry MYGRAPH and an entry named MYGRAPH already exists in the output catalog, SAS/GRAPH names the new entry MYGRAPH1, and then names the GIF file MYGRAPH1.GIF. Catalog entry names are limited to eight characters, so if the duplicate name has eight characters, SAS/GRAPH replaces the final character with the added number.

To replace an existing catalog entry, your program can first use the GREPLAY procedure to delete the existing catalog or catalog entries (although doing so is not required). For example, assume that the output catalog is the default, WORK.GSEG, and assume that you use BY-group processing on the GCHART procedure to run a program that generates three catalog entries that are named GCHART, GCHART1, and GCHART2 by default. To run the same program again in the same session, and to ensure that the catalog entries receive the same names, you can first run the following GREPLAY procedure to delete the three existing catalog entries, otherwise the new entries will be named gchart3, gchart4, and gchart5:

```
proc greplay igout=work.gseg nofs;
    delete gchart gchart1 gchart2;
run; quit;
```

To delete all of the catalog entries, use:

```
proc greplay igout=work.gseg nofs;
    delete _all_;
run;
quit;
```

Enhancing Web Presentations Generated with the GIF, JPEG, or PNG Device Driver

This section shows you how to enhance the appearance and functionality of Web presentations that are generated with a GIF, JPEG, or PNG device driver. For information on the default configurations of these Web presentations, see “Developing Web Presentations with the GIF, JPEG, and PNG Device Drivers” on page 443.

The available enhancements are as follows:

- Add drill-down links to graph elements or legend elements or both. See “Generating Drill-Down Web Presentations with the GIF, JPEG, or PNG Device Driver” on page 447.
- Format your HTML output file using ODS. See “Overview of ODS Enhancements for Web Output” on page 487.
- Add drill-down links to graphical elements specified in an Annotate data set. See “Generating Web Links with the Annotate Facility” on page 500.

Generating Drill-Down Web Presentations with the GIF, JPEG, or PNG Device Driver

Using the GIF, JPEG, or PNG device driver, you can generate a complete drill-down Web presentation with selectable elements in the graph or legend. To enable the drill-down functionality, the graphics procedure must support the HTML= or the HTML_LEGEND= option.

Follow these steps to generate a drill-down graph with the GIF, JPEG, or PNG device driver.

- 1 To save resources, close the ODS listing destination.

```
ods listing close;
```

- 2 Set graphics options.

```
goptions reset=all device=gif;
```

- 3 Initialize one or two link variables and add values to those variables. The link variables provide the drill-down URLs that will appear in the image map of the HTML output file, as described in “GIF Output with Hotspot Links” on page 452.

- 4 Generate an HTML output file using ODS.

```
ods html file="mygif1.htm"
      gpath="C:\mypath\web"
```

- 5 Generate the graph as an image output file. In the statement that generates the graph (such as VBAR3D), assign the name of a link variable as the values of the HTML= option or the HTML_LEGEND= option or both. (See “Links in GIF, JPEG, and PNG Files” on page 571.)

- 6 If necessary, create the HTML pages to be linked to.

- 7 Close the HTML output file and open the ODS listing destination.

```
ods html close;
ods listing;
```

Run the program and display the HTML output file in the SAS Results window or in a Web browser. Selecting an element in the graph points the Web browser to the associated drill-down URL.

Sample Programs for Static Images

The following sample programs create a Web presentation with a static image:

- “Using ODS with the ACTXIMG Device Driver” on page 447
- “Generating GIF Output Using ODS” on page 450
- “GIF Output with Hotspot Links” on page 452

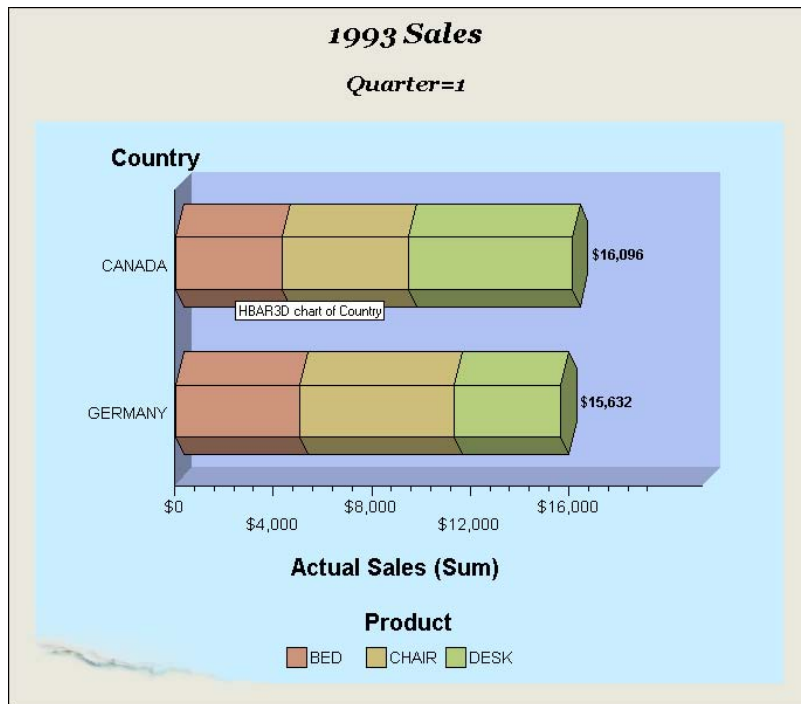
Using ODS with the ACTXIMG Device Driver

The following sample program uses ODS HTML to create an HTML file that references four PNG files created by a SAS procedure when DEVICE=ACTXIMG. Because the ACTXIMG device driver invokes an ActiveX Control, you can only run this example in a Windows environment.

The GCHART procedure in this example includes a BY statement to display the results of each of the four quarters of the year. Consequently, the procedure produces

four separate PNG files, only the first of which is shown here. A user would have to scroll down the page in the browser to see all the PNG images displayed.

Display 13.3 Using ODS with the ACTXIMG Device Driver



The following is the complete SAS code to generate PNG files from a SAS/GRAPH procedure using GOPTIONS DEVICE=ACTXIMG. You should notice the following:

- PROC GREPLAY is used to delete any old GRSEGS that were created. This is not necessary, but otherwise SAS creates new GRSEGS each time the procedure is run rather than replacing the old, and from them creates new PNG files, incrementing the suffix number for each new PNG file.
- The FILE= option of the ODS HTML statement specifies the path and filename of the HTML file to be created. If you want to run this example, then change the value of the option to the location where you want to store the file.

Note: You can specify the complete path and filename with the FILE= option (or the BODY= option, which is the same), or you can specify the path separately using the PATH= option, and just the filename with the FILE= or BODY= option. See the section “ODS HTML Statement” in the *SAS Output Delivery System: User’s Guide*. \triangle

- The GPATH= option of the ODS HTML statement specifies the directory where the PNG files are to be created. If you want to run this example, then change the value of the option to the location where you want to store the file.
- Specifying that GOPTIONS DEVICE=ACTXIMG causes the GCHART procedure to produce PNG output.

```
data prdsummary;
  input Year Quarter Country $ Product $ Actual dollar10.2;
  label Actual='Actual Sales';
```

```

format Actual dollar11.;
datalines;
1993 1 CANADA BED $4,337.00
1993 1 CANADA CHAIR $5,115.00
1993 1 CANADA DESK $6,644.00
1993 1 GERMANY BED $5,026.00
1993 1 GERMANY CHAIR $6,276.00
1993 1 GERMANY DESK $4,330.00
1993 2 CANADA BED $2,437.00
1993 2 CANADA CHAIR $3,115.00
1993 2 CANADA DESK $5,654.00
1993 2 GERMANY BED $3,026.00
1993 2 GERMANY CHAIR $2,276.00
1993 2 GERMANY DESK $3,320.00
1993 3 CANADA BED $6,337.00
1993 3 CANADA CHAIR $7,145.00
1993 3 CANADA DESK $7,614.00
1993 3 GERMANY BED $5,026.00
1993 3 GERMANY CHAIR $3,276.00
1993 3 GERMANY DESK $6,340.00
1993 4 CANADA BED $9,337.00
1993 4 CANADA CHAIR $2,115.00
1993 4 CANADA DESK $3,646.00
1993 4 GERMANY BED $6,026.00
1993 4 GERMANY CHAIR $7,276.00
1993 4 GERMANY DESK $8,350.00
1994 1 CANADA BED $3,327.00
1994 1 CANADA CHAIR $5,345.00
1994 1 CANADA DESK $7,624.00
1994 1 GERMANY BED $4,026.00
1994 1 GERMANY CHAIR $3,276.00
1994 1 GERMANY DESK $3,340.00
1994 2 CANADA BED $5,356.00
1994 2 CANADA CHAIR $3,115.00
1994 2 CANADA DESK $7,623.00
1994 2 GERMANY BED $8,026.00
1994 2 GERMANY CHAIR $5,276.00
1994 2 GERMANY DESK $7,321.00
1994 3 CANADA BED $4,321.00
1994 3 CANADA CHAIR $3,115.00
1994 3 CANADA DESK $5,658.00
1994 3 GERMANY BED $6,026.00
1994 3 GERMANY CHAIR $5,276.00
1994 3 GERMANY DESK $6,398.00
1994 4 CANADA BED $5,357.00
1994 4 CANADA CHAIR $4,166.00
1994 4 CANADA DESK $7,662.00
1994 4 GERMANY BED $4,026.00
1994 4 GERMANY CHAIR $5,246.00
1994 4 GERMANY DESK $3,329.00
;
/* delete previously created grsegs before creating new ones */
proc greplay igout=work.gseg nofs;
  delete _all_;

```

```

        /* could also specify: delete _1993, _19931, etc. */
run;
quit;

ods listing close;
/* gpath specifies the directory where PNGs are created */
ods html file='u:\public_html\Web_output\ods_actximg.htm'
      gpath='u:\public_html\Web_output\'
      style=torn;
options reset=all device=actximg;

title1 '1993 Sales';
proc gchart data=prdsummary(where=(year=1993));
  hbar3d country / sumvar=actual subgroup=product sum
  shape=hexagon caxis=black cframe=CXb0c1f4 name='_1993';
  by quarter;
run;
quit;

ods html close;
ods listing;

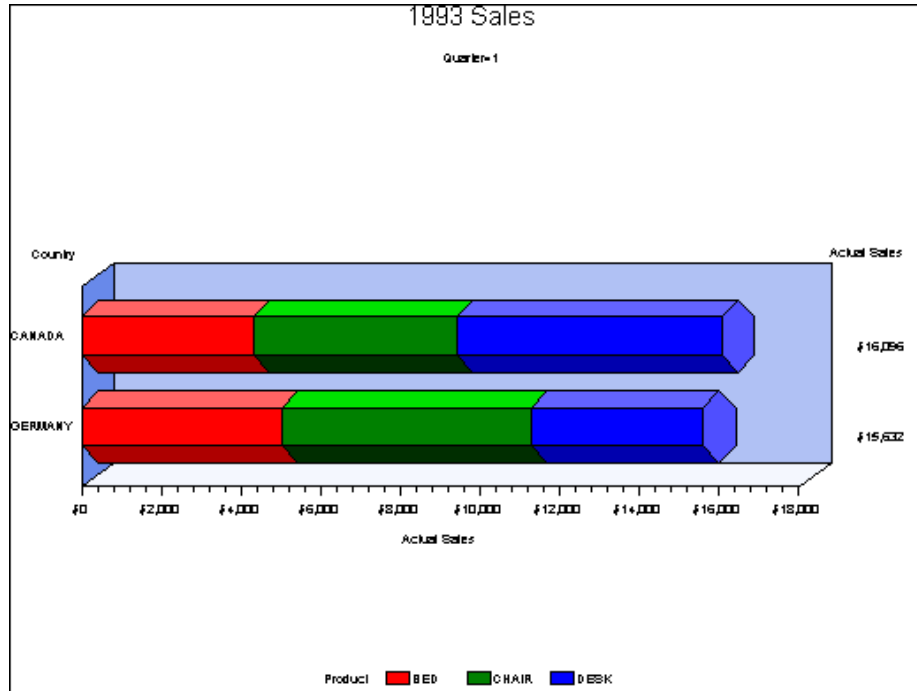
```

Generating GIF Output Using ODS

The following sample program uses ODS to create an HTML file that references four GIF files created by a SAS procedure when `DEVICE=GIF`. The GIFs are displayed one after the other in the HTML page, so that a user would have to scroll to see all the graphs.

The `GCHART` procedure in this example includes a `BY` statement to display the results of each of the four quarters of the year. Consequently, the procedure produces four separate GIF files, only the first of which is shown here. A user would have to scroll the page in the browser to see all the GIF images displayed.

Display 13.4 Generating GIF Output Using ODS



The following is the complete SAS code to generate GIF files from a SAS/GRAPH procedure. You should notice the following:

- PROC GREPLAY is used to delete the GRSEGs that are already created. This is not necessary, but otherwise SAS creates new GRSEGs each time the procedure is run, rather than overwriting the old, and from the new GRSEGs creates new GIF files, incrementing the suffix number for each new GIF.
- The FILE= option of the ODS HTML statement specifies the path and filename of the HTML file to be created. If you want to run this example, then change the value to the directory where you want to store the HTML file.

Note: You can specify the complete path and filename with FILE= (or BODY=, which is a synonym), or you can specify the path separately using PATH=, and just the filename with FILE= (or BODY=) See the *SAS Output Delivery System: User's Guide* for information on the ODS HTML statement. △

- The GPATH= option of the ODS HTML statement specifies the directory where the GIF files are to be created. If you want to run this example, then change the value of the option to the location where you want to store the file.
- The statement GOPTIONS DEVICE=GIF causes the GCHART procedure to produce GIF output.

```
data prdsummary;
  input Year Quarter Country $ Product $ Actual dollar10.2;
  label Actual = 'Actual Sales';
  format Actual dollar11.;
  datalines;
1993 1 CANADA BED $4,337.00
1993 1 CANADA CHAIR $5,115.00
1993 1 CANADA DESK $6,644.00
```

```

1993 1 GERMANY BED $5,026.00
1993 1 GERMANY CHAIR $6,276.00
1993 2 GERMANY CHAIR $2,276.00
... more data lines ...
1994 4 CANADA CHAIR $4,166.00
1994 4 CANADA DESK $7,662.00
1994 4 GERMANY BED $4,026.00
1994 4 GERMANY CHAIR $5,246.00
1994 4 GERMANY DESK $3,329.00
;
/* delete previously created grsegs before creating new ones */
proc greplay igout=work.gseg nofs;
  delete _all_;
run;
quit;

ods listing close;
/* "file=" specifies the html file to be created */
/* Change the value of file= to the directory where you want */
/* to store the HTML file */
/* Change file= to the directory where you want to store the HTML file */
/* "gpath=" specifies the directory where GIFs are created */
/* Change the value of gpath= to the directory that you are using */
ods html file='u:\public_html\Web_output\ods_gif.htm'
  gpath='u:\public_html\Web_output\';
goptions reset=all device=gif
  border
  ftext="Helvetica" ftitle="Helvetica";

title1 '1993 Sales';
proc gchart data=prdsummary(where=(year=1993));
  hbar3d country / sumvar=actual subgroup=product sum
  shape=hexagon caxis=black cframe=CXb0c1f4 name='1993_';
  by quarter;
run;
quit;
ods html close;
ods listing;

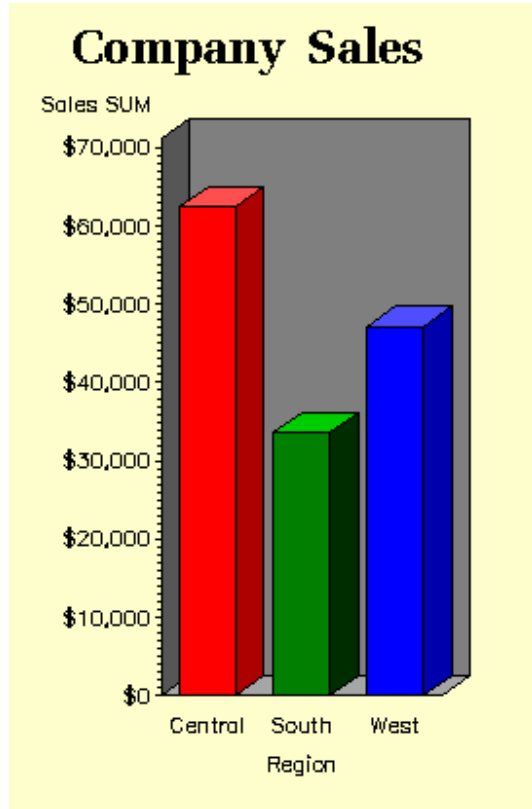
```

GIF Output with Hotspot Links

This example shows you how to generate Web output with drill-down functionality using the GIF device driver (see also “Generating Drill-Down Web Presentations with the GIF, JPEG, or PNG Device Driver” on page 447).

In the program, the `DEVICE=GIF` specification generates image output files and the `ODS HTML` statement generates an HTML output file. The `HTML=` option identifies a link variable that provides drill-down URLs. The values of the link variables are added to the data set with `IF/THEN` statements. ODS inserts the drill-down URLs into an image map that it generates in the HTML output file.

When you display the HTML output file in a Web browser and select one of the three blocks in the chart, you see a table of the data for that block.

Display 13.5 Three-Dimensional Vertical Bar Chart with Drill-Down Links

State	Sales
IL	\$18,038
IN	\$13,611
OH	\$11,084
MI	\$19,660

Here is the example code, which is available in the SAS Sample Library under the name GWBDRILL:

```

/* Close the listing destination */
ods listing close;

/* Set graphic options. */
goptions reset=global gunit=pct
          transparency noborder

```

```

        htitle=6 htext=3
        device=gif;

/* Create the data set REGSALES. */
data regsales;
    length Region State $ 8;
    format Sales dollar8.;
    input Region State Sales;

/* Initialize the link variable. */
    length rpt $40;

/* Assign values to the link variable. */
if Region='Central' then
    rpt='href="central.html"';
else if Region='South' then
    rpt='href="south.html"';
else if Region='West' then
    rpt='href="west.html"';

    datalines;
West CA 13636
West OR 18988
West WA 14523
Central IL 18038
Central IN 13611
Central OH 11084
Central MI 19660
South FL 14541
South GA 19022
;

/* Remove the comments below to open the HTML destination for ODS output. */
/* Specify the filename in BODY= and the output path in PATH=. */

/* ods html body='your-filename.htm'
    path='your-web-path'; */

/* Create a chart that uses the link variable. */
title1 'Company Sales';
proc gchart data=regsales;
    vbar3d region / sumvar=sales
    patternid=midpoint
    html=rpt;
run;
quit;

/* Remove the comments below, and specify the filename and */
/* path to open an HTML file that will contain the report. */

/* ods html body='your-filename.htm' */
/* path='your-web-path'; */

title1 'Central Sales';

```

```
proc print data=regsales noobs;
  var state sales;
  where region='Central';
run;
quit;

title1 'Southern Sales';
/* Remove the comments below, and specify the filename and */
/* path to open an HTML file that will contain the report. */

/* ods html body='your-filename2.htm' */
/*   path='your-web-path';          */

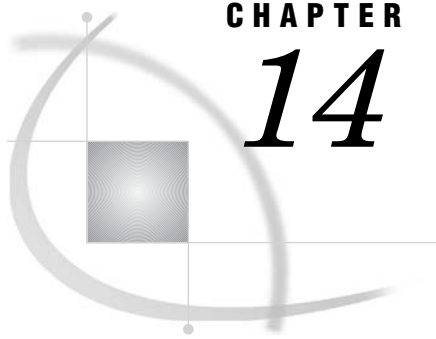
proc print data=regsales noobs;
  var state sales;
  where region='South';
run;
quit;

title1 'Western Sales';
/* Remove the comments below, and specify the filename and */
/* path to open an HTML file that will contain the report. */

/* ods html body='your-filename3.htm' */
/*   path='your-web-path';          */

proc print data=regsales noobs;
  var state sales;
  where region='West';
run;
quit;

/* Close the HTML output file and */
/* open the listing destination. */
ods html close;
ods listing;
```

CHAPTER

14

Generating Web Animation with GIFANIM

<i>Developing Web Presentations with the GIFANIM Device Driver</i>	457
<i>When to Use the GIFANIM Device Driver</i>	457
<i>Creating an Animated Sequence</i>	458
<i>Preparing the Header</i>	458
<i>Preparing the Body</i>	458
<i>Preparing the Trailer</i>	458
<i>GOPTIONs for Configuring GIFANIM Presentations</i>	459
<i>Sample Programs: GIFANIM</i>	459
<i>Sample Animated GIF, with HTML from PUT Statements</i>	460
<i>Results Shown in a Browser</i>	460
<i>SAS Code</i>	460
<i>HTML File</i>	463
<i>Generating an Animated Web Presentation with the GIFANIM Device Driver</i>	463

Developing Web Presentations with the GIFANIM Device Driver

The GIFANIM device driver enables you to create sequences of images that are displayed automatically from a single GIF file. These animated sequences are commonly referred to as slide shows. The display sequence repeats until the Web user selects Stop in the Web browser or displays another Web page.

You can customize GIFANIM Web presentations by specifying the display time of individual images, the number of loops before stopping, and the size of the images. Graphics options are used to configure GIFANIM presentations, as described in “GOPTIONs for Configuring GIFANIM Presentations” on page 459.

When to Use the GIFANIM Device Driver

The GIFANIM device driver is useful for slide shows or animations that do not need to be controlled by the Web user. Infinite looping is appropriate for unattended kiosk displays. To enable the Web user to specify the display rate, change the size of the animation, drill down for additional information, and utilize the optimal rendering capabilities of the SAS/GRAPH Java library, use the JAVAMETA device driver. This device driver generates Web presentations that run in the Metaview Applet, as described in “Developing Web Presentations for the Metaview Applet” on page 469.

Creating an Animated Sequence

To create an animated sequence with the GIFANIM device driver, you need to ensure that the resulting data stream is constructed properly. The GIFANIM data stream has three parts: header, body, and trailer.

To see an example of a program that uses the GIFANIM device driver, see “Sample Programs: GIFANIM” on page 459.

Preparing the Header

When creating a new animated GIF data stream, you must issue GOPTIONS GSFMODE=REPLACE; prior to the invocation of the first SAS/GRAPH procedure. The driver will then construct a new data stream by writing a valid GIF header and inserting graphical data from the first procedure.

Preparing the Body

After the first procedure has been executed, you must construct the body of the GIF animation. You can think of the body as all of the graphic images between the first and the last images in the sequence. Set GOPTIONS GSFMODE=APPEND to suppress the header information and to begin appending graphic data to the current data stream. The GOPTIONS GSFMODE=APPEND statement must appear after the first and before second SAS/GRAPH procedures.

Note: If you use BY-group processing on the first graphics procedure to generate multiple graphs, then the output is automatically appended to the same GIF file. Thus, you do not need to specify GSFMODE=APPEND for that first procedure. If you do not use a second graphics procedure to append additional graphs to the GIF file, you do not need to set the GSFMODE= option in the body section of your program. \triangle

Preparing the Trailer

The final step in the GIF animation process is to mark the end of the animation by appending a GIF trailer ('3B'x) to the data stream. The way to do this depends on whether or not the last procedure uses BY-group processing.

- Without BY-group processing, set GOPTIONS GEPILOG='3B'X before the last SAS/GRAPH procedure.
- With BY-group processing, do not assign a value to GEPILOG; otherwise your GIF animation sequence will be incomplete. Because a GEPILOG is written after each graph in a BY-group, the GIF decoder will interpret the first '3B'x as the end of the animation. Instead, you should use a DATA step to add the trailer to the data stream:

```
data _null_;
  file out recfm=n mod;
  put '3B'x;
run;
```

In the preceding example, OUT is the fileref of the GIF output file.

After the animation is complete, issue a GOPTIONS RESET=ALL statement to prepare for succeeding SAS jobs.

GOPTIONS for Configuring GIFANIM Presentations

You can specify the following options in the GOPTIONS statement to configure Web presentations that were generated with the GIFANIM device driver.

ITERATION=*iteration-count*

specifies the number of times to repeat the animation loop, or that the loop repeats continuously. The default value of 0 continues the animation indefinitely (until the Web user selects Stop or displays another Web page in the Web browser). Specifying a number greater than 0 repeats the animated sequence for the specified number of iterations, and then continuously displays the last image in the sequence, unless the DISPOSAL= graphics option specifies otherwise.

GSFMODE=REPLACE | APPEND

specifies whether the graphics output should replace the contents of an existing file or be appended to it. In this case, the value of REPLACE specifies that the device driver is to write a GIF header. Use the GSFMODE= option to specify when to write the GIF header. Specify REPLACE before you generate the first GIF image, then specify APPEND in a second statement before you generate the rest of the images.

DELAY=*delay-time*

specifies the amount of time that each image is displayed, in hundredths of a second. For example, a value of 1 specifies a delay of 0.01 seconds. The default value is 0.

DISPOSAL=NONE | BACKGROUND | PREVIOUS | UNSPECIFIED

specifies how the image sequence is to be displayed.

NONE

superimposes the images in the sequence, without removing any of them from the screen. This is the default value.

BACKGROUND

restores the background color before displaying the next image.

PREVIOUS

replaces the current image with the previous image before displaying the next image.

UNSPECIFIED

takes no further action before displaying the next image.

XPIXELS=*horizontal-size*

YPIXELS=*vertical-size*

specify the size of the images in the sequence.

USERINPUT | **NOUSERINPUT**

allows or does not allow user input during image animation if user input is supported by the browser displaying the animation.

TRANSPARENCY | **NOTRANSPARENCY**

specifies whether the background of the image should be replaced by the background color of the Web browser.

Sample Programs: GIFANIM

The following sample programs generate animated GIFs:

- “Sample Animated GIF, with HTML from PUT Statements” on page 460
- “Generating an Animated Web Presentation with the GIFANIM Device Driver” on page 463

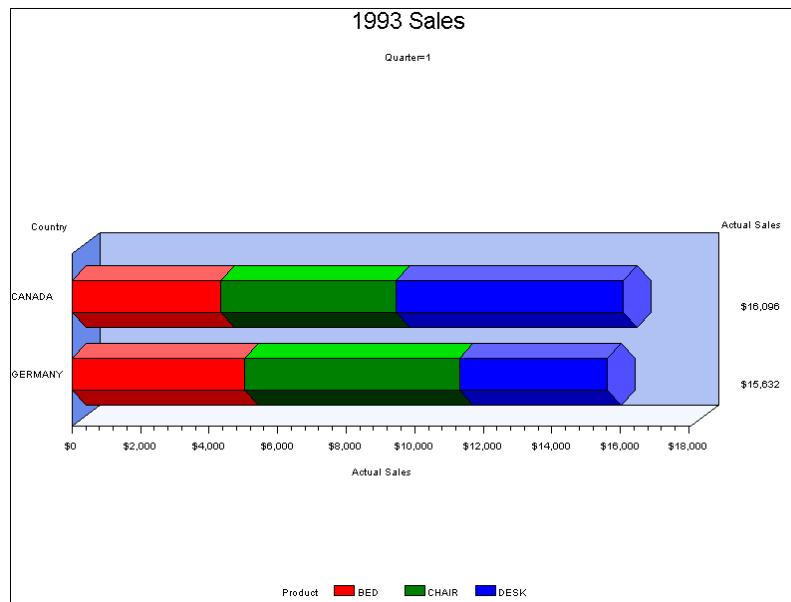
Sample Animated GIF, with HTML from PUT Statements

The following sample program generates an animated GIF from a SAS data set and two invocations of PROC GCHART, each of which contains a BY statement.

- “Results Shown in a Browser” on page 460
- “SAS Code” on page 460
- “HTML File” on page 463

Results Shown in a Browser

The following picture shows only the first picture of the animated GIF. After a specified time lapse, the chart for each quarter of each of the two years is displayed in turn.



SAS Code

The following is the complete SAS code to generate the animated GIF and an HTML file that references it. You should notice the following:

- The GSFNAME= option of the GOPTIONS statement specifies the name of the GIF to be created. In this example, the value of GSFNAME is specified in an associated FILENAME statement. If you want to run this example, then change the value of the FILENAME statement to something that makes sense for you.
- The following statement

```
goptions gsfmode=append;
```

is included before the second invocation of PROC GCHART so that the output is appended to the same GIF file.

- A FILE statement specifies the complete path and file name of the HTML file to be created by the PUT statements. If you want to run this example, then change the value to something that makes sense for you

```

data prdsummary;
  input Year Quarter Country $ Product $ Actual dollar10.2;
  label Actual = 'Actual Sales';
  format Actual dollar11.;
  datalines;
1993 1 CANADA BED $4,337.00
1993 1 CANADA CHAIR $5,115.00
1993 1 CANADA DESK $6,644.00
1993 1 GERMANY BED $5,026.00
1993 1 GERMANY CHAIR $6,276.00
1993 1 GERMANY DESK $4,330.00
1993 2 CANADA BED $2,437.00
1993 2 CANADA CHAIR $3,115.00
1993 2 CANADA DESK $5,654.00
1993 2 GERMANY BED $3,026.00
1993 2 GERMANY CHAIR $2,276.00
1993 2 GERMANY DESK $3,320.00
1993 3 CANADA BED $6,337.00
1993 3 CANADA CHAIR $7,145.00
1993 3 CANADA DESK $7,614.00
1993 3 GERMANY BED $5,026.00
1993 3 GERMANY CHAIR $3,276.00
1993 3 GERMANY DESK $6,340.00
1993 4 CANADA BED $9,337.00
1993 4 CANADA CHAIR $2,115.00
1993 4 CANADA DESK $3,646.00
1993 4 GERMANY BED $6,026.00
1993 4 GERMANY CHAIR $7,276.00
1993 4 GERMANY DESK $8,350.00
1994 1 CANADA BED $3,327.00
1994 1 CANADA CHAIR $5,345.00
1994 1 CANADA DESK $7,624.00
1994 1 GERMANY BED $4,026.00
1994 1 GERMANY CHAIR $3,276.00
1994 1 GERMANY DESK $3,340.00
1994 2 CANADA BED $5,356.00
1994 2 CANADA CHAIR $3,115.00
1994 2 CANADA DESK $7,623.00
1994 2 GERMANY BED $8,026.00
1994 2 GERMANY CHAIR $5,276.00
1994 2 GERMANY DESK $7,321.00
1994 3 CANADA BED $4,321.00
1994 3 CANADA CHAIR $3,115.00
1994 3 CANADA DESK $5,658.00
1994 3 GERMANY BED $6,026.00
1994 3 GERMANY CHAIR $5,276.00
1994 3 GERMANY DESK $6,398.00

```

```

1994 4 CANADA BED $5,357.00
1994 4 CANADA CHAIR $4,166.00
1994 4 CANADA DESK $7,662.00
1994 4 GERMANY BED $4,026.00
1994 4 GERMANY CHAIR $5,246.00
1994 4 GERMANY DESK $3,329.00
;
/* delete previously created gsegs before creating new ones */
/* (SAS creates gsegs before creating gifs from them          */
proc greplay igout=work.gseg nofs;
  delete _all_;
  /* could also specify: delete _1993, _19931, etc. */
run; quit;

/* use filename to specify output folder for gif files */
filename myimages 'u:\public_html\Web_output\gifanim.gif';
goptions reset=all device=gifanim gsfname=myimages
          gsfmode=replace /* not necessary when using "BY" */
          delay=150      /* set delay between images      */
          border
          ftext="Helvetica" ftitle="Helvetica";

title1 '1993 Sales';
proc gchart data=prdsummary(where=(year=1993));
  hbar3d country / sumvar=actual subgroup=product sum
  shape=hexagon caxis=black cframe=CXb0c1f4;
  by quarter;
run;
quit;
goptions gsfmode=append;
title1 '1994 Sales';
proc gchart data=prdsummary(where=(year=1994));
  hbar3d country / sumvar=actual subgroup=product sum
  shape=hexagon caxis=black cframe=CXb0c1f4;
  by quarter;
run;
quit;
data _null_ ;
  file 'u:\public_html\Web_output\gifanim.htm' ;
put '<HTML>';
put '<HEAD>';
put '<TITLE> GIFANIM </TITLE>';
put '</HEAD>';
put '<BODY>';
put '<IMG src="gifanim.gif">';

put '</BODY>';
put '</HTML>';
run;

```

HTML File

The following is the HTML file that is generated by the PUT statements. Of course, instead of embedding PUT statements in a SAS program, you can hand-create your own HTML file using whatever editor you prefer.

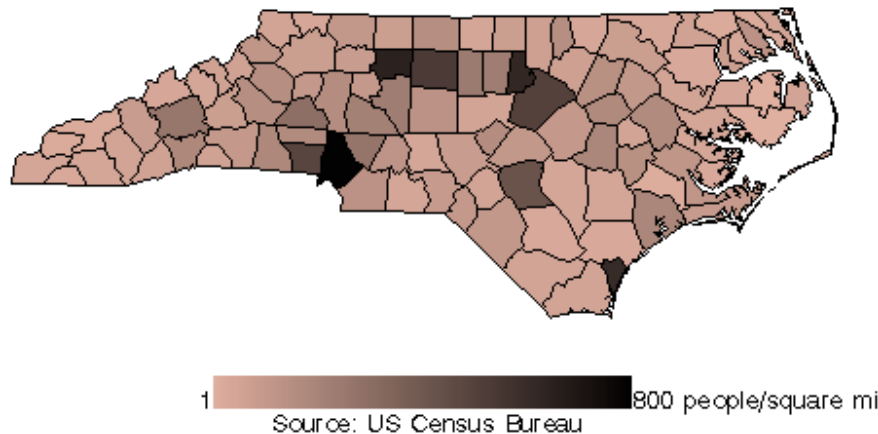
```
<HTML>
<HEAD>
<TITLE> GIFANIM </TITLE>
</HEAD>
<BODY>
<IMG src="gifanim.gif">
</BODY>
</HTML>
```

Generating an Animated Web Presentation with the GIFANIM Device Driver

This example uses the GIFANIM device driver to generate an output file in GIF format. A simple HTML file is also created, using PUT statements. The HTML file enables you to display the GIF file in a Web browser. The browser displays a continuous loop of ten maps of the state of North Carolina, one of which is shown in the following illustration. Successive maps illustrate population growth by changing the color of each county as the population of that county changes over time.

North Carolina Population 1900—1990

1990



The GIFANIM device driver does not provide a pull-down menu or other interactive controls, but it does provide a simple means of translating multiple SAS/GRAPH images into a single automated loop. The resulting GIF output is easily displayed in your own HTML files.

To change the amount of time that each image is displayed, you can change the value of the DELAY= option.

The SAS code for this example is in the SAS Sample Library and is named GWBANIMA.

```

/** Designate a GIF file for the GMAP output.  */
filename animmap 'your_web_path/your_gif.gif';

/** Designate an HTML output file. Use the path */
/** name that you specified for animmap.      */
filename htmlfile 'your_web_path/your_htm.htm';

/** Define a macro variable that points to the */
/** GMAP output. Use the path and file names  */
/** names that you specified for animmap. No  */
/** quotes are needed here.                  */
%let mapsrc=your_web_path/your_gif.gif;
goptions reset=all dev=gifanim gsfmode=replace
          gsfname=animmap xpixels=792 ypixels=600
          transparency iteration=0 delay=150
          disposal=background ftext='Swiss'
          htext=1.5;

/* Create the HTML file that will display the */
/* GIF animation.                            */

data _null_;
file htmlfile;
  source=quote("&mapsrc");
  put '<HTML>';
  put '<BODY>';
  put '<BLOCKQUOTE>';
  put '<img src=' source
      ' alt="NC Population Growth"
      ' width="792" height="600"></P>';
  put '</BLOCKQUOTE>';
  put '</BODY>';
  put '</HTML>';

/** Generate the data for the PROC GMAP      */
/** statements.                             */

data NCPop;
  length Name $ 32;
  input State County P1990 P1980 P1970 P1960
        P1950 / P1940 P1930 P1920 P1910 P1900 /
        AreaLand AreaWater / X Y / Name &;
  SquareMiles = AreaLand * 0.000386102158496;
  Pop1900 = P1900 / SquareMiles;
  Pop1910 = P1910 / SquareMiles;
  Pop1920 = P1920 / SquareMiles;
  Pop1930 = P1930 / SquareMiles;
  Pop1940 = P1940 / SquareMiles;
  Pop1950 = P1950 / SquareMiles;

```

```

Pop1960 = P1960 / SquareMiles;
Pop1970 = P1970 / SquareMiles;
Pop1980 = P1980 / SquareMiles;
Pop1990 = P1990 / SquareMiles;
label
  Pop1900 = '1900 Population Per Square Mile'
  Pop1910 = '1910 Population Per Square Mile'
  Pop1920 = '1920 Population Per Square Mile'
  Pop1930 = '1930 Population Per Square Mile'
  Pop1940 = '1940 Population Per Square Mile'
  Pop1950 = '1950 Population Per Square Mile'
  Pop1960 = '1960 Population Per Square Mile'
  Pop1970 = '1970 Population Per Square Mile'
  Pop1980 = '1980 Population Per Square Mile'
  Pop1990 = '1990 Population Per Square Mile';
format P1990 P1980 P1970 P1960 P1950 P1940 P1930
      P1920 P1910 P1900 comma8.;
label P1990 = '1990 Population'
      P1980 = '1980 Population'
      P1970 = '1970 Population'
      P1960 = '1960 Population'
      P1950 = '1950 Population'
      P1940 = '1940 Population'
      P1930 = '1930 Population'
      P1920 = '1920 Population'
      P1910 = '1910 Population'
      P1900 = '1900 Population';
datalines;
37 1   108213  99319  96362  85674  71220
      57427  42140  32718  28712  25665
      1115485      10649
      -79.398440      36.043667
      Alamance County
37 3   27544  24999  19466  15625  14554
      13454  12922  12212  11592  10960
      674244      7682
      -81.176957      35.921840
      Alexander County
/** See sample GWBANGIF for the full data set.  **/
37 199 15419  14934  12629  14008  16306
      17202  14486  15093  12072  11464
      809243      1752
      -82.310012      35.902682
      Yancey County
;
run;

/** Extract the NC data from the Counties map  **/
/** data in the MAPS library.                **/

data NCC;
  set Maps.Counties;
  where State = 37 and Density <= 3;
run;

```

```

proc gproject data=NCC out=NCCCounty;
           id State County;
run;

/** Produce a color ramp in temp.sas and use    **/
/** the file to generate PATTERN statements.   **/

data _null_;
  file 'temp.sas';
  r = 224; rinc = -r/100;
  g = 176; ginc = -g/100;
  b = 160; binc = -b/100;
  do i = 1 to 99;
    put 'pattern' i ' v=s c=cx' r hex2. g hex2.
      b hex2. ';' ;
    r + rinc;
    g + ginc;
    b + binc;
  end;
run;

%inc 'temp.sas';

/** Create an Annotate data set to    **/
/** produce a legend for the map.     **/

data a;
  length color function style $ 8 text $ 20;
  retain xsys ysys '3' when 'A' style 'S';
  r = 224; rinc = -r/100;
  g = 176; ginc = -g/100;
  b = 160; binc = -b/100;
  x = 25; xinc = 0.5;
  do i = 1 to 99;
    color = 'cx' ||
      put( r, hex2. ) ||
      put( g, hex2. ) ||
      put( b, hex2. );
    function = 'MOVE';
    y = 5;
    output;
    function = 'BAR';
    x + xinc;
    y = 10;
    output;
    r + rinc;
    g + ginc;
    b + binc;
  end;
  function = 'LABEL';
  Style = "'Swiss'";
  Text = '800 people/square mile';
  Position = '6';

```



```

x + xinc;
y = 7.5;
output;
Text = '1';
Position = '4';
x = 24.5;
y = 7.5;
output;
run;

/* Set the title, footnote, and legend. */

title f='Swiss' h=2
      'North Carolina Population 1900-1990';
footnote 'Source: US Census Bureau';

legend1 frame label=(position=(top center));

/** Generate the multiple images that    **/
/** make up the GIF animation.          **/

proc gmap map=NCCounty data=NCPop anno=a;
  id State County;
  title2 f='Swiss' h=3 '1900';
  choro Pop1900 /
  midpoints=8 to 792 by 8 coutline=black
  nolegend;
run;

/* Suppress header information and begin */
/* appending additional images.          */

goption gsfmode=append;

  title2 f='Swiss' h=3 '1910';
  choro Pop1910 /
  midpoints=8 to 792 by 8 coutline=black
  nolegend;
run;

  title2 f='Swiss' h=3 '1920';
  choro Pop1920 /
  midpoints=8 to 792 by 8 coutline=black
  nolegend;
run;

  title2 f='Swiss' h=3 '1930';
  choro Pop1930 /
  midpoints=8 to 792 by 8 coutline=black
  nolegend;
run;

  title2 f='Swiss' h=3 '1940';
  choro Pop1940 /

```

```
midpoints=8 to 792 by 8 coutline=black
nolegend;
run;

title2 f='Swiss' h=3 '1950';
choro Pop1950 /
midpoints=8 to 792 by 8 coutline=black
nolegend;
run;

title2 f='Swiss' h=3 '1960';
choro Pop1960 /
midpoints=8 to 792 by 8 coutline=black
nolegend;
run;

title2 f='Swiss' h=3 '1970';
choro Pop1970 /
midpoints=8 to 792 by 8 coutline=black
nolegend;
run;

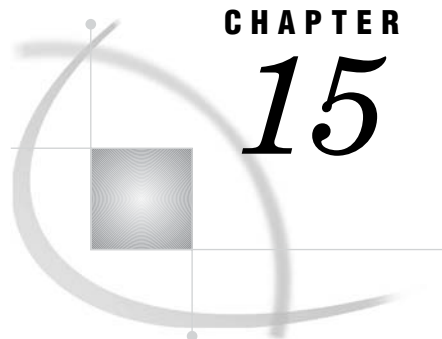
title2 f='Swiss' h=3 '1980';
choro Pop1980 /
midpoints=8 to 792 by 8 coutline=black
nolegend;
run;

/* Mark the end of the animation by      */
/* appending a GIF trailer to the data  */
/* stream.                               */

goptions gepilog='3B'x;

/* Generate the final image.             */

title2 f='Swiss' h=3 '1990';
choro Pop1990 /
midpoints=8 to 792 by 8 coutline=black
nolegend;
run;
quit;
```



CHAPTER

15

Generating Interactive Metagraphics Output

<i>Developing Web Presentations for the Metaview Applet</i>	469
<i>Using ODS with JAVAMETA</i>	470
<i>Using the META2HTM Macro</i>	471
<i>Adding Run-Time Controls to a Presentation</i>	471
<i>Page-Selection Slider Control</i>	472
<i>Slide-Show Control</i>	473
<i>Zoom Control</i>	473
<i>Enhancing Web Presentations for the Metaview Applet</i>	474
<i>Specifying Non-English Resource Files and Fonts</i>	474
<i>Metaview Applet Parameters</i>	475
<i>Specifying Applet Parameters Using the ODS PARAMETERS= Statement</i>	477
<i>META2HTM Macro Arguments</i>	478
<i>Sample Programs: Metaview Applet</i>	478
<i>Metacode Output with HTML from ODS</i>	478
<i>SAS Code</i>	479
<i>Producing a Web Presentation with the META2HTM Macro</i>	481
<i>SAS Code</i>	482
<i>Embedding Multiple Instances of the Metaview Applet on the Same HTML Page with META2HTM</i>	483
<i>SAS Code</i>	484

Developing Web Presentations for the Metaview Applet

The JAVAMETA device driver generates graphs that are stored in metagraphics format and displayed by the Metaview Applet to create interactive Web presentations. Most procedures that generate GRSEG catalog entries are also capable of generating metagraphics output. (For a list of these procedures, see “Metaview Applet” on page 375.) The metacodes that make up the metagraphics format can be stored in metagraphics files, or the metacodes can be embedded directly in the HTML output file. Metacodes are simple ASCII codes that look like the following:

```

37   8 106  97 118  97 109 101 116  97  30   0  10   1  13   5
  0   0   0  50   8  32  32  32  32  32  32  32  32  51  18  57
 46  48  48  46  48  48  77  48  68  48  56  48  49  50  48  48

```

You can use a GOPTIONS statement with a device type of JAVAMETA to create metacode output from one or more SAS/GRAPH procedures. At run time, HTML code passes the metacodes as a parameter to the Metaview Java applet.

The Metaview applet runs with the Java Virtual Machine that is included with Web browser. Unlike the other SAS/GRAPH applets (such as Graph, Constellation, and

Treeview applets) the Metaview applet does not require installation of a Java Plug-in on the client machine.

Interactive features of the Metaview Applet include pan and zoom. Compared to raster images (GIF, JPEG, PNG), the Metaview applet offers faster data tips and the advantage of zooming that increases the graph's resolution rather than degrading it. You can add data tips, specify resource files for language translation, specify background colors and text fonts, and drill down to HTML files, metagraphics files, and sets of metacodes. You can also provide a list of selectable drill-down URLs in the pop-up menu. For information on these enhancements, see "Enhancing Web Presentations for the Metaview Applet" on page 474.

Two methods for generating Metaview applet presentations are

- using ODS with JAVAMETA device driver
- using the META2HTM macro.

To see examples of programs that generate a Web presentation for the Metaview Applet, see "Sample Programs: Metaview Applet" on page 478.

Using ODS with JAVAMETA

The following steps use ODS to develop a Web presentation for the Metaview Applet. The presentation displays a single graph. The metacodes for that graph are embedded in the body of the HTML output file.

- 1 Specify the JAVAMETA device driver.

```
goptions reset=all device=javameta;
```

- 2 Close the ODS listing destination (the Output and Graph windows) to conserve resources.

```
ods listing close;
```

- 3 Open an HTML output file by using an ODS statement and specifying a filename.

```
ods html file="C:\meta\bar.htm";
```

If you need to specify a Java archive location other than the location that is specified as the value of the APPLETLOC= system option, you can use the CODEBASE= option.

```
ods html file="C:\meta\bar.htm"
      codebase="http://ourweb/sasJava";
```

You can enhance your Web presentation by specifying other applet parameters, as described in "Metaview Applet Parameters" on page 475.

- 4 Generate the metacodes and embed those metacodes in the body of the HTML output file.

```
proc gchart data=sashelp.class;
  vbar height / group=age;
run; quit;
```

- 5 Close the HTML file and reopen the listing destination.

```
ods html close;
ods listing;
```

Run the program to generate the HTML output file. The applet may run in the SAS Results window, or you can display the HTML output file in a Web browser.

Using the META2HTM Macro

As an alternative to using ODS statements, you can use the META2HTM macro to generate Web presentations that run in the Metaview Applet.




To use the META2HTM macro, set up your data, call the macro with the arguments that you need, run your graphics procedures, then run the META2HTM macro again.

Macro arguments for META2HTM enable you to format the HTML output file and configure applet enhancements. For information on the arguments of the META2HTM macro, see “META2HTM Macro Arguments” on page 478.

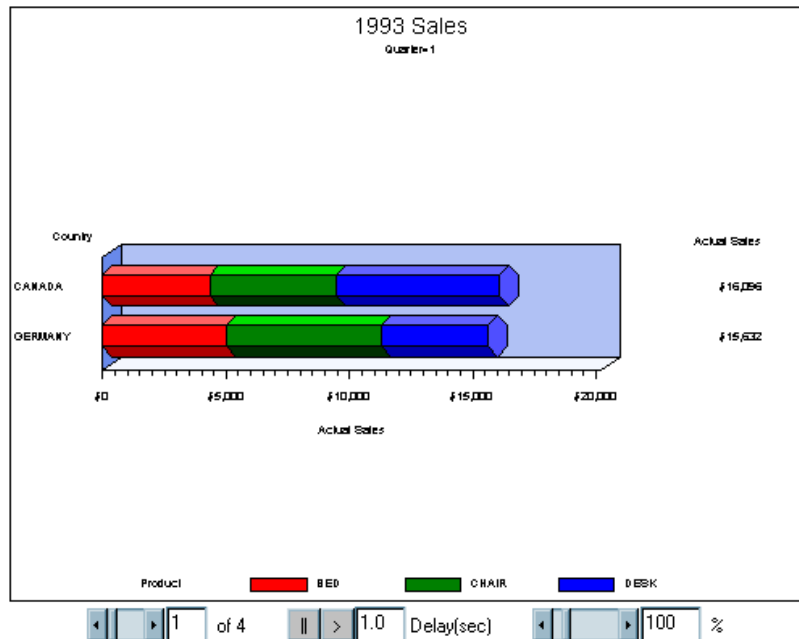
Note that there is no need to compile the Annotate macros in order to access the META2HTM macro.

Adding Run-Time Controls to a Presentation

One thing that distinguishes the Metaview applet is the run-time controls that it provides to users. The following table shows some of the controls that the Metaview applet can present.

	Page-selection slider control
	Slide-show control
	Zoom control

The following picture shows a graphic in which all three controls are present. You can also use parameters to suppress the display of any of the controls.



Page-Selection Slider Control

Metacodes (that are passed with the METACODES parameter to the Metaview applet) can contain multiple graphs when they are the output of a SAS/GRAPH procedure containing a BY statement, or when they are the output (concatenated together) of multiple SAS/GRAPH procedures. Because ODS only passes a single graph at a time with the METACODES parameter to the Metaview applet, the recommended way to enable a page-selection slider control at run time is by using the META2HTM macro.

The following code fragment uses the META2HTM macro. Notice that two instances of PROC GCHART are invoked in between the first call to META2HTM with CAPTURE=ON and the final call to META2HTM with CAPTURE=OFF. Notice also that in the final call to META2HTM the parameter OPENMODE=APPEND so that the metacodes from the second process are appended to those from the first (with embedded delimiters between graphs). In fact, because both GCHART procedures include a BY QUARTER statement, each procedure produces four graphs, for a total of eight graphs in all.

```
filename _webout 'path_and_filename.htm';

%meta2htm(capture=on,
          href=_webout,
          openmode=replace,
          /* Specify codebase if metafile.zip not in same directory */
          /* as html file. */
          codebase=http://web_server_name/sasweb/graph
          archive=metafile.zip,
          hspace=1,
          vspace=2);
```

```

goptions reset=all device=javameta
      border
      ftext="Helvetica" ftitle="Helvetica";
title1 '1993 Sales';
proc gchart data=prdsummary(where=(year=1993));
  hbar3d country / sumvar=actual subgroup=product sum
  shape=hexagon caxis=black cframe=CXb0c1f4;
  by quarter;
run;
quit;

title1 '1994 Sales';
proc gchart data=prdsummary(where=(year=1994));
  hbar3d country / sumvar=actual subgroup=product sum
  shape=hexagon caxis=black cframe=CXb0c1f4;
  by quarter;
run;
quit;
%meta2htm(capture=off,
          htmlfref=_webout,
          openmode=append);

quit;

```

Slide-Show Control

Metacodes (that are passed with the METACODES parameter to the Metaview applet) can contain multiple graphs when they are the output of a SAS/GRAPH procedure containing a BY statement, or when they are the output (concatenated together) of multiple SAS/GRAPH procedures. To suppress the slide-show control, do one of the following:

ODS Specify SLIDECONTROLENABLED=FALSE in the ODS statement, which should look like this:

```
ods html file="filename.htm"
      parameters=( "SLIDECONTROLENABLED=FALSE" );
```

META2HTM Specify SLIDECTL=N. For example:

```
%meta2htm(capture=on,
          htmlfref=_webout,
          openmode=replace,
          slidectl=n,
          ...);
```

Zoom Control

Unless you choose to suppress it, the Metaview applet always displays a zoom control which allows a user to zoom in on and out of the image. To suppress the zoom control, do one of the following:

ODS Specify ZOOMCONTROLENABLED=FALSE in the ODS statement, which should look like this:

```
ods html file="filename.htm"
      parameters=("ZOOMCONTROLENABLED=FALSE");
```

META2HTM Specify ZOOMCTL=N. For example:

```
%meta2htm(capture=on,
           htmlfref=_webout,
           openmode=replace,
           zoomctl=n,
           ...);
```

Enhancing Web Presentations for the Metaview Applet

The JAVAMETA device driver is used to generate interactive Web presentations that run in the Metaview Applet. The Metaview Applet displays and provides interactivity for graphs that have been generated in metagraphics format. This format can be generated by most SAS/GRAPH procedures (For a list, see “Metaview Applet” on page 375), as well as some other such as PROC GANTT.

Programming for the default configuration of the Metaview Applet consists of specifying the JAVAMETA device driver, generating an HTML output file, and generating a graph. For information on programming for this default configuration, see “Developing Web Presentations for the Metaview Applet” on page 469.

You can enhance the default configuration as follows:

- Specify a non-English resource file and font for Java 1.02 presentations. See “Specifying Non-English Resource Files and Fonts” on page 474.
- Display and configure a zoom control. See the applet parameters that begin with ZOOM, in “Metaview Applet Parameters” on page 475.
- Replace the default applet help with application-specific help. Set the applet parameter HELPLOCATION to point to your help files.
- Set the background color by setting the applet parameter BACKGROUNDCOLOR.
- Disable embedded controls that otherwise appear automatically in presentations that include multiple graphs. See the parameters SLIDESHOWCONTROLENABLED and PAGECONTROLENABLED.

Note that you can combine almost all of the available enhancements, including different drill-down modes.

Note that the META2HTM macro enables you to generate enhanced Web presentations by specifying selected macro arguments, as described in “Using the META2HTM Macro” on page 471.

To learn how to specify applet parameters, see “Specifying Applet Parameters Using the ODS PARAMETERS= Statement” on page 477. Reference information on applet parameters is provided in “Metaview Applet Parameters” on page 475.

Specifying Non-English Resource Files and Fonts

The Metaview Applet supports Java 1.02, which is good in that it runs in most browsers. Unfortunately, Java 1.02 does not support the use of resource files and fonts, which would enable the automated use of translated text and localized formats as supported by Java 1.2. To overcome this limitation, the Metaview Applet enables you to name a resource file and a resource font by specifying applet parameters. In this resource file you can hard-code translated versions of the text that the Metaview Applet uses.

Follow these steps to manually translate the text in the Metaview Applet:

- 1 Specify the LOGRESOURCES parameter in your SAS job, generate the HTML, and view it in a browser. (See “Metaview Applet Parameters” on page 475.) The Metaview Applet will then write its tag/value pairs to the Java console.
- 2 Copy the tag/value pairs that you want to translate out of the Java console and paste them into your resources file. Then translate those values to your language. You do not need to translate all of the tag/value pairs. The defaults will be used where translations are not provided.
- 3 Store your resources file in a location that can be accessed by your Web audience.
- 4 In the SAS program, remove the LOGRESOURCES parameter specification. Then specify the RESOURCES parameter. The value of that parameter is the URL of your resources file.

Note that you do not have to specify the RESOURCES parameter if you name the resource file MVAResources.properties and store that file in the same location as the HTML output file.

- 5 If your resources file requires a non-English text font, then specify that font as the value of the parameter RESOURCESFONTNAME. To display this font, your Web audience must have this font installed.
- 6 Run your program and test your Web output.

For information on specifying applet parameters, see “Specifying Applet Parameters Using the ODS PARAMETERS= Statement” on page 477. For reference information on the Metaview Applet parameters, see “Metaview Applet Parameters” on page 475.

Metaview Applet Parameters

The following parameters may be specified for the Metaview Applet. For information on how to specify these parameters, see “Specifying Applet Parameters Using the ODS PARAMETERS= Statement” on page 477.

BACKGROUNDCOLOR=*color*

specifies the background for the applet as an RGB color in hexadecimal. White is 0xffff. Red is 0xff0000. If not specified, the background color is 0xd3d3d3 (gray).

DATATIPHIGHLIGHTCOLOR=*color*

specifies a 6-digit hexadecimal RGB color that is displayed as the outline of the graph element that is displaying its data tip information. The default color is red. This parameter is valid only if the DATATIPSTYLE parameter is set to the value HIGHLIGHT.

DATATIPSTYLE= HIGHLIGHT | STICK | STICK_FIXED

specifies the style of the data tip pop-up window. Values can be:

HIGHLIGHT

causes the data tip to appear above the segment with no connecting line. The border of the graph element is highlighted.

STICK

connects the data tip pop-up window to the graph element with a line. The pop-up window is positioned over the cursor. While the cursor remains in the element, moving the cursor moves the pop-up window and the connecting line.

STICK_FIXED

connects a stationary data tip pop-up window to the graph element with a line drawn into the middle of the graph element.

DEFAULTTARGET=*target-name*

specifies where the browser will display drill-down URLs by default. The value of this parameter can be an HTML target such as `_BLANK` or the name of a window or frame in the Web presentation. The default value is `_BLANK`, which displays drill-down URLs in a new browser window. The value of the `DEFAULTTARGET` parameter is superseded by the optional drill-down tag `TARGET`.

HELPLLOCATION=*URL*

specifies a location for application-specific help that replaces the default help that is provided for the Metaview Applet. The default help location is the SAS web site.

LOGRESOURCES=TRUE | FALSE

specifying a value of `TRUE` logs tag/value pairs in the key definition file. The default value is `FALSE`. The tag value pairs are copied out of the key definition file and modified to create a resource file. The resource file is identified with the `RESOURCES` parameter, which enables the Metaview Applet text to be translated to another language. See also the `RESOURCESFONTNAME` parameter.

METACODES=*codes-or-file-specification*

identifies a text file that contains metagraphics codes, or it provides inline metagraphics codes. The file specification is an absolute or relative URL address.

METACODES1-METACODESn=*codes-or-file-specification*

identifies additional metacode specifications when you need to identify more than one file or more than one set of inline metagraphics codes.

METACODESLABEL=*menu-label***METACODES1LABEL-METACODESnLABEL=*menu-label***

names the text labels that are used to identify the graphs specified in the `METACODES` and `METACODESn` parameters. If specified, there should be as many `METACODESLABEL` parameters as there are `METACODESn` parameters. Always specify `METACODESLABEL` parameters in sequential order (`METACODESLABEL`, `METACODES1LABEL`, `METACODES2LABEL`, and so on). The applet displays the labels in an embedded graph-selection control.

PAGECONTROLENABLED=TRUE | FALSE

enables or disables the display of a scroll control in the Metaview Applet. The applet displays the control by default, when more than one graph is contained in the metacodes set. Specify `FALSE` to disable the scroll control.

RESOURCES=*text-URL*

specifies the relative or absolute URL of an ASCII-formatted resources file. This file enables the translation of the English text that is provided in the Metaview Applet. The resource file is provided so that the Metaview Applet, which is Java 1.02 compliant, can provide translation capabilities that are similar to the resource files that are enabled in Java 1.2. Note that you do not have to specify the `RESOURCES` parameter if you provide a resource file with the name `MVAResources.properties` in the same directory as the HTML output file. The Metaview Applet looks for this file by default and uses it if it is found. Specifying a value for the `RESOURCES` parameter overrides this default applet behavior. For information on creating a resources file, see “Specifying Non-English Resource Files and Fonts” on page 474. See also the parameters `LOGRESOURCES` and `RESOURCESFONTNAME`.

RESOURCESFONTNAME=*font-name*

specifies the name of the font family that is used to display the resource values in a user-defined resource file. This allows the Metaview Applet, which is Java 1.02 compliant, to emulate the language translation capabilities of Java 1.2. The applet first tries to use the specified *font-name*, then it tries to use the SansSerif font, then it tries to use the Serif font, then it uses the first font that is returned by the Java.Awt.Toolkit. The first font that is found is the font that is used. See also the parameters LOGRESOURCES and RESOURCES.

SLIDESHOWCONTROLENABLED=TRUE | FALSE

displays the embedded slide-show control when the current set of metagraphics codes contains more than one graph. The default is TRUE. Displaying the slide-show control allows you to start and stop a loop that displays each graph for a specified amount of time. You can change the amount of time that each graph is displayed. Specifying a value of FALSE prevents the display of the slide-show control.

ZOOMCONTROLENABLED=TRUE | FALSE

displays the embedded zoom control under the graph. The default is TRUE. Specifying a value of FALSE disables the display of the zoom control.

ZOOMCONTROLMIN=*minimum-percentage*

specifies a new lower limit for the zoom feature. The default value is 25 percent of initial size. Valid values range from 1 to 99.

ZOOMCONTROLMAX=*maximum-percentage*

specifies a new upper limit for the zoom feature. The default value is 500 percent of initial size. Valid values range from 100 to 25000.

Specifying Applet Parameters Using the ODS PARAMETERS= Statement

You can control the initial appearance of your Web output and configure aspects of the applet's user interface by specifying applet parameters. The applet parameters are generally specified as follows in the PARAMETERS= option of the ODS statement.

ODS HTML FILE=*HTML-output-file-specification*

```
PARAMETERS=(
    "parameter-name1"="parameter-value1"...
    "parameter-nameN"="parameter-valueN");
```

An example might look like this:

```
ods html file="ncpop.htm"
  parameters=( "DATATIPSTYLE"="STICK"
    "HELPLLOCATION"="http://www.mysite.com/myjavametahelp.htm"
    "ZOOMCONTROLENABLED"="FALSE" );
```

You can specify any number of parameters in a single PARAMETERS= statement. The parameters can be specified in any order. Blank spaces separate multiple parameter specifications. You can also use multiple PARAMETERS= statements within a given ODS statement. The quotation marks and parentheses are required. Additional quotation marks are required in the specification of certain parameter values.

META2HTM Macro Arguments

The META2HTM macro generates Web presentations that run in the Metaview Applet. This applet displays and provides interactive features for graphs that have been stored in metagraphics format. The macro is provided as an alternative to using ODS to generate the requisite HTML files. For information on programming for the Metaview Applet, see “Developing Web Presentations for the Metaview Applet” on page 469.

The syntax of the META2HTM macro is as follows:

```
%META2HTM(argument1=value1, argument2=value2, ...);
```

The macro arguments are divided into the following categories:

- “Arguments for the APPLET Tag” on page 536
(the ARCHIVE and CODEBASE arguments are required)
- “META2HTM Arguments for Saving the HTML File” on page 564
- “Arguments for Page Formatting” on page 552
- “Arguments for Stylesheets” on page 554
- “Arguments for the SAS TITLE and FOOTNOTE Tags” on page 556
- “Arguments for Character Transcoding” on page 561
- “META2HTM Arguments for Applet Behavior” on page 565.

Sample Programs: Metaview Applet

The following sample programs use DEVICE=JAVAMETA to generate metacodes to be displayed by the Metaview applet:

- “Metacode Output with HTML from ODS” on page 478
- “Producing a Web Presentation with the META2HTM Macro” on page 481
- “Embedding Multiple Instances of the Metaview Applet on the Same HTML Page with META2HTM” on page 483.

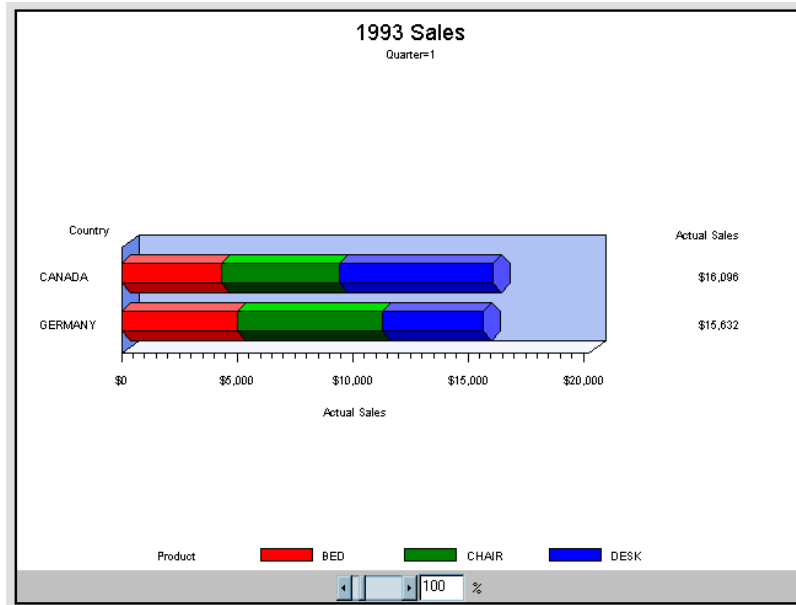
Metacode Output with HTML from ODS

The following sample program uses ODS to create an HTML file, and GOPTIONS DEVICE=JAVAMETA with two instances of PROC GCHART to create graphical output in the form of metacodes. Because both instances of PROC GCHART contain a BY statement, the HTML file created by ODS contains multiple invocations of the applet—one invocation for each value of the BY statement for each procedure (eight invocations in all). The metacodes produced by PROC GCHART are passed to the applet as a parameter.

When you use DEVICE=JAVAMETA with ODS, only one graph can be passed to an instance of the Metaview applet at a time. ODS generates a separate invocation of the Metaview applet for each SAS/GRAPH procedure that it runs. And, if a procedure includes BY GROUP processing, then it generates another separate invocation of the Metaview applet for each BY-group chart. In sum, Metaview applet presentations generated by ODS never contain a slider page control or drop-down list graph control to allow a user to select which graph is to be displayed. Although an HTML page generated by ODS can contain multiple instances of the Metaview applet, each instance can display one picture only, and a user must scroll the HTML page to see all the pictures.

Each GCHART procedure in this example includes a BY statement to display the results of each of the four quarters of the year. Consequently, ODS generates eight

separate invocations of the Metaview applet, only the first of which is shown here. A user would have to scroll the page in the browser to see all four quarters displayed. Notice the slider control at the bottom of the image. Because the image is displayed by the Metaview, the run-time option is available to the user to control the magnification of the chart.



SAS Code

The following is the complete SAS code to generate a Web presentation. You should notice the following:

- PROC GREPLAY is used to delete the GSEGS that are created. This is not necessary, but otherwise SAS creates new GSEGS each time the procedure is run, rather than overwriting the old ones.
- The HTML file is created using ODS HTML.
- The FILE= option of the ODS statement specifies the path and file name of the HTML file to be created. If you run this example, then change the value of the option to something that makes sense for you.
- The statement GOPTIONS DEVICE=JAVAMETA causes PROC GCHART to produce metacodes which are embedded in the HTML file produced by ODS and passed to the Metaview applet as parameters.

```
data prdsummary;
  input Year Quarter Country $8. Product $6. Actual dollar10.2;
  label Actual = 'Actual Sales';
  format Actual dollar11.;
  datalines;
1993 1 CANADA BED $4,337.00
1993 1 CANADA CHAIR $5,115.00
1993 1 CANADA DESK $6,644.00
1993 1 GERMANY BED $5,026.00
1993 1 GERMANY CHAIR $6,276.00
1993 1 GERMANY DESK $4,330.00
```

```

1993 2 CANADA BED $2,437.00
1993 2 CANADA CHAIR $3,115.00
1993 2 CANADA DESK $5,654.00
1993 2 GERMANY BED $3,026.00
1993 2 GERMANY CHAIR $2,276.00
1993 2 GERMANY DESK $3,320.00
1993 3 CANADA BED $6,337.00
1993 3 CANADA CHAIR $7,145.00
1993 3 CANADA DESK $7,614.00
1993 3 GERMANY BED $5,026.00
1993 3 GERMANY CHAIR $3,276.00
1993 3 GERMANY DESK $6,340.00
1993 4 CANADA BED $9,337.00
1993 4 CANADA CHAIR $2,115.00
1993 4 CANADA DESK $3,646.00
1993 4 GERMANY BED $6,026.00
1993 4 GERMANY CHAIR $7,276.00
1993 4 GERMANY DESK $8,350.00
1994 1 CANADA BED $3,327.00
1994 1 CANADA CHAIR $5,345.00
1994 1 CANADA DESK $7,624.00
1994 1 GERMANY BED $4,026.00
1994 1 GERMANY CHAIR $3,276.00
1994 1 GERMANY DESK $3,340.00
1994 2 CANADA BED $5,356.00
1994 2 CANADA CHAIR $3,115.00
1994 2 CANADA DESK $7,623.00
1994 2 GERMANY BED $8,026.00
1994 2 GERMANY CHAIR $5,276.00
1994 2 GERMANY DESK $7,321.00
1994 3 CANADA BED $4,321.00
1994 3 CANADA CHAIR $3,115.00
1994 3 CANADA DESK $5,658.00
1994 3 GERMANY BED $6,026.00
1994 3 GERMANY CHAIR $5,276.00
1994 3 GERMANY DESK $6,398.00
1994 4 CANADA BED $5,357.00
1994 4 CANADA CHAIR $4,166.00
1994 4 CANADA DESK $7,662.00
1994 4 GERMANY BED $4,026.00
1994 4 GERMANY CHAIR $5,246.00
1994 4 GERMANY DESK $3,329.00
;
/* Delete previously created gsegs before creating new ones. */
proc greplay igout=work.gseg nofs;
  delete _all_;
  /* Could also specify: delete gchart, gchart1, etc. */
run; quit;

ods html file='u:\public\Web_output\ods_javameta_exp.htm';
goptions reset=all device=javameta
  border
  ftext="Helvetica" ftitle="Helvetica";

```

```

title1 '1993 Sales';
proc gchart data=prdsummary(where=(year=1993));
  hbar3d country / sumvar=actual subgroup=product sum
  shape=hexagon caxis=black cframe=blue;
  by quarter;
run;
quit;

title1 '1994 Sales';
proc gchart data=prdsummary(where=(year=1994));
  hbar3d country / sumvar=actual subgroup=product sum
  shape=hexagon caxis=black cframe=blue;
  by quarter;
run;
quit;

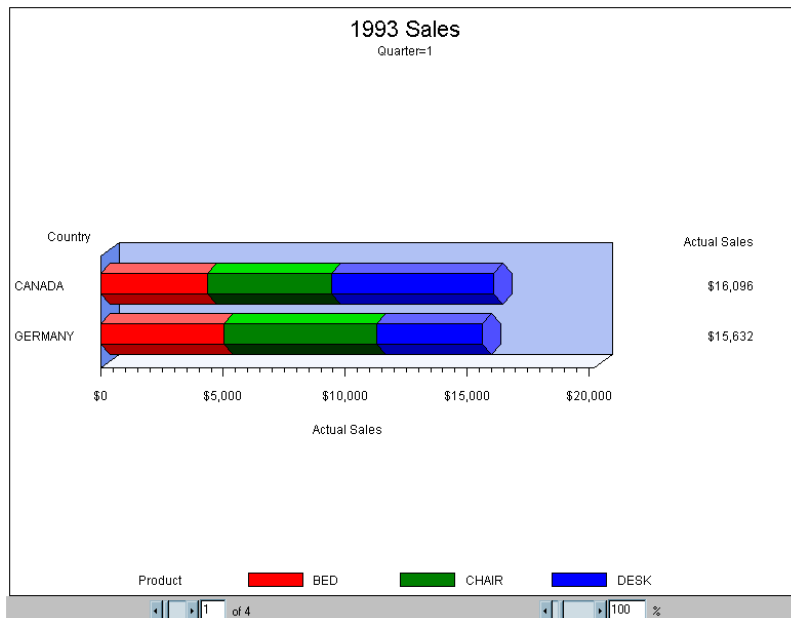
ods html close;

```

Producing a Web Presentation with the META2HTM Macro

The following sample program uses the META2HTM macro to create an HTML file, and GOPTIONS DEVICE=JAVAMETA with PROC GCHART to create graphical output in the form of metacodes. When you use the META2HTM macro, the metacodes produced by a SAS/GRAPH procedure are embedded in the HTML file. This enables you to display multiple charts with one invocation of the Metaview applet.

The sample codes contains one invocation of PROC GCHART with a BY statement to produce charts for each quarter of the year. However, each of the four charts is displayed in sequence on a single output area of the same HTML page (no scrolling is necessary). The Metaview applet provides a slider control, which allows a user to select which quarter to display.



SAS Code

The following is the complete SAS code to generate a web presentation. You should notice the following:

- The statement `FILENAME _WEBOUT` specifies the name of the HTML file to be produced by the `META2HTM` macro. When `GOPTIONS DEVICE=JAVAMETA`, the output of a `SAS/GRAPH` procedure is directed to the file specified by `_WEBOUT`. Because the `META2HTM` macro produces an HTML file, the metacodes produced by the `SAS/GRAPH` procedure are embedded in the HTML file. If you run this sample, change the value of `_WEBOUT` to something that makes sense for you.
- The `META2HTM` macro is invoked twice—once before the `SAS/GRAPH` procedure in order to specify parameters for the procedure, and a second time after the procedure to close the HTML file created.

```
data prdsummary;
  input Year Quarter Country $8. Product $6. Actual dollar10.2;
  label Actual = 'Actual Sales';
  format Actual dollar11.;
  datalines;
1993 1 CANADA BED $4,337.00
1993 1 CANADA CHAIR $5,115.00
1993 1 CANADA DESK $6,644.00
1993 1 GERMANY BED $5,026.00
1993 1 GERMANY CHAIR $6,276.00
...more data lines...
1994 4 CANADA CHAIR $4,166.00
1994 4 CANADA DESK $7,662.00
1994 4 GERMANY BED $4,026.00
1994 4 GERMANY CHAIR $5,246.00
1994 4 GERMANY DESK $3,329.00
;
run;

/* When goptions device=javameta, the procedure output goes to _webout. */
/* In this case the metacodes are embedded in the html file. */
filename _webout 'u:\public\Web_output\meta2htm_javameta_sample1.htm';

%meta2htm(capture=on,
  htmlfref=_webout,
  openmode=replace,
  /* Specify codebase if metafile.zip is not in same */
  /* directory as the html file. */
  codebase=http://web_server_name/sasweb/graph
  archive=metafile.zip,
  slidect1=n,
  /* don't advance pictures automatically like a slideshow */
  hspace=1,
  vspace=2);

goptions reset=all device=javameta
  border
  ftext="Helvetica" ftitle="Helvetica";
title1 '1993 Sales';
```



```

proc gchart data=prdsummary(where=(year=1993));
  hbar3d country / sumvar=actual subgroup=product sum
  shape=hexagon caxis=black cframe=blue;
  by quarter;
run;
quit;

%meta2htm(capture=off,
          htmlfref=_webout,
          openmode=append);

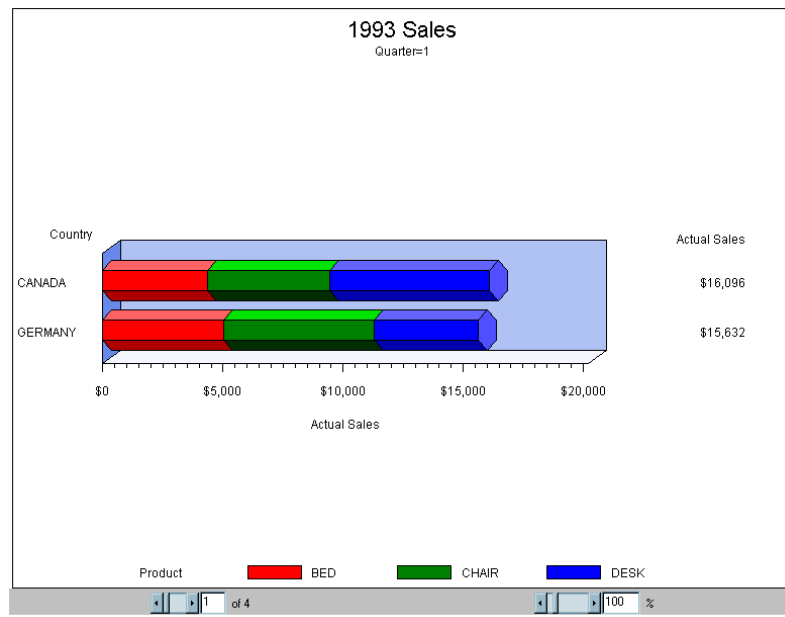
quit;

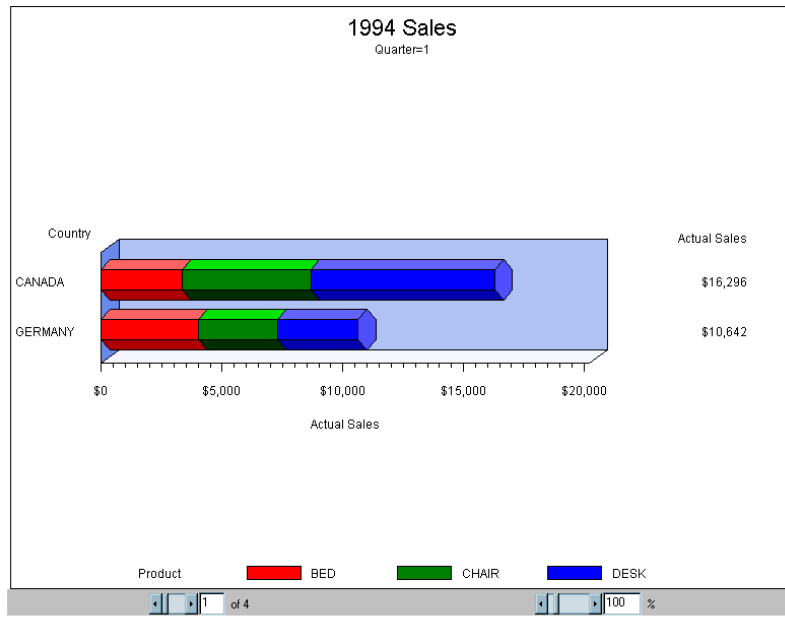
```

Embedding Multiple Instances of the Metaview Applet on the Same HTML Page with META2HTM

Using the META2HTM macro, you can embed multiple instances of the Metaview applet in a single HTML page, and for each instance you can display the output of a different SAS/GRAPH procedure, as illustrated in the current example.

The sample code contains two invocations of PROC GCHART, each invocation with a BY statement, to produce a total of eight charts (four quarters per year times two years). The Metaview applet is invoked twice, and each time provides a slider control, which enables a user to select which quarter to display for that particular year.





SAS Code

The following is the complete SAS code to generate a Web presentation. You should notice the following:

- The statement `FILENAME _WEBOUT` specifies the name of the metacode file to be produced by `PROC GCHART`. When `GOPTIONS DEVICE=JAVAMETA`, the output of a SAS/GRAPH procedure is directed to the file specified by `_WEBOUT`. If you run this sample, change the value of `_WEBOUT` to something that makes sense for you.
- When the `META2HTM` macro is invoked prior to the second occurrence of `PROC GCHART`, it is invoked with the parameter `OPENMODE=APPEND`, so that the second invocation of the Metaview applet is included in the same HTML file as the first one.

```
data prdsummary;
  input Year Quarter Country $8. Product $6. Actual dollar10.2;
  label Actual = 'Actual Sales';
  format Actual dollar11.;
  datalines;
1993 1 CANADA BED $4,337.00
1993 1 CANADA CHAIR $5,115.00
1993 1 CANADA DESK $6,644.00
1993 1 GERMANY BED $5,026.00
1993 2 GERMANY BED $3,026.00
...more data lines...
1994 4 CANADA CHAIR $4,166.00
1994 4 CANADA DESK $7,662.00
1994 4 GERMANY BED $4,026.00
1994 4 GERMANY CHAIR $5,246.00
1994 4 GERMANY DESK $3,329.00
;
```

```
/* When goptions device=javameta, the output of the procedure */
/* goes to _webout. */
/* In this case, the metcodes output is embedded in the html */
/* file produced by meta2htm. */
filename _webout 'u:\public\Web_output\meta2htm_javameta_sample2.htm';

%meta2htm(capture=on,
          htmlfref=_webout,
          openmode=replace,
          /* Specify codebase if metafile.zip is not in same directory as the */
          /* html file. */
          codebase=http://web_server_name/sasweb/graph
          archive=metafile.zip,
          pagepart=head,
          slidectl=n,
          /* don't advance pictures automatically like a slideshow */
          hspace=1,
          vspace=2);

goptions reset=all device=javameta
          border
          ftext="Helvetica" ftitle="Helvetica";
title1 '1993 Sales';
proc gchart data=prdsummary(where=(year=1993));
  hbar3d country / sumvar=actual subgroup=product sum
  shape=hexagon caxis=black cframe=blue;
  by quarter;
run;
quit;

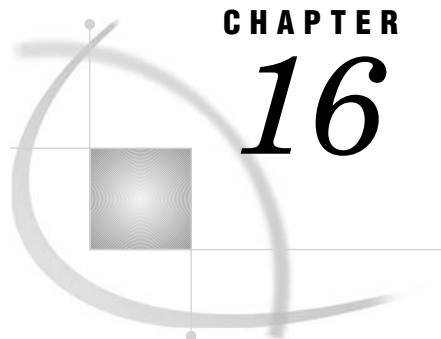
%meta2htm(capture=off,
          htmlfref=_webout,
          openmode=append,
          pagepart=body);

%meta2htm(capture=on,
          htmlfref=_webout,
          archive=metafile.zip,
          openmode=append,
          slidectl=n,
          /* don't advance pictures automatically like a slideshow */
          pagepart=body);

title1 '1994 Sales';
proc gchart data=prdsummary(where=(year=1994));
  hbar3d country / sumvar=actual subgroup=product sum
  shape=hexagon caxis=black cframe=blue;
  by quarter;
run;
quit;

%meta2htm(capture=off,
```

```
        htmlfref=_webout,  
        openmode=append,  
        pagepart=foot);  
  
quit;
```



CHAPTER

16

Managing Web Output with ODS

<i>Overview of ODS Enhancements for Web Output</i>	487
<i>Using ODS Styles</i>	488
<i>Managing ODS Destinations</i>	489
<i>ODS and Procedures that Support RUN-Group Processing</i>	490
<i>Specifying Body Files for Displaying Graphs</i>	491
<i>Controlling Titles and Footnotes with ODS Output</i>	492
<i>Controlling Where Titles and Footnotes are Rendered</i>	492
<i>Controlling the Text Font, Size, and Color</i>	493
<i>Using Graphics Options with ODS</i>	493
<i>Adding Non-Graphics Output to a Web Page</i>	494
<i>Linking to Output through a Table of Contents</i>	495
<i>Linking to Output through a Table of Pages</i>	496
<i>Using Frames to Display ODS Output</i>	497

Overview of ODS Enhancements for Web Output

Using ODS with SAS/GRAPH has numerous advantages over generating HTML output by other means. With ODS, you can

- ❑ specify parameters for presentations that run in the Java applets or in the SAS/GRAPH Control for ActiveX.
- ❑ use ODS styles to enhance the appearance of your graphs with images, color gradients and blends, transparency, and shading (see “Using ODS Styles” on page 488).
- ❑ name the body file(s) for storing the ODS output (see “Specifying Body Files for Displaying Graphs” on page 491).
- ❑ determine whether titles and footnotes are written as part of the graphs or as part of the HTML files (see “Controlling Titles and Footnotes with ODS Output” on page 492).
- ❑ combine graphics and non-graphics output in your Web page (see “Adding Non-Graphics Output to a Web Page” on page 494).
- ❑ generate a Table of Contents to link to the output (see “Linking to Output through a Table of Contents” on page 495).
- ❑ generate a Table of Pages to link to the output (see “Linking to Output through a Table of Pages” on page 496).
- ❑ use HTML frames to display the Table of Contents or Table of Pages (see “Using Frames to Display ODS Output” on page 497).

At a minimum, to use ODS with SAS/GRAPH, you must do all of the following:

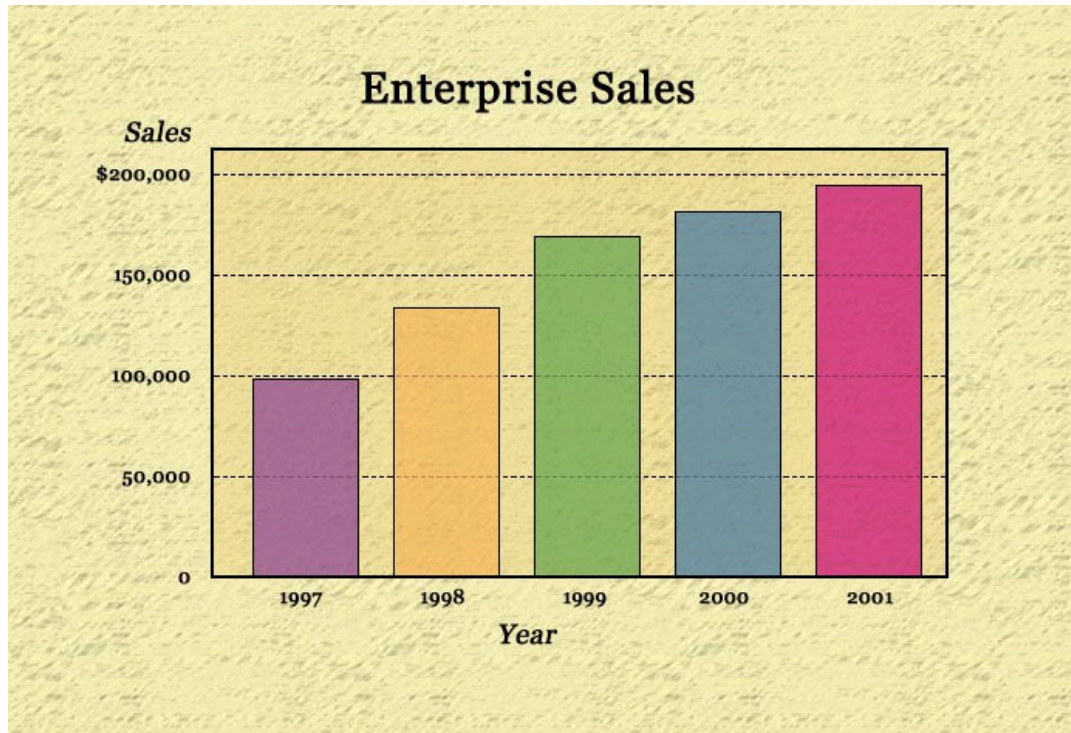
- 1 Use a GOPTIONS statement to specify a device driver with the DEVICE= graphics option.
- 2 Open an HTML output file using an ODS statement such as ODS HTML or ODS MARKUP. At a minimum, you must use the BODY= (alias FILE=) option to specify a body file. For device drivers that generate image output files, use the PATH= or GPATH= option to ensure that all output files are stored in the same location.
- 3 Run a graphics procedure.
- 4 Close the HTML destination.

There are many ODS statements that enable many types of output. There are also a number of ODS options other than the ones that are discussed here that can be used to configure HTML output. For further information, refer to the *SAS Output Delivery System: User's Guide*.

Using ODS Styles

You can use ODS styles to enhance the appearance of your graphical output. The styles provide a consistent look and visual theme, using color schemes, image files, enhanced fonts, transparency, shading, and other appearance enhancements, such as anti-aliasing. SAS provides a number of styles, and you can create your own, as described in the PROC TEMPLATE information in the *SAS Output Delivery System: User's Guide*.

Display 16.1 Example of the Sketch Style



Note: ODS graph styles are available only with the Java and ActiveX devices and are not supported by the Contour Applet. \triangle

To use a style, specify the `STYLE=` option in an ODS statement that generates HTML output. To modify or create a new style, use the `DEFINE STYLE` statement in the `TEMPLATE` procedure.

Predefined graph styles have been developed for particular industries, businesses, or visual themes. Here are a few examples of style names:

Analysis
Astronomy
Banker
Blockprint
Curve
Education
Electronics
Money
Science
Statistical
Watercolor

To view the list of all styles available, run the following code:

```
proc template;
  list styles;
run;
```

For more information on viewing the style definitions that are shipped with SAS software, see *SAS Output Delivery System: User's Guide*.

ODS styles act as a “graphical stylesheet” for standardization purposes. The visual enhancements that you can make with styles allow you tailor the appearance of your graphs to the needs of your presentation and your audience.

Note: Certain ODS styles map textures onto graph elements. For the Java devices, these textures can be applied to 2D rectangles only. Therefore, styles with textures cannot be applied to three-dimensional bar and pie charts in Java graphs. \triangle

For troubleshooting information on graph styles, see Table 23.1 on page 579.

Managing ODS Destinations

ODS supports multiple destinations for procedure output. When using ODS with SAS/GRAPH, you manage two of those locations: the *listing* destination, which is the destination that receives traditional SAS output, and the *HTML* destination, which receives the HTML and image files needed for Web output.

ODS destinations can be open or closed. When a destination is open, ODS can send output to it, and when a destination is closed, ODS cannot send output to it. An open destination always uses system resources.

By default, the listing destination is open and the HTML destination is closed. Specifying an ODS statement that generates HTML output opens the HTML destination, but it has no effect on the listing destination, which remains open unless you explicitly close it.

When using SAS/GRAPH procedures with ODS, you should conserve system resources by closing the listing destination before issuing the ODS statement. After generating ODS output, you must close the HTML destination before you can view that output. A typical ODS session with SAS/GRAPH should be structured like this:

```

/* specify the output location */
filename odsout 'path-to-Web-server';
goptions device=gif;

/* close the listing destination */
ods listing close;

/* open the html destination */
ods html path=odsout body='myfile.html';

/* SAS/GRAPH program code */

/* close the html destination */
ods html close;

/* open the listing destination */
ods listing;

```

If both the listing destination and the HTML destination are open when you use the GIF device driver to generate graphics for the Web, the following output is generated:

HTML Destination	Listing Destination
The GIF driver creates a GIF file for each graph. Each file's name corresponds to the name of the GRSEG entry for the same graph.	The GIF driver creates a GIF file named <i>sasgraph.gif</i> and writes or appends to it (depending on the GSFMODE= setting) each time it generates a graph.
All the HTML files specified on the ODS statement are created. Only the GIF files that are created in the HTML destination are referenced in the HTML files.	

Output in the Listing destination is superfluous for Web use for the following reasons:

- The file *sasgraph.gif* is not referenced in an HTML file.
- Previewing GRSEG entries in the GRAPH window is not a reliable way to proofread the graphs for use with the Web because the GRAPH window and a Web browser render graphs differently.

Note: For more information on ODS destinations, see *SAS Output Delivery System: User's Guide*. \triangle

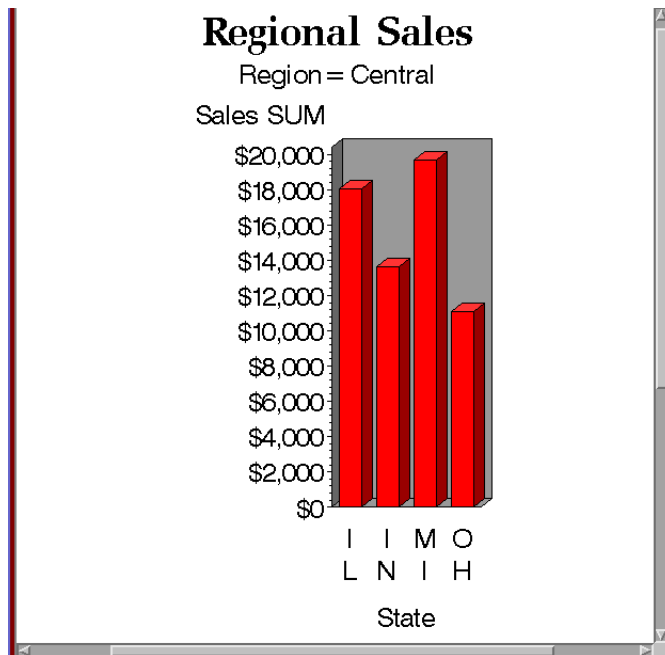
ODS and Procedures that Support RUN-Group Processing

When you use ODS, it is wise to specify a QUIT statement at the end of every procedure that supports RUN-group processing. If you end every procedure step explicitly, rather than waiting for the next PROC or DATA step to end it for you, then the QUIT statement will cause the selection list to clear, and you will be less likely to encounter unexpected results.

Specifying Body Files for Displaying Graphs

When you use ODS with SAS/GRAPH, you specify a body file to reference the graphics output. A body file is simply an HTML file that is created by ODS to contain non-graphics output, and to reference graphics output so that it displays as if it were part of the HTML file. You can use many body files during the SAS session, although only one at a time can be open. The following table shows a body file that references charts that were generated by PROC GCHART.

Display 16.2 Displaying a Body File in a Browser



To open a body file, use the ODS statement option `BODY=`. The following code creates a body file named `sales.html`, which is created in the output location specified on the `FILENAME` statement.

```
filename odsout 'path-to-Web-server';
options device=gif;
ods html body='sales.html' path=odsout;
```

The body file remains open and all graphics and non-graphics output is written to it until the HTML destination is closed or another body file is opened.

To direct output to multiple body files, use an ODS statement with the `BODY=` option each time you want to close the current body file and open another:

```
filename odsout 'path-to-Web-server';
options device=gif;

ods html body='sales.html' path=odsout;

/* code whose output goes to sales.html */
```

```
ods html body='costs.html' path=odsout;

/* code whose output goes to costs.html */

ods html close;
```

Using the PATH= option puts all output files in one location. Using PATH= and GPATH= puts the image output files in a different location.

If you use BY-group processing on a graphics procedure, a separate graph is generated for each value of the BY variable. In that case, all the graphs will be referenced in the same body file, unless you use the ODS statement's NEWFILE= option. For example, you might use NEWFILE=OUTPUT:

```
/* use newfile= to open a new */
/* body file for each graph */
filename odsout 'path-to-Web-server';
goptions device=gif;
ods html body='sales.html' path=odsout
    newfile=output;
```

NEWFILE=OUTPUT opens a new body file for each new graph that is generated, whether the graphs are generated with BY-group processing or by multiple procedure runs. The new body files are named by appending consecutive numbers to the name you specify in the BODY= option. In the example above, the initial body file is named sales.html, and the additional body files that are created will be named sales1.html, sales2.html, and so on.

To determine the appearance of output on the Web page, the body file uses table definitions and style definitions. This document shows output with the default definitions. Other definitions are available with the STYLE= option. You can also create your own style definitions. For information on table definitions and style definitions, see *SAS Output Delivery System: User's Guide*.

Controlling Titles and Footnotes with ODS Output

When you use ODS to send your graphs to an HTML destination, you can choose whether titles and footnotes are rendered as part of the HTML body file, as they are with tabular output, or the graphical image that appears in the Web page.

Where titles and footnotes are rendered determines how you control their font, size, and color.

Controlling Where Titles and Footnotes are Rendered

Where titles and footnotes are rendered depends on the device driver that you are using and on the setting of the ODS statement options GTITLE and GFOOTNOTE.

For the JAVA, JAVAIMG, ACTIVEEX, and ACTXIMG device drivers, titles and footnotes are always rendered as part of the HTML body file. The GTITLE and GFOOTNOTE options are ignored for these drivers.

For all other devices, the GTITLE and GFOOTNOTE options determine where the titles and footnotes are rendered. The default settings, GTITLE and GFOOTNOTE, render titles and footnotes as part of the graphic image. If you want titles and footnotes to appear within the HTML body file and not as part of the graphical image, you must specify the NOGTITLE or NOGFOOTNOTE option, as in the following example.

```

/* direct titles and footnotes */
/* to the HTML body file      */
filename odsout 'path-to-Web-server';
goptions device=gif;
ods html body='sales.html' path=odsout
      nogtitle nogfootnote;

```

If the title or footnote is being output through an ODS markup destination (such as HTML) and the corresponding ODS option NOGTITLE or NOGFOOTNOTE is specified, then the title or footnote is rendered in the body of the HTML file rather than in the graphic itself. Specifying NOGTITLE or NOGFOOTNOTE results in increasing the amount of space allowed for the procedure output area, which can result in increasing the size of the graph. Space that would have been used for the title or footnote is devoted instead to the graph. You might need to be aware of this possible difference if you are using annotate or map coordinates.

Controlling the Text Font, Size, and Color

When you use ODS to send graphics to an HTML destination, and titles and footnotes are rendered as part of the HTML body file instead of the graphic image, then SAS looks for information about how to format titles and footnotes in the following order:

- 1 SAS looks for options on the TITLE and FOOTNOTE statement. For example, you can specify BOLD, ITALIC, FONT=, or HEIGHT= options on these statements.
- 2 SAS looks for global options such as CTEXT and FTITLE on the GOPTIONS statement. For more information, see “Using Graphics Options with ODS” on page 493.
- 3 SAS looks for information specified in the style being used. If you did not specify a style on the ODS statement, then SAS uses information specified in the default style.

When titles and footnotes are rendered as part of the graphic image, SAS looks first for options on the TITLE and FOOTNOTE statement and then for options on the GOPTIONS statement. When titles and footnotes are rendered as part of the graphic image, you do not need to specify the ODS USEGOPT statement.

When titles and footnotes are rendered as part of the body of the HTML file, font sizes that are specified as a percentage are interpreted as a percentage of the size specified by the current style. When titles and footnotes are rendered as part of the image, font sizes that are specified as a percentage are interpreted as a percentage of graphics output area. For more information about specifying fonts and font sizes, refer to

- “FTEXT” on page 294 and “FTITLE” on page 294
- “HTEXT” on page 316 and “HTITLE” on page 317
- “GUNIT” on page 309
- “TITLE, FOOTNOTE, and NOTE Statements” on page 210.

Using Graphics Options with ODS

When you use ODS to send graphics to an HTML destination, and titles and footnotes are rendered as part of the HTML body file instead of the graphic image, ODS will not recognize the settings for the following graphics options unless you also specify the ODS USEGOPT statement:

- CTEXT=
- CTITLE=

- FTEXT=
- FTITLE=
- HTEXT=
- HTITLE=

For example, the following code generates two graphs. The title for the first graph uses the text color and font as defined by the current style (ASTRONOMY). The title for the second graph uses the font size and color specified by the HTITLE and CTEXT options.

```
ods html file="C:\Public\myout1.htm" style=astronomy;
goptions reset=all dev=activex htitle=8 ctext="black";

ods nousegopt;
title 'My title';
footnote 'My footnote';
proc gchart data=sashelp.class;
  pie age / discrete legend;
run;

ods usegopt;
  pie sex / legend;
run;

quit;
ods nousegopt;
ods html close;
```

While ODS USEGOPT is in effect, the settings for these graphics options will affect all of the titles and footnotes rendered by ODS. To turn off the use of these graphics option settings for nongraphic output, specify the ODS NOUSEGOPT statement.

The default setting is ODS NOUSEGOPT.

Adding Non-Graphics Output to a Web Page

When you open a body file in ODS, all graphics and non-graphics output is referenced in that body file until the HTML destination is closed or another body file is opened. Thus, you do not have to do anything special to combine graphics and non-graphics output. Simply open a body file and run the procedures whose output you want to combine:

```
filename odsout 'path-to-Web-server';
goptions device=java;

/* close the listing destination */
ods listing close;

/* open html destination and a body file */
ods html
  body='sales.html'
  style=money
  path=odsout;

/* generate graphics and */
/* non-graphics output  */
```

```

/* using the current data set and style */
proc gchart;
    vbar3d state / sumvar=sales;
run;
quit;

proc print noobs;
run;

/* close the html destination, */
ods html close;
/* open the listing destination */
ods listing;

```

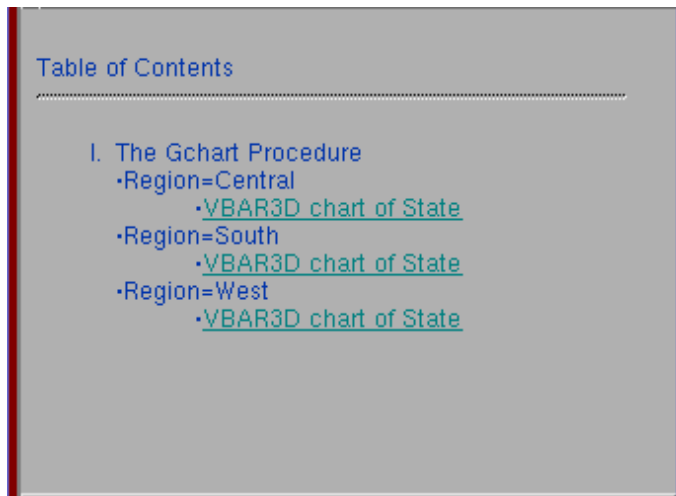
In this example, the GCHART procedure output is referenced in the body file, and then the PRINT procedure output is written below it. The two outputs appear together on the same Web page when file sales.html is viewed in a browser.

For a more complete example, see “Example 9. Combining Graphs and Reports in a Web Page” on page 248.

Linking to Output through a Table of Contents

With ODS, you can create a contents file to link to the graphics and non-graphics output generated during a SAS session. A contents file is simply a file that uses a Table of Contents to link to the output. You can use multiple contents files during the SAS session, although only one at a time can be open.

Display 16.3 Displaying a Contents File in a Browser



To create a contents file, specify a name for the file in the ODS statement option CONTENTS=.

The following code creates a contents file named salesCon.html, which is created in the output location specified in the FILENAME statement.

```

filename odsout 'path-to-Web-server';
goptions device=gif;

```

```
ods html body='sales.html' path=odsout
  contents='salesCon.html';
```

The contents file remains open and links are written to it for all graphics and non-graphics output that is generated by the SAS program until one of the following occurs:

- The HTML destination is closed.
- Another contents file is opened.

To open a new contents file, specify another ODS statement and use CONTENTS= option to specify the new filename.

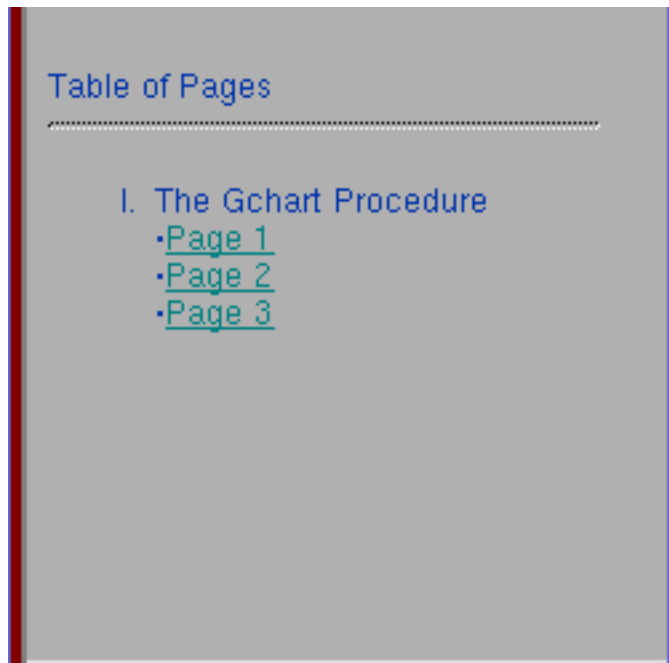
For graphics procedures, use the ODS DESCRIPTION= option to specify the text to be displayed for the links to that procedure's output. If you don't use the DESCRIPTION= option, the procedure's default description text is used.

To use the Table of Contents, view the contents file in the browser. When you select a link from the Table of Contents, the browser goes to the target output referenced by that link. To use the Table of Contents again, you must use the browser's [Back](#) button or some other mechanism to return to the contents page. If your browser supports HTML frames, you can keep the Table of Contents visible and its links accessible at all times by displaying the contents page in a frame (see "Using Frames to Display ODS Output" on page 497).

Linking to Output through a Table of Pages

With ODS, you can create a page file to link to the graphics and non-graphics output generated during a SAS session. A page file is simply a file that uses a Table of Pages (page references) to link to output. You can use multiple page files during the SAS session, although only one at a time can be open.

Display 16.4 Displaying a Page File in a Browser



To create a page file, specify a name for the file in the ODS statement option `PAGE=`. The following code creates a page file named `salePage.html`, which is created in the output location specified in the `FILENAME` statement.

```
filename odsout 'path-to-Web-server';
options device=gif;
ods html body='sales.html' path=odsout
      page='salePage.html';
```

The page file remains open and links are written to it for all graphics and non-graphics output that is generated by the SAS program until one of the following occurs:

- The HTML destination is closed.
- Another page file is opened.

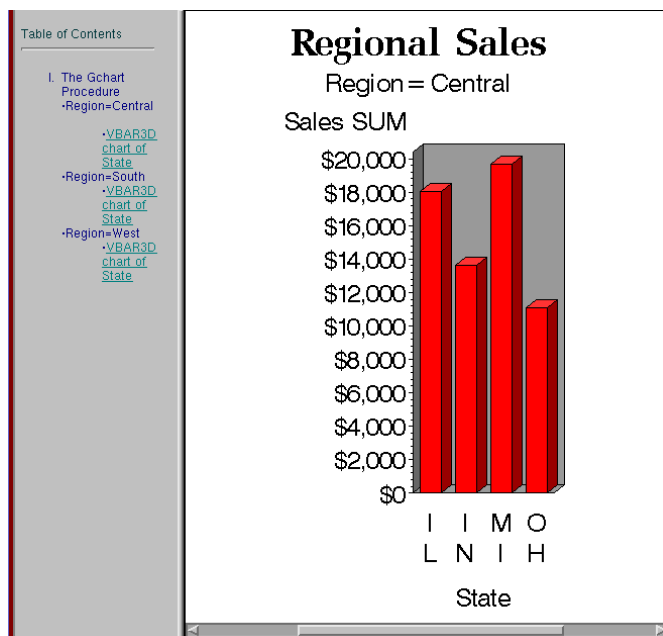
To open a new page file, specify another ODS statement and use the `PAGE=` option to specify the new file name.

To use the Table of Pages, view the page file in the browser. When you select a link from the Table of Pages, the browser goes to the target output referenced by that link. To use the Table of Pages again, you must use the browser's [Back](#) button or some other mechanism to return to the page file. If your browser supports HTML frames, you can keep the Table of Pages visible and its links accessible at all times by displaying the page file in a frame.

Using Frames to Display ODS Output

With ODS, you can create a frame file to display a Table of Contents, a Table of Pages, or both. A frame file is simply a file that uses two frames: one to reference a contents file, a page file, or both and a second to display output that is selected from the table of contents or pages. To use a frame file, your browser must support HTML frames. Display 16.5 on page 497 shows a frame file that is displaying a Table of Contents.

Display 16.5 Displaying a Frame File in a Browser

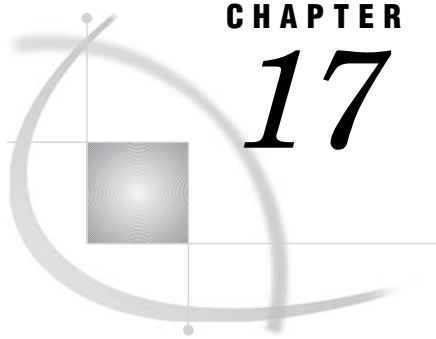


To create a frame file, specify a name for the file on the ODS statement option `FRAME=`. You must also use the options the `CONTENTS=` or `PAGE=` options, or both to provide a table to display in the left frame.

The following code creates a frame file named `saleFram.html`, which is created in the output location specified in the `FILENAME` statement.

```
filename odsout 'path-to-Web-server';
options device=gif;
ods html body='sales.html' path=odsout
      contents='saleCon.html'
      frame='saleFram.html';
```

To use the frame file, view the frame file in the browser. When you select a link from the Table of Contents or Table of Pages, the content of the right frame changes to display the output that is the target of the selected link, but the links from the contents or pages remain accessible in the left frame.



CHAPTER

17

Generating Web Output with the Annotate Facility

<i>Overview of Generating Web Output with the Annotate Facility</i>	499
<i>Generating Web Output with the Annotate Facility</i>	499
<i>When to Use PROC GANNO to Generate Web Output</i>	500
<i>When to Apply Annotate Data Sets to Web Output</i>	500
<i>Generating Web Links with the Annotate Facility</i>	500
<i>Examples</i>	501

Overview of Generating Web Output with the Annotate Facility

You can use the Annotate facility to enhance your Web presentation, or you can generate an entire Web presentation using Annotate and the GANNO procedure. In either case you can use the Annotate facility to generate drill-down presentations with the GIF, JPEG, or PNG device driver.

Note that your graph may conceal your annotations unless your annotations are specified with the option `WHEN=AFTER`. Specifying this option causes the annotations to be displayed after the graph, so that they will not be occluded. This is particularly important for interactive presentations, where the back wall of the graph may be made visible by default.

Note also that annotations disappear when the Web user selects another graph type. The annotations reappear when the Web user selects the Refresh button in the Web browser.

To learn how to use Annotate data sets to generate drill-down Web presentations, see “Generating Web Output with the Annotate Facility” on page 499.

Reference information on generating and applying Annotate data sets is provided in Chapter 25, “Annotate Dictionary,” on page 613. Usage information is provided in Chapter 24, “Using Annotate Data Sets,” on page 587. For information on the GANNO procedure, see Chapter 26, “The GANNO Procedure,” on page 707.

Generating Web Output with the Annotate Facility

You can use the Annotate facility to generate drill-down Web presentations in two ways: you can use PROC GANNO and an Annotate data set as the sole basis of a drill-down presentation, or you can apply an Annotate data set to add drill-down functionality to a Web presentation that is generated with the GIF, JPEG, or PNG device driver.

When to Use PROC GANNO to Generate Web Output

You can use ODS, the GANNO procedure, an Annotate data set, and a device driver to generate a Web presentation with drill-down links. This method of generating a drill-down presentation is preferred if you do not need to use an image from another SAS/GRAPH procedure in your Web presentation. For example, you could use PROC GANNO to generate an HTML output file that showed a JPEG image, with accompanying text, and a selectable label containing the text “Click Here”. Larger presentations with multiple drill-down links are also entirely feasible.

To generate a drill-down graph with PROC GANNO, see “Generating Web Links with the Annotate Facility” on page 500.

When to Apply Annotate Data Sets to Web Output

You can use Annotate data sets to add drill-down links to Web presentations generated by any procedure that uses the ANNOTATE= option. The Web presentation must be generated with the GIF, JPEG, or PNG device driver.

Using an Annotate data set to add drill-down links is preferable in the following circumstances:

- When you cannot add drill-down functionality by other means. Some SAS/GRAPH statements do not support the HTML= option, which SAS/GRAPH needs to generate an image map in the HTML output file. If the procedure does support the ANNOTATE= option, then you can use that procedure as the basis of a drill-down Web presentation.
- When you do not want Web users to drill down by selecting graph elements. For example, if you did not want your Web users to drill down by selecting the bars in a bar chart, you could define graphics elements with drill-down links using the Annotate facility.

To use the Annotate facility to add drill-down links to a Web presentation, see “Generating Web Links with the Annotate Facility” on page 500.

Generating Web Links with the Annotate Facility

Follow these steps if you are adding drill-down links to a Web presentation or if you are generating an entire Web presentations with PROC GANNO:

- 1 Plan your Web presentation so that you know how and where you want to apply Annotate graphical elements with drill-down links. Also determine your drill-down URLs.
- 2 Generate an Annotate data set. Elements that can be defined as drill-down hot zones are generated by Annotate functions that use the HTML variable. To see which functions use the HTML variable, refer to Figure 24.4 on page 593. To generate the Annotate data set, see Chapter 24, “Using Annotate Data Sets,” on page 587.
- 3 Specify the GIF, JPEG, or PNG device driver using the DEVICE= option in a GOPTIONS statement.
- 4 Close the listing destination and open an HTML output file in ODS.

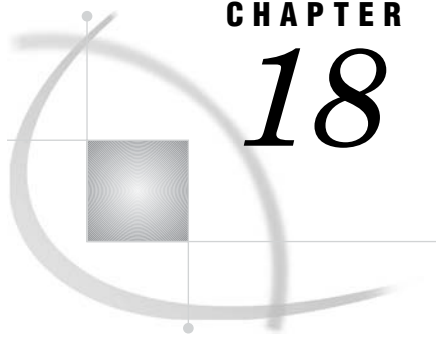
```
ods listing close;
ods html file="annodril.htm"
style=science;
```

- 5 Generate a GIF, JPEG, or PNG image and identify the Annotate data set. Use the GANNO procedure or another SAS/GRAPH procedure that uses the ANNOTATE= option.
- 6 Close the HTML output file.
- 7 Generate any additional HTML files or images as needed to provide files that are named in drill-down URLs.

Examples

For an example of creating web output with the GANNO procedure, see Example 4 on page 719.

For examples of applying Annotate data sets to web output, see “Examples” on page 604.



CHAPTER

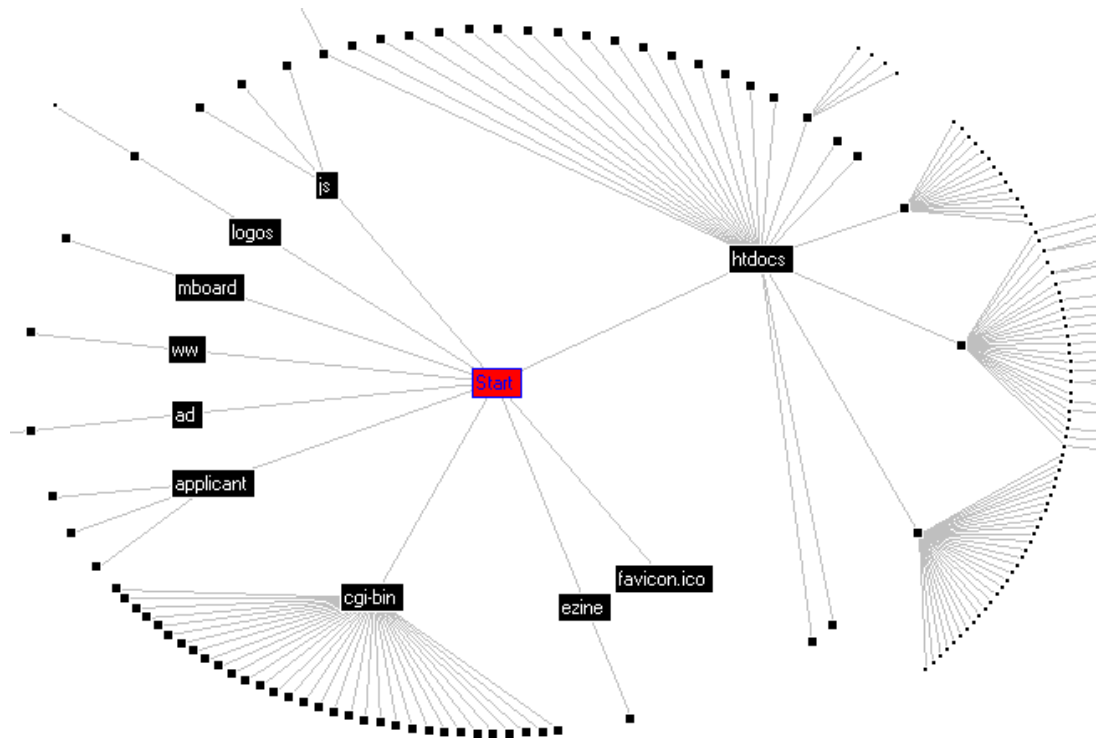
18

Creating Interactive Treeview Diagrams

<i>Creating Treeview Diagrams</i>	503
<i>When to Use the Treeview Applet</i>	504
<i>Interactivity Enabled by the Treeview Applet</i>	505
<i>Programming with the DS2TREE Macro for the Treeview Applet</i>	505
<i>Enhancing Presentations for the Treeview Applet</i>	506
<i>DS2TREE Macro Arguments</i>	507
<i>Sample Programs: Treeview Macro</i>	507
<i>Sample Treeview with XML Embedded in the HTML File</i>	507
<i>Results Shown in a Browser</i>	508
<i>SAS Code</i>	508
<i>Sample Treeview with XML Written to an External File</i>	509
<i>SAS Code</i>	509
<i>Treeview with Hotspots</i>	510
<i>SAS Code</i>	510

Creating Treeview Diagrams

The Treeview applet generates node/link diagrams for hierarchical data, with optional fish-eye distortion that highlights the central area of interest, as shown in the following figure:

Display 18.1 A Treeview Applet Web Link Diagram

You can scroll across the diagram by selecting off-center nodes or by searching for nodes. Positioning the cursor over a node can display optional data tips. If you then right-click, you access a pop-up menu. The menu enables you to highlight or hide subtrees or drill-down to an optional URL. The menu also enables you to select all nodes, display all previously hidden nodes, reset the view, display applet help, and search for nodes using various search parameters.

SAS/GRAPH programming for the Treeview applet differs from some of the other applets in that it does not use ODS, a device driver specification, or a SAS/GRAPH procedure. Instead, the DS2TREE macro references data sets to generate and configure an HTML output file that runs the Treeview applet.

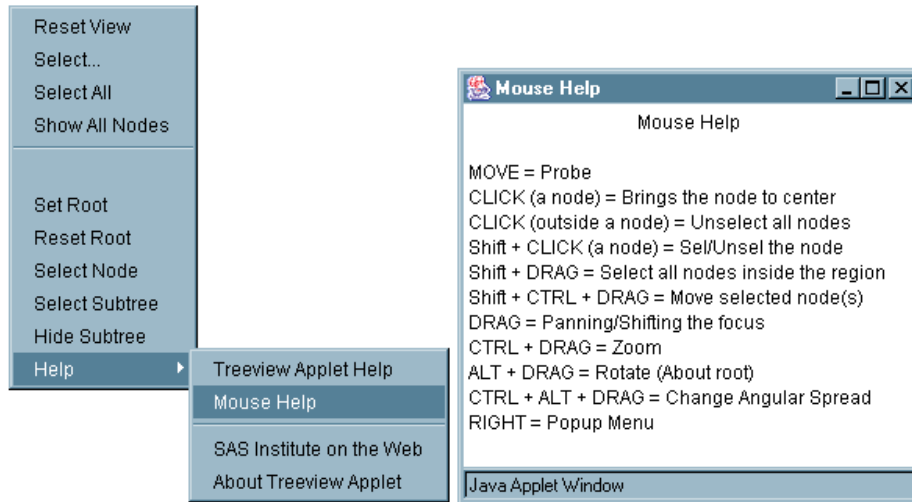
When to Use the Treeview Applet

The Treeview applet is well-suited for the illustration of hierarchical data sets. The fish-eye distortion factor, coupled with extensive node selection features, means that a single node/link diagram can accommodate large data sets. Applet parameters can be set to specify the layout of the diagram. You specify a starting node, and then you specify how the other nodes are to be drawn in relation to that node. The resulting diagram can be as complex as the Web link diagram in Display 18.1 on page 504, or as simple as an organizational tree for a department in a corporation.

If you need a higher degree of configurability to illustrate weighted relationships between the nodes and links in your diagram, then the Constellation applet might be a better choice than the Treeview applet, as described in “Creating Constellation Diagrams” on page 513.

Interactivity Enabled by the Treeview Applet

The following picture shows the pop-up menu that a user can invoke by right-clicking a Treeview diagram in a browser. The picture shows all the options that are available for interacting with the diagram. For a description of these options, right-click on any Treeview diagram and select **Treeview Applet Help** from the pop-up menu.



Programming with the DS2TREE Macro for the Treeview Applet

The DS2TREE macro generates HTML output files for the Treeview applet. Macro arguments enable you to generate and format an HTML file and to customize the appearance of your node/link diagram.

Follow the steps shown in the following code to generate a Web presentation that runs the Treeview applet.

```

/* 1. Define the name and storage location of the HTML output file */
/*    and the location of the jar files.                                */
%let htmlfile = your_path_and_filename.htm;
%let jarfiles = http://your_path_to_archive;

/* 2. Define a data set that contains parent-child relationships. */
data myorg;
input name $ empno mgrno deptname $22. deptcode $;
cards;
Peter      2620   1420   Documentation          DOC
Linda     6915   1420   Research & Development  R&D
Maria     1320   1420   Legal                  LGL
Vince     1420   1750   Executive               EXE
Jim        6710   6915   Quality Assurance      QA
Nancy     22560  6915   Quality Assurance      QA
Patrick   28470  6915   Quality Assurance      QA
Elsa      33075  6915   Development             DEV
Clement  22010  6915   Development             DEV
Murielle  3020   6915   Development             DEV
    
```

```

David      11610   6915   Research      RES
;
run;

/* 3. Specify titles and footnotes: (optional). */
title1 'Organizational Chart';
footnote1 'To display the department name, place the cursor over a node.';
footnote2 'To rotate the chart, click and drag a node.';

/* 4. Run the DS2TREE macro. */
/* You must change the CODEBASE= argument (using either http://      */
/* or a directory path such as C:/) to specify the location of your  */
/* sas.graph.treeview.jar file and its associated jar files          */
/* (sas.graph.nld.jar, sas.graph.j2d.jar). See the CODEBASE= argument in: */
/* 'Arguments for the APPLET Tag' on page 536.*/
/* (Make sure that ods listing is open when running macro.) */
ods listing;
%ds2tree(ndata=myorg,          /* data sets and files */
         codebase=&jarfiles,
         xmltype=inline,
         htmlfile=&htmlfile,
         nid=empno,          /* roles of variables */
         nparent=mgrno,
         ntip=deptname,
         nlabel=name,
         height=500,        /* appearance */
         width=600,
         tcolor=navy,
         fcolor=black);

```

Display the resulting HTML file in a Web browser to run the Treeview applet and display the node/link diagram.

The preceding example shows how the arguments of the DS2TREE macro identify a data set and specify how the variables in that data set are to be interpreted to generate the diagram. Appearance arguments define the size of the diagram and the color of the text in the title and footnotes.

For information on generating more complex diagrams for the Treeview applet, see “Enhancing Presentations for the Treeview Applet” on page 506.

For definitions of all DS2TREE macro arguments, see “DS2TREE Macro Arguments” on page 507.

Enhancing Presentations for the Treeview Applet

The Treeview applet displays interactive node/link diagrams. The diagrams are generated in SAS using a hierarchical data set and the DS2TREE macro, as described in “Programming with the DS2TREE Macro for the Treeview Applet” on page 505.

To enhance Treeview applet presentations, specify additional arguments for the DS2TREE macro. The following table describes some of the available enhancements and identifies the DS2TREE arguments that implement them. For a complete list of macro arguments, see “Macro Arguments” on page 535.

Table 18.1 Treeview Applet Enhancements

Enhancement	DS2TREE Argument
Specify a stylesheet to format your HTML output file.	SSFILE, SSFREF, SSHREF, SSMEDIA, SSREL, SSREV, SSTITLE, SSTYPE
Specify dash patterns for link lines.	LSTIP, LSTIPFAC
Specify a background color, image, or drill-down URL.	IBACKPOS, IBACKLOC, IBACKURL
Add pop-up data tips to nodes.	NTIP, TIPS
Add drill-down URLs to nodes.	NURL
Specify an action for the pull-down menu.	ACTION, NACTION
Change the amount of fisheye distortion.	FACTOR, FISHEYE
Specify a JavaScript response to node selection.	SELIFUNC, SELLFUNC, SELUFUNC
Determine layout of diagram.	SPREAD, TREEDIR, TREESPAN

DS2TREE Macro Arguments

The arguments of the DS2TREE macro specify the configuration of the HTML output file, the location of the data that is used to generate the diagram, and the configuration of the applet's interactive features.

The DS2TREE macro uses the following syntax:

```
%DS2TREE(argument1=value1, argument2=value2, ...);
```

The arguments of the DS2TREE macro can be divided into the following categories:

- “Arguments for the APPLET Tag” on page 536. The CODEBASE argument is required.
- “DS2TREE and DS2CONST Arguments for Data Definition” on page 537. For DS2TREE the arguments NDATA and NID are required.
- “Arguments for Generating HTML and XML Files” on page 544.
- “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545.
- “Arguments for Page Formatting” on page 552.
- “Arguments for Stylesheets” on page 554.
- “Arguments for the SAS TITLE and FOOTNOTE Tags” on page 556.
- “Arguments for Character Transcoding” on page 561.

Sample Programs: Treeview Macro

The following sample programs generate Treeview diagrams:

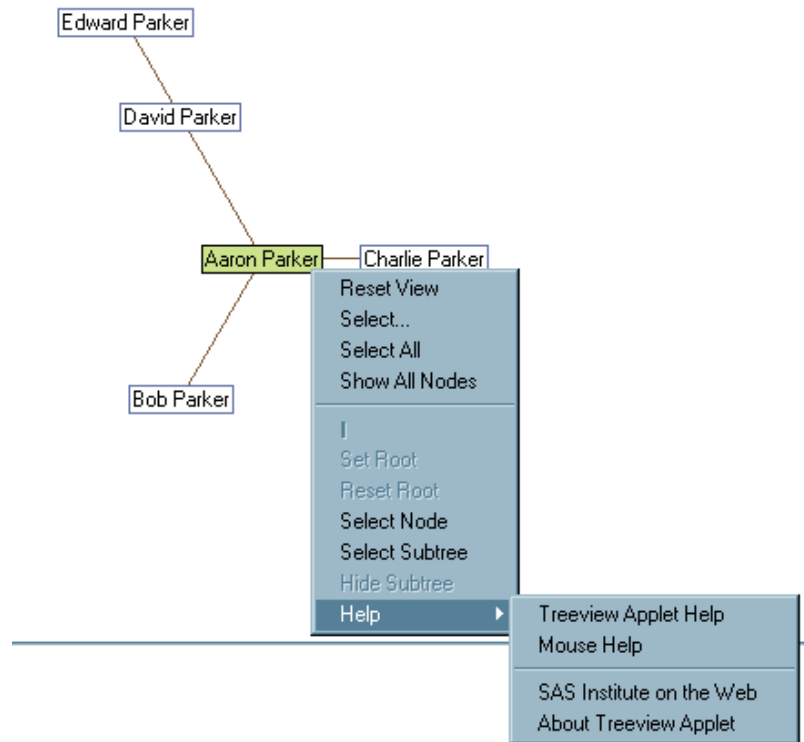
- “Sample Treeview with XML Embedded in the HTML File” on page 507
- “Sample Treeview with XML Written to an External File” on page 509
- “Treeview with Hotspots” on page 510.

Sample Treeview with XML Embedded in the HTML File

This sample program generates a very simple Treeview diagram.

Results Shown in a Browser

The following is the Treeview diagram that is generated by the sample code. Notice the pop-up menu. Because the diagram is displayed by the Treeview applet, it is not just a static picture. A user can manipulate the diagram, for example, by bringing selected nodes to the center, spreading out the nodes, and searching for nodes.



SAS Code

The following is the complete SAS code used to generate the Treeview diagram from a SAS data set. Note the following:

- The parameter `HTMLFILE=` specifies the complete path and name of the HTML file to be created by the `DS2TREE` macro. If you want to run this sample, then change the values of `HTMLFILE` and `CODEBASE` to the locations that you want to use.
- The parameter `XMLTYPE=INLINE` tells the `DS2TREE` macro that the XML it generates from the SAS data set should be included inline in the HTML file.
- The parameter `CUTOFF=1` specifies that every node on the graph be labeled. Use this parameter to suppress node labels for diagrams with numerous nodes.

```
data father_and_sons;
input id $8. name $15. father $8.;
cards;
aaron  Aaron Parker
bob    Bob Parker   aaron
charlie Charlie Parker aaron
david  David Parker aaron
edward Edward Parker david
```

```

;
run;

/* make sure ods listing is open when running macro */
ods listing;
/* run the macro */
%ds2tree(ndata=father_and_sons, /* data set */
/* specify complete url if jar files are not in same directory as html file */
codebase=http://your_path_to_archive,
xmltype=inline,
htmlfile=your_path_and_filename.htm,
nid=id, /* use this variable as the id */
cutoff=1, /* display the name on every node */
nparent=father, /* this identifies the parent of each node */
nlabel=name, /* display this on each node */
height=400,
width=400,
tcolor=navy,
fcolor=black);

```

Sample Treeview with XML Written to an External File

This sample program generates the same Treeview as the previous example, “Sample Treeview with XML Embedded in the HTML File” on page 507, with the difference that the XML is written to an external file instead of being embedded in the HTML file.

SAS Code

The following is the complete SAS code to generate the Treeview diagram from a SAS data set. Note the following:

- The parameter HTMLFILE= specifies the complete path and name of the HTML file to be created by the DS2TREE macro. If you want to run this sample, then change the value of HTMLFILE to something that makes sense for you.
- The parameter XMLTYPE=EXTERNAL tells the DS2TREE that the XML it generates from the SAS data set should be written to an external file.
- The parameter XMLFILE= specifies the path and file name of the XML file to be created.
- The parameter XMLURL= specifies how the XML file is to be addressed from within the HTML file.
- The parameter CUTOFF=1 specifies that every node on the graph be labeled. Use this parameter with a value between 0 and 1 to suppress node labels for diagrams with numerous nodes.

```

data father_and_sons;
input id $8. name $15. father $8.;
cards;
aaron   Aaron Parker
bob     Bob Parker   aaron
charlie Charlie Parker aaron
david   David Parker   aaron
edward  Edward Parker   david
;

```

```

run;
goptions reset=all;
/* make sure ods listing is open when running macro */
ods listing;
/* run the macro */
%ds2tree(ndata=father_and_sons, /* data set */
         codebase=http://your_path_to_archive,
         htmlfile=your_path_and_filename.htm,
         xmltype=external,
         makexml=y,
         xmlurl=http://www.xtz.com/weboutput_treeview2_sample.xml,
         xmlfile=u:/public/weboutput_treeview2_sample.xml,
         nid=id,          /* as the id, use this variable specified here */
         cutoff=1,       /* display the name on every node */
         nparent=father, /* this identifies the parent of each node */
         nlabel=name,    /* display the value of this variable on each node */
         height=400,
         width=400,
         tcolor=navy,
         fcolor=black);

```

Treeview with Hotspots

This sample program generates the same Treeview as the previous example, “Sample Treeview with XML Embedded in the HTML File” on page 507, with the difference that a node is associated with a URL and can be activated by a user double-clicking the node.

SAS Code

The following is the complete SAS code to generate the Treeview diagram from a SAS data set. Note the following:

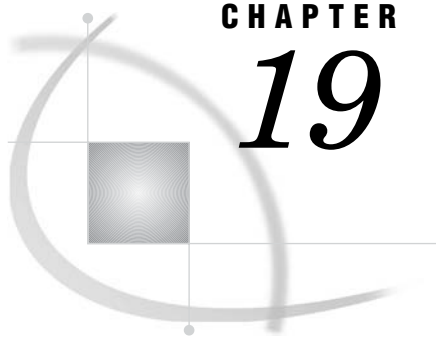
- The parameter `NURL=` specifies that the URL to be opened when the corresponding node is double-clicked.
- The parameter `DRILTARG=_TOP` specifies that the HTML file is to be opened in the same window as the Treeview diagram instead of in a new window, as is the default.

```

data father_and_sons;
input id $8. name $15. father $8. url $30.;
cards;
aaron  Aaron Parker          http://www.xyz.com/index.html
bob    Bob Parker      aaron  http://www.xyz.com/index.html
charlie Charlie Parker aaron  http://www.xyz.com/index.html
david  David Parker    aaron  http://www.xyz.com/index.html
edward Edward Parker  david  http://www.xyz.com/index.html
;
run;
/* make sure ods listing is open when running macro */
ods listing;
/* run the macro */
%ds2tree(ndata=father_and_sons, /* data set */
         /* specify complete url if jar files are not in same directory as html file */
         codebase=http://your_path_to_archive,
         xmltype=inline,

```

```
htmlfile=your_path_and_filename.htm,  
nid=id,          /* as the id, use the variable specified here */  
cutoff=1,       /* display the name on every node */  
nparent=father,/* this identifies the parent of each node */  
nlabel=name,    /* display the value of this variable on each node */  
height=400,  
width=400,  
tcolor=navy,  
fcolor=black,  
nurl=url,  
driltarg=_top );
```

CHAPTER

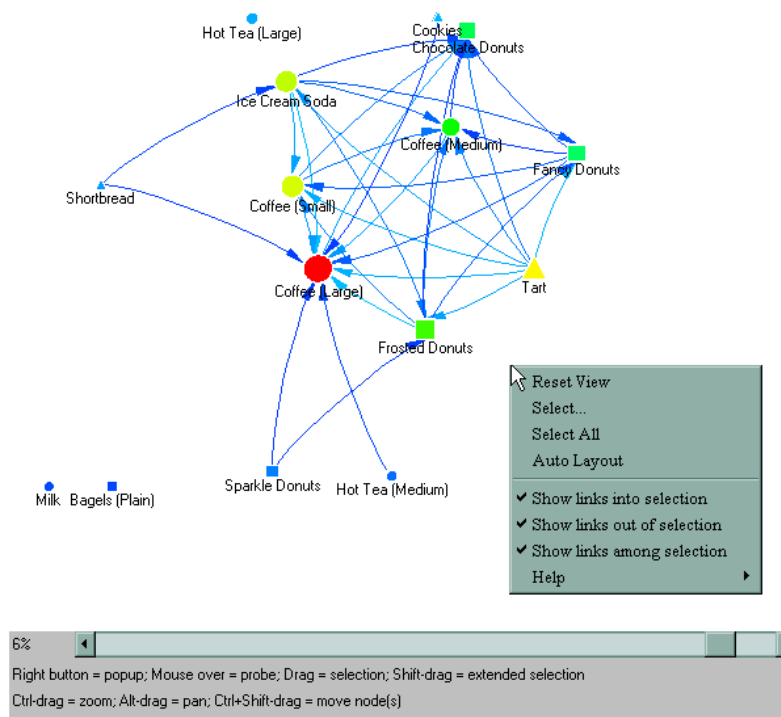
19

Creating Interactive Constellation Diagrams

<i>Creating Constellation Diagrams</i>	513
<i>When to Use the Constellation Applet</i>	514
<i>Programming with the DS2CONST Macro for the Constellation Applet</i>	515
<i>Enhancing Presentations for the Constellation Applet</i>	517
<i>DS2CONST Macro Arguments</i>	518
<i>Sample Programs: Constellation Macro</i>	518
<i>Constellation Chart with DATATYPE=ARCS</i>	518
<i>Results Shown in a Browser</i>	519
<i>SAS Code</i>	519
<i>Constellation Chart with DATATYPE=ASSOC</i>	520
<i>Results Shown in a Browser</i>	520
<i>SAS Code</i>	521
<i>Constellation Chart with XML Written to an External File</i>	522
<i>SAS Code</i>	523
<i>Constellation Chart with Hotspots</i>	524
<i>SAS Code</i>	524

Creating Constellation Diagrams

The Constellation Applet provides interactivity for node/link diagrams that illustrate data that is associative, hierarchical, or requires an arc list. Node and link color and size can be associated with specified data values.

Display 19.1 A Constellation Applet Affinity Diagram

Interactive features of the Constellation Applet include pop-up data tips for links and nodes, subsetting of links via an embedded scroll bar, pan and zoom, fish-eye distortion of a central image area, and several node and link selection modes. You can define drill-down URLs for nodes, specify menu text for the drill-down action, insert a background image, and specify a drill-down URL for the background image, among other enhancements. You can also specify your own JavaScript methods to define responses to drill-down actions.

The Constellation Applet, like the Treeview and Rangeview applets, differ from the other applets in that the diagrams that they display are not generated by SAS/GRAPH procedures. The DS2CONST macro generates and formats an HTML output file, and specifies the appearance and behavior of the node/link diagram based on values in a data set.

When to Use the Constellation Applet

The Constellation Applet is best used to illustrate relationships between links and nodes, which can be shown in affinity, sequence, and Web-click path diagrams, for example. Colors, link line widths, and link directional indicators can be specified to illustrate relationships. Pop-up data tips can be specified for nodes and links, along with drill-down URLs for nodes and for an optional background image. For diagrams that illustrate associative data, an embedded scroll bar subsets the data in the diagram dynamically.

The Constellation Applet can be used to display hierarchical data, but so can the Treeview Applet, which should also be considered for hierarchical diagrams such as organizational trees, because of its unique layout capabilities. For information on the Treeview Applet, see “Creating Treeview Diagrams” on page 503.

Programming with the DS2CONST Macro for the Constellation Applet

The DS2CONST macro enables you to generate complete Web presentations for the Constellation Applet. The macro has a large number of arguments that you can use to generate and format an HTML output file, configure the diagram, and describe how data sets and variables are to be applied to the diagram.

The macro arguments are structured so that you can associate a variable with an aspect of the diagram. The values of the variable are then used for that part of the diagram. For example, the NLABEL argument specifies the name of the variable whose values define the text labels that are to be applied to the nodes. Other arguments provide default values that are used when no variable value is provided.

Descriptions of all of the arguments of the DS2CONST macro are provided in “DS2CONST Macro Arguments” on page 518.

Follow the steps outlined in the following code to use the DS2CONST macro to generate a Web presentation for the Constellation Applet.

```

/* 1. Define the name and storage location of the HTML output file, */
/*    and the location of the jar files.                               */
%let htmlfile = your_path_and_filename.htm;
%let jarfiles = http://your_path_to_archive;

/* 2. Define a nodes data set */
data nodedata;
length nodeLabel $8 nodeId xLoc yLoc 8;
input nodeLabel nodeID xLoc yLoc;
cards;
Snacks      5  375  25
SftDrink    3   25  25
Books       2  200  25
BeerWine    4  200  300
Dairy       1   25  300
Bakery      0  375  300
;
run;

/* 3. Define a links data set */
data linkdata;
length from to value 8 tip $45;
input from to value @13 tip $char45.;
cards;
0 0 0      Bakery => Bakery #0
0 1 0.7    Bakery => Dairy #0.7
0 2 0.1    Bakery => Books #0.1
0 3 0.3    Bakery => SftDrink #0.3
0 4 0.1    Bakery => BeerWine #0.1
0 5 0.2    Bakery => Snacks #0.2
1 0 0.5    Dairy => Bakery #0.5
1 1 0      Dairy => Dairy #0
1 2 0.1    Dairy => Books #0.1
1 3 0.2    Dairy => SftDrink #0.2
1 4 0.3    Dairy => BeerWine #0.3
1 5 0.4    Dairy => Snacks #0.4
2 0 0.1    Books => Bakery #0.1
2 1 0.2    Books => Dairy #0.2

```

```

2 2 0    Books => Books #0
2 3 0.3  Books => SftDrink #0.3
2 4 0.3  Books => BeerWine #0.3
2 5 0.3  Books => Snacks #0.3
3 0 0.2  SftDrink => Bakery #0.2
3 1 0.3  SftDrink => Dairy #0.3
3 2 0.2  SftDrink => Books #0.2
3 3 0    SftDrink => SftDrink #0
3 4 0.6  SftDrink => BeerWine #0.6
3 5 0.8  SftDrink => Snacks #0.8
4 0 0.2  BeerWine => Bakery #0.2
4 1 0.4  BeerWine => Dairy #0.4
4 2 0.1  BeerWine => Books #0.1
4 3 0.5  BeerWine => SftDrink #0.5
4 4 0    BeerWine => BeerWine #0
4 5 0.9  BeerWine => Snacks #0.9
5 0 0.3  Snacks => Bakery #0.3
5 1 0.4  Snacks => Dairy #0.4
5 2 0.1  Snacks => Books #0.1
5 3 0.5  Snacks => SftDrink #0.5
5 4 0.5  Snacks => BeerWine #0.5
5 5 0    Snacks => Snacks #0
run;

/* 4. Define a title */
title1 'Grocery Affinity Diagram.';

/* make sure ods listing is open when running macro */
ods listing;
/* 5. Run the DS2CONST macro */
%ds2const(ndata=nodedata,
          ldata=linkdata,
          datatype=arcs,
          cnode=red,
          colormap=y,
          height=650,
          width=600,
          codebase=&jarfiles,
          htmlfile=&htmlfile,
          nid=nodeID,
          nlabel=nodelabel,
          nx=xLoc,
          ny=yLoc,
          lfrom=from,
          lto=to,
          lvalue=value,
          ltip=tip);

```

Note: You must change the CODEBASE= argument (using either *http://* or a directory path such as *C:/*) to specify the location of your *sas.graph.constapp.jar* file and its associated jar files (*sas.graph.nld.jar*, *sas.graph.j2d.jar*). See the CODEBASE= argument in “Arguments for the APPLETTAG” on page 536. \triangle

Display the resulting HTML file in a Web browser to run the applet and generate the diagram.

Arguments in the DS2CONST macro identify the name of the nodes and links data sets. In the nodes data set, arguments identify a node ID variable and a node label variable. Other arguments identify the links data set and the variables that define the nodes at the start and end of each link line.

For information on more complex presentations for the Constellation Applet, see “Enhancing Presentations for the Constellation Applet” on page 517.

Enhancing Presentations for the Constellation Applet

The Constellation Applet displays interactive node/link diagrams. These diagrams can show relationships between nodes and links. The Constellation Applet displays affinity, sequence, and ring diagrams that are generated out of arc, associative, or hierarchical data sets. The Constellation Applet provides a number of interactive features by default, as described in “Creating Constellation Diagrams” on page 513.

Enhancements to Constellation Applet presentations are configured in your SAS/GRAPH program by specifying arguments in the DS2CONST macro. The following table lists some of the available enhancements and the DS2CONST arguments that implement them. These enhancements enable you to provide data tips and drill-down URLs for nodes and links, and to increase the visible distinctions between the data values that are associated with the nodes and links.

Table 19.1 Constellation Applet Enhancements

Enhancements	DS2CONST Arguments
Specify link weights and configure a scroll bar that controls the display of links based on weight.	LVALUE, MINLNKWT, SCLNKWT
Configure a fish-eye lens for large diagrams.	FACTOR, FISHEYE
Lay out the diagram automatically or as specified in a data set.	LAYOUT
Specify a stylesheet to format the HTML output file.	BDCLASS, SEPCLASS, SPCLASS, SSFILE, SSHREF See “Arguments for Stylesheets” on page 554.
Add pop-up data tips to nodes and links.	LTIP, NTIP
Define responses to node selection by defining JavaScript methods	SELIFUNC, SELNFUND, SELUFUNC
Define drill-down URLs for nodes and links.	LURL, NURL
Specify menu option text for a drill-down action.	ACTION, NACTION
Specify a browser window or frame that displays drill-down URLs.	DRILTARG
Add a background color, image, or drill-down URL.	IBACKLOC, IBACKPOS, IBACKURL,
Specify text colors, fonts, styles, and sizes.	NFNTNAME, NSFNTNAM, CTEXT, CATEXT See “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545.

Enhancements	DS2CONST Arguments
Specify colors for nodes and links.	NCOLVAL, NCOLOR, CNODE, LCOLVAL, LCOLOR, CLINK
Specify dashed link lines.	LSTIP and LSTIPFAC

Note that a number of enhancements apply only to associative data sets when you specify the macro argument `DATATYPE=ASSOC`. The macro argument definitions identify which features apply only to associative data.

The DS2CONST macro requires you to specify node and link data sets. As an enhancement, you can define a node styles data set that contains style information only. You can use the node styles data set to standardize the appearance of a series of diagrams, among other uses.

Reference information on the arguments of the DS2CONST macro is provided in “DS2CONST Macro Arguments” on page 518.

DS2CONST Macro Arguments

The arguments of the DS2CONST macro specify the configuration of the HTML output file, the location of the data that is used to generate the diagram, and the configuration of the applet’s interactive features.

The DS2CONST macro uses the following syntax:

```
%DS2CONST(argument1=value1, argument2=value2, ...);
```

The arguments of the DS2CONST macro can be divided into the following categories:

- “Arguments for the APPLET Tag” on page 536. The CODEBASE argument is required.
- “DS2TREE and DS2CONST Arguments for Data Definition” on page 537. For DS2CONST the arguments NDATA, NID, LDATA, and LTO are required.
- “Arguments for Generating HTML and XML Files” on page 544.
- “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545.
- “Arguments for Page Formatting” on page 552.
- “Arguments for Stylesheets” on page 554.
- “Arguments for the SAS TITLE and FOOTNOTE Tags” on page 556.
- “Arguments for Character Transcoding” on page 561.

Sample Programs: Constellation Macro

The following sample programs generate these kinds of Constellation diagrams:

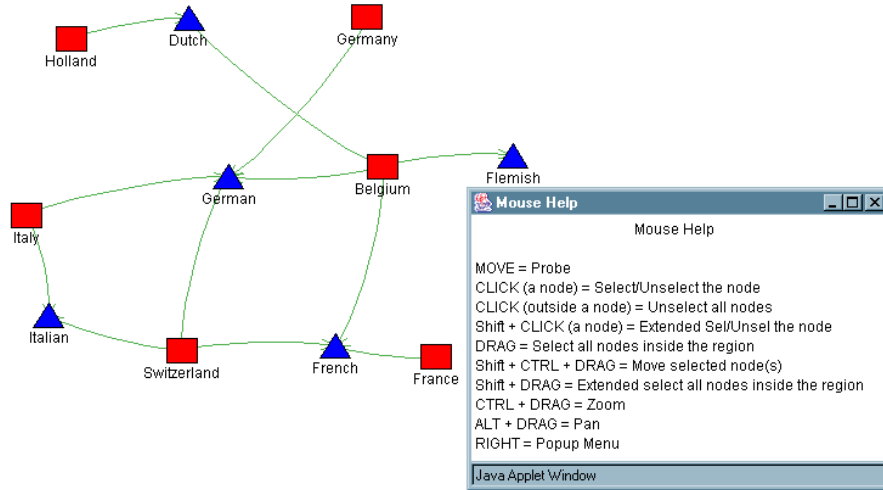
- “Constellation Chart with `DATATYPE=ARCS`” on page 518
- “Constellation Chart with `DATATYPE=ASSOC`” on page 520
- “Constellation Chart with XML Written to an External File” on page 522
- “Constellation Chart with Hotspots” on page 524.

Constellation Chart with `DATATYPE=ARCS`

This sample program generates a very simple Constellation diagram. It displays a number of countries and the languages spoken in those countries.

Results Shown in a Browser

The following is the Constellation diagram that is generated by the sample code shown below. Notice the help window. Because the diagram is displayed by the Constellation applet, it is not just a static picture. A user can manipulate the diagram, for example, by moving nodes and searching for nodes. The Mouse Help window in the following diagram documents for the user what interactivity is available (right-click a diagram to invoke the window).



SAS Code

The following is the complete SAS code used to generate a Constellation diagram from a SAS data set. Notice the following:

- The parameter HTMLFILE= specifies the complete path and name of the HTML file to be created by the DS2CONST macro. If you want to run this sample, then change the value of HTMLFILE to the location where you want the HTML file stored.
- The parameter NSHAPE= specifies the variable in the SAS data set that encodes the shape of each node.
- The parameter NCOLOR= specifies the variable in the SAS data set that encodes the color of each node.

```

/*Define a nodes data set of countries and languages */
data nodedata;
input nodeLabel $15. shape $10. color $8. size;
cards;
France          square    red      .1
Germany        square    red      .1
Italy           square    red      .1
Belgium        square    red      .1
Switzerland    square    red      .1
Holland        square    red      .1
German         triangle  blue    .1
French         triangle  blue    .1
Italian        triangle  blue    .1
Flemish        triangle  blue    .1
    
```

```

Dutch          triangle blue    .1
;
run;

/*Define a links data set */
data linkdata;
input from $15. to $15.;
cards;
France         French
Germany        German
Belgium        French
Belgium        German
Belgium        Flemish
Belgium        Dutch
Switzerland   French
Switzerland   German
Switzerland   Italian
Italy          Italian
Italy          German
Holland        Dutch
;
run;
goptions reset=all;
/* make sure ods listing is open when running macro */
ods listing;
/*Run the DS2CONST macro*/
%ds2const(ndata=nodedata,
          ldata=linkdata,
          datatype=arcs,
          cnode=red,
          colormap=y,
          height=400,
          width=500,
          code=ConstChart,
          codebase=http://your_path_to_archive,
          htmlfile=your_path_and_filename.htm,
          nid=nodelabel,
          nlabel=nodelabel,
          lfrom=from,
          lto=to,
          fntsize=12,
          nshape=shape,
          ncolor=color,
          nsize=size);

```

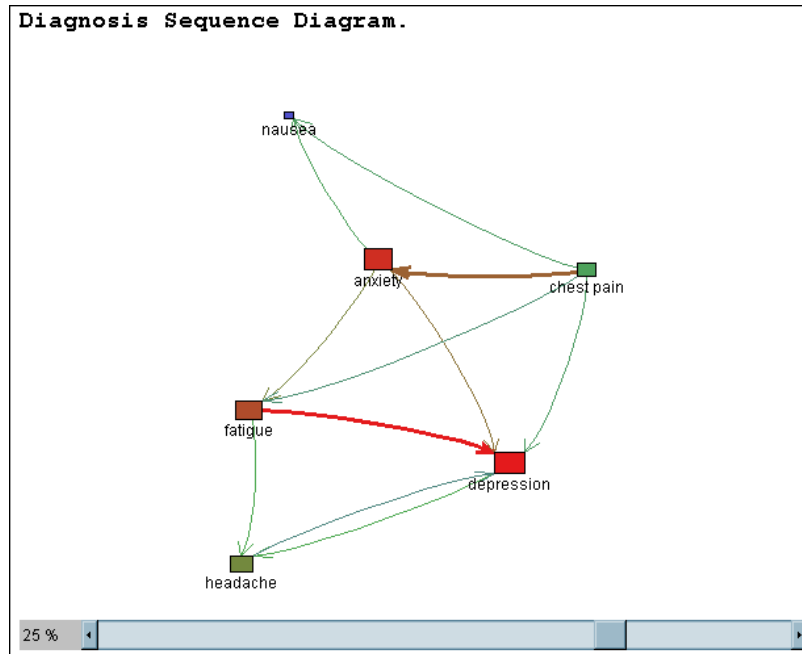
Constellation Chart with DATATYPE=ASSOC

This sample program generates a very simple Constellation diagram with DATATYPE=ASSOC.

Results Shown in a Browser

The following is the Constellation diagram that is generated by the sample code. A Constellation diagram with DATATYPE=ASSOC depicts the strength of the

relationships among variables. Variables in the SAS data set determine the size and color of nodes, as well as the width and color of the lines between nodes. At the bottom of the picture, notice the slider bar which allows a user to choose how many of the links on the diagram are displayed. Move the slider to the left, and only the most important links are displayed. Move the slider to the right, and all of the links are displayed.



SAS Code

The following is the complete SAS code to generate a Constellation diagram from a SAS data set. Notice the following:

- The parameter HTMLFILE= specifies the complete path and name of the HTML file to be created by DS2CONST. If you want to run this sample, then change the value of HTMLFILE to something that makes sense for you.
- The parameter NVALUE= specifies the data set variable that is used to determine the size and color of each node.
- The parameter LVALUE= specifies the data set variable that is used to determine the width and color of each line between nodes.

```
data nodedata;
length nodeID value 8 label $11 tip $25;
input nodeID value @11 label $char11. @25 tip $char25.;
cards;
0 6556 depression depression: #6556
1 6322 anxiety anxiety: #6322
2 5980 fatigue fatigue: #5980
3 5286 headache headache: #5286
4 4621 chest pain chest pain: #4621
6 3149 nausea nausea: #3149
;
run;

data linkdata;
```

```

length from to linkvalue 8 tip $40;
input from to linkvalue @13 tip $char40.;
cards;
2 0 5978 #5978, Support:63.0790, Conf:99.9833
4 1 4621 #4621, Support:48.7602, Conf:100.0000
1 0 4307 #4307, Support:45.4469, Conf:68.1272
1 2 3964 #3964, Support:41.8276, Conf:62.7017
2 3 3010 #3010, Support:31.7611, Conf:50.3429
0 3 3009 #3009, Support:31.7506, Conf:47.5957
1 6 2772 #2772, Support:29.2498, Conf:43.8469
4 6 2609 #2609, Support:27.5298, Conf:56.4596
4 0 2606 #2606, Support:27.4982, Conf:56.3947
4 2 2263 #2263, Support:23.8789, Conf:48.9721
3 0 1980 #1980, Support:20.8927, Conf:40.6821
3 1 1701 #1701, Support:17.9487, Conf:34.9497
3 2 1701 #1701, Support:17.9487, Conf:34.9497
1 3 1593 #1593, Support:16.8091, Conf:25.1977
4 3 1152 #1152, Support:12.1557, Conf:24.9297
0 6 623 #623, Support:6.5738, Conf:9.8545
2 6 623 #623, Support:6.5738, Conf:10.4198
6 3 597 #597, Support:6.2995, Conf:20.0268
3 6 372 #372, Support:3.9253, Conf:7.6433
6 0 344 #344, Support:3.6298, Conf:11.5398
run;

/* make sure ods listing is open when running macro */
ods listing;

title1 'Diagnosis Sequence Diagram.';
%ds2const(ndata=nodedata,
          ldata=linkdata,
          datatype=assoc,
          minlnkwt=30,
          height=450,
          width=600,
          codebase=http://your_path_to_archive,
          htmlfile=your_path_and_filename.htm,
          colormap=y,
          nid=nodeID,
          nlabel=label,
          nvalue=value,
          fntsize=12,
          ntip=tip,
          lfrom=from,
          lto=to,
          lvalue=linkvalue,
          ltip=tip,
          linktype=arrow);

```

Constellation Chart with XML Written to an External File

This sample program generates the same Constellation diagram as the previous example, “Constellation Chart with DATATYPE=ASSOC” on page 520, with the

difference that the XML is written to an external file instead of being embedded in the HTML file.

SAS Code

The following is the complete SAS code to generate the Constellation diagram from a SAS data set. You can notice the following:

- The parameter HTMLFILE= specifies the complete path and name of the HTML file to be created by DS2CONST. If you want to run this sample, then change the value of HTMLFILE to something that makes sense for you.
- The parameter XMLTYPE=EXTERNAL tells the DS2CONST macro that the XML that it generates from the SAS data set should be written to an external file.
- The parameter XMLFILE= specifies the path and file name of the XML file to be created.
- The parameter XMLURL= specifies how the XML file is to be addressed from within the HTML file.

```
data nodedata;
length nodeID value 8 label $11 tip $25;
input nodeID value @11 label $char11. @25 tip $char25.;
cards;
0 6556 depression depression: #6556
1 6322 anxiety anxiety: #6322
2 5980 fatigue fatigue: #5980
3 5286 headache headache: #5286
4 4621 chest pain chest pain: #4621
6 3149 nausea nausea: #3149
;
run;
```

```
data linkdata;
length from to linkvalue 8 tip $40;
input from to linkvalue @13 tip $char40.;
cards;
2 0 5978 #5978, Support:63.0790, Conf:99.9833
4 1 4621 #4621, Support:48.7602, Conf:100.0000
1 0 4307 #4307, Support:45.4469, Conf:68.1272
1 2 3964 #3964, Support:41.8276, Conf:62.7017
2 3 3010 #3010, Support:31.7611, Conf:50.3429
0 3 3009 #3009, Support:31.7506, Conf:47.5957
1 6 2772 #2772, Support:29.2498, Conf:43.8469
4 6 2609 #2609, Support:27.5298, Conf:56.4596
4 0 2606 #2606, Support:27.4982, Conf:56.3947
4 2 2263 #2263, Support:23.8789, Conf:48.9721
3 0 1980 #1980, Support:20.8927, Conf:40.6821
3 1 1701 #1701, Support:17.9487, Conf:34.9497
3 2 1701 #1701, Support:17.9487, Conf:34.9497
1 3 1593 #1593, Support:16.8091, Conf:25.1977
4 3 1152 #1152, Support:12.1557, Conf:24.9297
0 6 623 #623, Support:6.5738, Conf:9.8545
2 6 623 #623, Support:6.5738, Conf:10.4198
6 3 597 #597, Support:6.2995, Conf:20.0268
3 6 372 #372, Support:3.9253, Conf:7.6433
```

```

6 0 344 #344, Support:3.6298, Conf:11.5398
run;
title1 'Diagnosis Sequence Diagram.';

/* make sure ods listing is open when running macro */
ods listing;

%ds2const(ndata=nodedata,
          ldata=linkdata,
          datatype=assoc,
          minlnkwt=30,
          height=450,
          width=600,
          codebase=http://your_path_to_archive,
          htmlfile=your_path_and_filename.htm,
          xmltype=external,
          makexml=y,
          xmlurl=http://www.xyz.com/Web_output/const_assoc_external.xml,
          xmlfile=u://Web_output/const_assoc_external.xml,
          colormap=y,
          nid=nodeID,
          nlabel=label,
          nvalue=value,
          fntsize=12,
          ntip=tip,
          lfrom=from,
          lto=to,
          lvalue=linkvalue,
          ltip=tip,
          linktype=arrow);

```

Constellation Chart with Hotspots

This sample program generates the same Constellation diagram as in “Constellation Chart with DATATYPE=ARCS” on page 518 and adds hotspots to the nodes of the diagram.

SAS Code

The following is the complete SAS code to generate the Constellation diagram from a SAS data set. Notice the following:

- The parameter NURL= specifies the variable in the SAS data set that contains the URL to be linked to when a user double-clicks the node.

```

/*Define a nodes data set of countries and languages */
data nodedata;
input nodeLabel $15. shape $10. color $8. size url $40.;
cards;
France          square    red        .1 http://www.xyz.com
Germany         square    red        .1 http://www.xyz.com/rnd/webgraphs/
Italy           square    red        .1 http://www.xyz.com

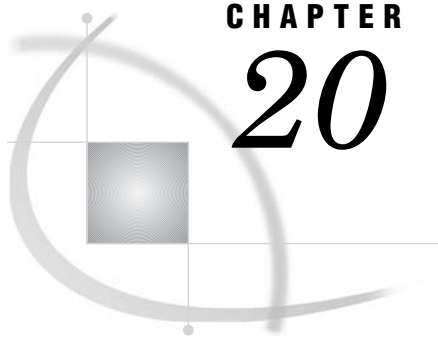
```

```

Belgium      square    red      .1 http://www.xyz.com/rnd/webgraphs/
Switzerland  square    red      .1 http://www.xyz.com
Holland      square    red      .1 http://www.xyz.com
German       triangle  blue     .1 http://www.xyz.com
French       triangle  blue     .1 http://www.xyz.com/rnd/webgraphs/odssyntax.htm
Italian      triangle  blue     .1 http://www.xyz.com
Flemish      triangle  blue     .1 http://www.xyz.com
Dutch        triangle  blue     .1 http://www.xyz.com
;
run;

/*Define a links data set: */
data linkdata;
input from $15. to $15.;
cards;
France      French
Germany     German
Belgium     French
Belgium     German
Belgium     Flemish
Belgium     Dutch
Switzerland French
Switzerland German
Switzerland Italian
Italy       Italian
Italy       German
Holland     Dutch
;
run;
goptions reset=all;
/* make sure ods listing is open when running macro */
ods listing;
/*Run the DS2CONST macro:*/
%ds2const(ndata=nodedata,
          ldata=linkdata,
          nurl=url,
          datatype=arcs,
          cnode=red,
          colormap=y,
          height=400,
          width=500,
          code=ConstChart,
          codebase=http://your_path_to_archive,
          htmlfile=your_path_and_filename.htm,
          nid=nodelabel,
          nlabel=nodelabel,
          lfrom=from,
          lto=to,
          fntsize=12,
          nshape=shape,
          ncolor=color,
          nsize=size);

```

CHAPTER 20

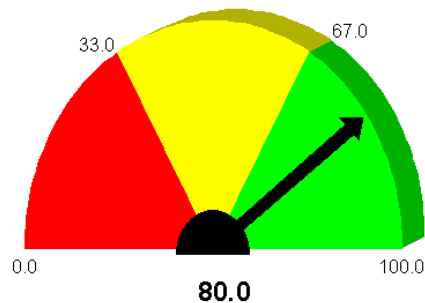
Creating Critical Success Factor Diagrams

<i>Using the DS2CSF Macro</i>	527
<i>When to Use the Rangeview Applet</i>	528
<i>Programming with the DS2CSF Macro for the Rangeview Applet</i>	528
<i>Enhancing Presentations for the Rangeview Applet</i>	529
<i>DS2CSF Macro Arguments</i>	530
<i>Sample Programs: DS2CSF Macro</i>	530
<i>Sample Diagrams Using DS2CSF</i>	530
<i>Results Shown in a Browser</i>	530
<i>SAS Code</i>	531
<i>Adding a Link to a Critical Success Factor Diagram</i>	532
<i>SAS Code</i>	532

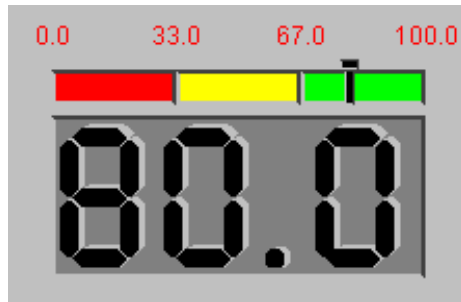
Using the DS2CSF Macro

The Rangeview Applet generates a critical success factor (CSF) diagram, as shown in the following figure.

Display 20.1 Critical Success Factor Diagram



The critical success factor diagram is fully configurable, based on either the classical style (shown above) or a style that resembles a digital gauge, as shown in the following figure.

Display 20.2 Digital Display

Ranges of values, colors, text sizes, and text fonts are all specified in the SAS/GRAPH program. You can also specify a drill-down URL that will be displayed in a specified browser window when the Web user selects any part of the diagram.

When to Use the Rangeview Applet

The Rangeview Applet provides a simple means of displaying a significant value within a range of values. The drill-down functionality enables you to provide additional information that is related to that critical success factor. While the level of interactivity is not high, the applet can generate CSF diagrams quickly, and uses your own style sheets to standardize the appearance of the diagram.

Programming with the DS2CSF Macro for the Rangeview Applet

The DS2CSF macro generates Web output for the Rangeview Applet. This macro generates and formats an HTML output file, identifies the location of the Rangeview Applet's Java archive, configures the diagram, and provides the data that the applet uses to display the diagram.

Follow the steps shown within this code to generate an HTML output file that will display a critical success factor diagram.

```

/* Define the name and storage location of the HTML output file */
/*   and the location of the jar file                               */
%let htmlfile = your_path_and_filename.htm;
%let jarfile = http://your_path_to_archive;

/* Run a DATA step */
data test;
  myvar=80;
run;

/* Specify an optional title */
title1 'Test Results Indicate Readiness';

/* Run the DS2CSF macro */
%ds2csf(htmlfile=&htmlfile, /* files and data sets */
        archive=rvapplet.jar,
        codebase=&jarfile,
        openmode=replace,
        data=test,
        var=myvar,          /* CSF variable */

```

```

csftyp=digital,      /* Specify appearance */
bgtype=color,
bg="#E0E0E0",
tcolor="#002288",
tsize=5,
tface="Arial",
fcolor="#002288",
fsize=3,
clabval=red,
clabtxt=fuchsia,
depth=Two_And_A_Half_Dimension,
valuepos=Bottom_Center,
labelpos=Top_Center);

```

Display the HTML output file in a Web browser to run the Rangeview Applet and display a critical success factor diagram.

In the preceding example, the arguments of the DS2CSF macro specify the path to the HTML output file and the Java archive of the Rangeview Applet. The VAR argument identifies the variable in the data set that is to be illustrated in the diagram. The value that is to be indicated as the critical success factor is the value for this variable in the first observation in the data set. A digital-style diagram is specified by the CSFTYP= argument.

For more information on usages of the Rangeview Applet, see “Enhancing Presentations for the Rangeview Applet” on page 529.

For definitions of all of the arguments of the DS2CSF macro, see “DS2CSF Macro Arguments” on page 530.

Enhancing Presentations for the Rangeview Applet

While the Rangeview Applet does not provide interactivity other than a drill-down URL, a number of enhancements enable you to control the appearance of the diagram, as described in the following table.

Table 20.1

Enhancement	DS2CSF Argument
Specify a stylesheet to format your HTML output file.	SSFILE, SSHREF, etc. See “Arguments for Stylesheets” on page 554.
Specify a color for the background, hub, and outline.	CBACK, CHUB, COUTLINE
Specify a drill-down URL and display target.	DRILURL, DRILTARG

Specify an analog or digital gauge.	CSFTYPE
Set the indicator shape, height, width, and color.	INDICTYP, HINDIC, WINDIC, CINDIC

DS2CSF Macro Arguments

The DS2CSF macro generates HTML output for the Rangeview Applet. The last three letters, CSF, stand for critical success factor, which is the type of diagram that is generated by this applet.

Macro arguments specify the configuration of the HTML output file, the location of the data used to generate the diagram, and the customizations, such as a drill-down URL, that you can add to your presentation.

The DS2CSF macro uses the following syntax:

```
%DS2CSF(argument1=value1, argument2=value2, ...);
```

The arguments of the DS2CSF macro can be divided into the following categories:

- “Arguments for the APPLET Tag” on page 536
 - The ARCHIVE and CODEBASE arguments are required.
- “DS2CSF Arguments for Data Definition” on page 561
- “Arguments for Generating HTML and XML Files” on page 544
- “DS2CSF Arguments for Diagram Appearance” on page 562
- “Arguments for Page Formatting” on page 552
- “Arguments for Stylesheets” on page 554
- “Arguments for the SAS TITLE and FOOTNOTE Tags” on page 556
- “Arguments for Character Transcoding” on page 561.

Sample Programs: DS2CSF Macro

The following sample programs generate diagrams using the DS2CSF macro:

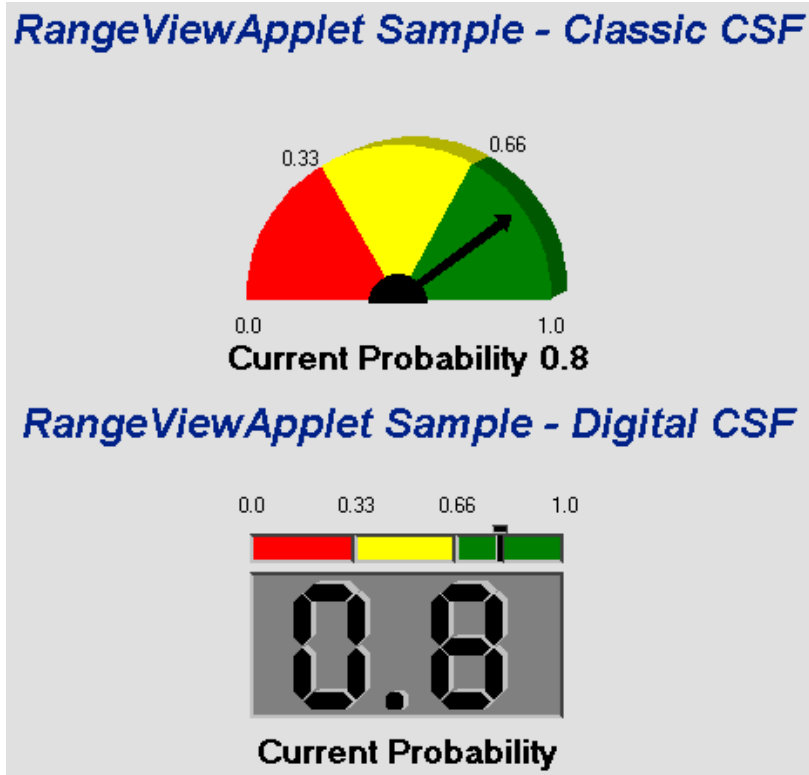
- “Sample Diagrams Using DS2CSF” on page 530
- “Adding a Link to a Critical Success Factor Diagram” on page 532.

Sample Diagrams Using DS2CSF

A CSF (Critical Success Factor) diagram represents a value in a range of data. This value can be obtained from a SAS data set variable or an SCL variable. The range of data used by the CSF is depicted by segments, with each segment containing a beginning and ending value, and possibly a different color. The number of segments and their corresponding colors are defined in a RANGE entry that is associated with the CSF.

Results Shown in a Browser

The sample program displays two forms of CSF—one an analog dial, and the other a digital display.



SAS Code

The following is the complete SAS code to generate a web presentation. Note the following:

- The HTMLFILE= parameter specifies the name of the HTML file to be produced by DS2CSF. If you want to run this sample, then change the value of HTMLFILE= to something that makes sense for you.
- The data set variable X specifies the value to be displayed on the CSF diagram. In this case, the critical value equals 0.8.
- The parameter

```
range=sashelp.javagrfsample1.range
```

specifies a range entry that defines the range segments and their corresponding color. In this case, the range entry defines three segments: 0–0.33, 0.33–0.66, and 0.66 to 1.

- The data set variable X specifies the value to be displayed on the CSF diagram. In this case, the critical value equals 0.8.

```
data test; x=0.8; label x='Current Probability'; run;
```

```
title 'RangeViewApplet Sample - Classic CSF';
```

```
%ds2csf(data=test,
        var=x,
        htmlfile=your_path_and_filename.htm,
        openmode=replace, pagepart=head,
        center=y,
```

```

it is not in the same directory as the html file */
    archive=http://your_path_to_archive,
    csftyp=classic, septype=none, cback=#e0e0e0,
    bgtype=color, bg="#e0e0e0",
    ttag=bold + italicized, tcolor="#002288", tsize=5,
    tface="Arial, Helvetica",
    range=sashelp.javagrfsample1.range);

title 'RangeViewApplet Sample - Digital CSF';

%ds2csf(data=test,
    var=x,
    htmlfile=your_path_and_filename.htm,
    openmode=append, pagepart=foot,
    center=y,
    archive=rvapplet.jar,
    /* specify complete url if not in same directory as the html file */
    archive=http://your_path_to_archive,
    csftyp=digital, labelpos=Bottom_Center, cback=#e0e0e0,
    bgtype=color, bg="#e0e0e0",
    ttag=bold + italicized, tcolor="#002288", tsize=5,
    tface="Arial, Helvetica",
    range=sashelp.javagrfsample1.range);

```

Adding a Link to a Critical Success Factor Diagram

You can add a hotspot to a diagram created with the DS2CSF macro. However, unlike the Treeview and Constellation macros, you can add only a single hotspot to a diagram, that is, to the diagram as a whole. You can not add hotspots to different portions of the diagram.

SAS Code

The SAS code creates a diagram and links to the specified URL when a user clicks anywhere on the diagram. DRILURL= specifies the URL to link to, while DRILTARG=_self specifies that the new Web page is to be displayed in the same window as the diagram.

```

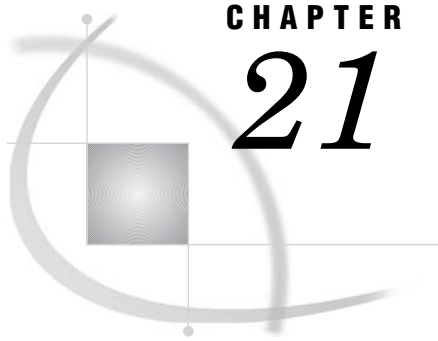
data test; x=0.8; label x='Current Probability'; run;

title 'RangeViewApplet Sample - Classic CSF';

%ds2csf(data=test,
    var=x,
    htmlfile=your_path_and_filename.htm,
    openmode=replace, pagepart=head,
    center=y,
    drilurl=http://www.xyz.com,
    archive=rvapplet.jar,
    /* specify complete url if not in same directory as the html file */
    archive=http://your_path_to_archive,
    csftyp=classic, septype=none, cback=#e0e0e0,
    bgtype=color, bg="#e0e0e0",
    ttag=bold + italicized, tcolor="#002288", tsize=5,

```

```
tface="Arial, Helvetica",  
range=sashelp.javagrfsample1.range);
```

CHAPTER

21

Macro Arguments for the DS2CONST, DS2TREE, DS2CSF, and META2HTM Macros

<i>Macro Arguments</i>	535
<i>Arguments for the APPLETTAG Tag</i>	536
<i>DS2TREE and DS2CONST Arguments for Data Definition</i>	537
<i>Arguments for Generating HTML and XML Files</i>	544
<i>DS2TREE and DS2CONST Arguments for Diagram Appearance</i>	545
<i>Arguments for Page Formatting</i>	552
<i>Arguments for Stylesheets</i>	554
<i>Arguments for the SAS TITLE and FOOTNOTE Tags</i>	556
<i>Arguments for Character Transcoding</i>	561
<i>DS2CSF Arguments for Data Definition</i>	561
<i>DS2CSF Arguments for Diagram Appearance</i>	562
<i>META2HTM Arguments for Saving the HTML File</i>	564
<i>META2HTM Arguments for Applet Behavior</i>	565
<i>Reserved Names</i>	566

Macro Arguments

Macro arguments specify the configuration of the HTML output file, the location of the data that is used to generate the diagram, and the configuration of the applet's interactive features.

The macros use the following syntax:

```
%macroname(argument1=value1, argument2=value2, ...);
```

The macro arguments can be divided into arguments used by all macros, arguments used by the DS2CSF macro, and arguments used by the META2HTM macro. The following arguments apply to all macros:

- “Arguments for the APPLETTAG Tag” on page 536. The CODEBASE argument is required.
- “DS2TREE and DS2CONST Arguments for Data Definition” on page 537. For DS2TREE the arguments NDATA and NID are required. For DS2CONST the arguments NDATA, NID, LDATA, and LTO are required.
- “Arguments for Generating HTML and XML Files” on page 544.
- “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545.
- “Arguments for Page Formatting” on page 552.
- “Arguments for Stylesheets” on page 554.
- “Arguments for the SAS TITLE and FOOTNOTE Tags” on page 556.
- “Arguments for Character Transcoding” on page 561.

The following arguments apply only to the DS2CSF macro:

- “DS2CSF Arguments for Data Definition” on page 561.
- “DS2CSF Arguments for Diagram Appearance” on page 562.

The following arguments apply only to the META2HTM macro:

- “META2HTM Arguments for Saving the HTML File” on page 564. All these arguments are required for saving the HTML file.
- “META2HTM Arguments for Applet Behavior” on page 565.

Arguments for the APPLET Tag

The following arguments configure the APPLET tag in the HTML output file. The CODEBASE argument is required.

AHUNITS=PIXELS | PERCENT

specifies the units of the HEIGHT= argument. The default value is PIXELS. See also the AWUNITS= argument.

Used by: DS2TREE, DS2CONST

ALIGN=position

specifies the alignment of the applet window in the browser window or frame. Values can be LEFT, RIGHT, TOP, BOTTOM, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, or ABSBOTTOM.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

ALT=text

specifies the text that will be displayed on mouseover by browsers that understand the tag but cannot run Java applets. The default value is **SAS Institute Inc. applet_name**.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

ARCHIVE=filename

specifies the name of the Java archive file(s). This argument is required for DS2CSF and META2HTM.

Note: The path to the Java archive is specified in the CODEBASE argument. △

The following table shows what archive files to use with each of the macros. For DS2TREE and DS2CONST, you do not have to specify a value for ARCHIVE= because the values shown are generated by default.

<i>DS2TREE</i>	archive=%str(sas.graph.treeview.jar, sas.graph.nld.jar, sas.graph.j2d.jar)
<i>DS2CONST</i>	archive=%str(sas.graph.constapp.jar, sas.graph.nld.jar, sas.graph.j2d.jar)
<i>DS2CSF</i>	archive=rvapplet.jar
<i>META2HTM</i>	archive=metafile.zip

Note: Before SAS 9.1, treeview.jar and constapp.jar also contained the classes that are now included in the auxiliary JAR files (sas.graph.nld.jar and sas.graph.j2d.jar). Although you can continue to use the older JAR files by specifying ARCHIVE=treeview.jar or ARCHIVE=constapp.jar, future versions may not support these older JAR files. △

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

AWUNITS=PIXELS | PERCENT

specifies the units of the WIDTH= argument. The default value is PIXELS. See also the HEIGHT= and AHUNITS= arguments.

Used by: DS2TREE, DS2CONST

CODEBASE=path-or-URL

specifies the path of the SAS Java archives specified in the ARCHIVE= argument. *The CODEBASE argument is required.* You can specify CODEBASE="." if the HTML file and Java archive files are in the same directory.

Note: You can specify the location pointed to by the SAS system option APPLETLOC=, or you can specify a different location. To display the current value of APPLETLOC, run the following code:

```
proc options option=appletloc;
run;
```

The value of the APPLETLOC system option is not used as the default value. Δ

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

HEIGHT=applet-height

specifies the height of the applet window. The unit of measure is pixels unless changed by the AHUNITS= argument. The default value is 600 for all macros except the DS2CSF macro. The default for the DS2CSF macro is 175.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

HSPACE=pixels

specifies the amount of horizontal space, in pixels, to the left and right of the graph or diagram.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

NAME=applet-name

specifies the name for this instance of the applet. You need to use this argument only if you have more than one instance of the APPLET tag in your HTML file, and if you have included your own scripts or DHTML that communicates with or acts on a particular instance of the applet.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

VSPACE=pixels

specifies the amount of vertical space, in pixels, to the top and bottom of the graph or diagram.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

WIDTH=applet-width

specifies the width of the applet window. The unit of measure defaults to pixels unless specified by the AWUNITS= argument. The default value is 800 for all macros except the DS2CSF macro. The default for the DS2CSF macro is 225.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

DS2TREE and DS2CONST Arguments for Data Definition

The following arguments for the DS2TREE and DS2CONST macros define how the applet will use the data set to generate the node/link diagram.

For DS2TREE the arguments NDATA and NID are required.

For DS2CONST the arguments NDATA, NID, LDATA, and LTO are required.

DATATYPE=ARCS | ASSOC | HIER

specifies the type of the XML data. Valid values are defined as follows:

ARCS

indicates that the data set is in the form of an arc list. This is the default value.

ASSOC

indicates that the data set is associative. The links can be displayed based on their weighted values, and node size and link width can represent the relative size of the node and link values.

HIER

indicates that the data set is hierarchical.

Used by: DS2CONST

LABELS=Y | N

indicates whether or not node labels are displayed in the diagram. The default value is Y.

Used by: DS2CONST, DS2TREE

LAYOUT=AUTO | USER

when the value is AUTO (default), specifies that the Constellation Applet lays out the diagram using stress and strain equations. Specifying the value USER indicates that the node positions are specified in the NX and NY arguments.

Used by: DS2CONST

LCOLOR=*variable-name*

specifies the name of the variable that determines the color of the link lines. The values of this variable must be HTML 3.2 color names, or you must use the LCOLFMT= argument to convert those values to valid color names. The default color is provided by the CLINK= argument (see “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545).

In the DS2CONST macro, the LCOLOR= argument is overridden by the LCOLVAL= argument.

Used by: DS2CONST, DS2TREE

LCOLFMT=*user-defined-format-name*

specifies the name of a user-defined SAS format that converts the values in the variable named in the LCOLOR= argument to valid HTML color names. Note that the SAS format does not change any values in the data set. The formatted values are applied to the diagram only.

Used by: DS2CONST, DS2TREE

LCOLVAL=*variable-name*

specifies the name of the variable that determines the color mapping of link lines. This argument is valid only when the value of the DATATYPE= argument is ASSOC, and only when the value of the COLORMAP= argument is Y. If the LCOLVAL= argument is not specified, the link colors are determined by the following arguments in the following order: LCOLOR= (see above) and CLINK= (see “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545).

Used by: DS2CONST

LDATA=*data-set-name*

specifies the name of the SAS data set that contains the link data that is used to generate the diagram.

This argument is required.

Used by: DS2CONST

LFROM=*variable-name*

specifies the name of the variable whose values define the nodes at the start of link lines. The LFROM variable values must be coordinated with the values of the variables that are named in the NID= and LTO= arguments.

This argument is required.

Used by: DS2CONST

LINKTYPE=LINE | ARROW

when the value is ARROW (default), indicates that link lines are to be drawn with arrowheads that indicate the direction of flow.

Used by: DS2CONST

LPT=*password*

specifies the password that is needed for accessing a password-protected link data set (specified with the LDATA= argument). The LPT= argument is required if the data set has a READ or PW password. You do not need to specify this argument if the data set has a WRITE or ALTER password.

Used by: DS2CONST

LSTIP=*variable-name*

specifies the name of the variable in the data set that determines the stipple mask. The stipple mask generates dashed or dotted link lines. The value of the variable must be an integer, which is then converted into a binary value. In the binary value, a "1" bit means that a pixel is to be drawn and a "0" bit means that no pixel is to be drawn. For example, if the variable has a value of 61680, the binary conversion of that value will be 1111000011110000. This stipple mask generates a dashed link line with dashes and spaces that are four pixels wide. See also the LSTIPFAC= argument.

Used by: DS2CONST, DS2TREE

LSTIPFAC=*variable-name*

specifies the name of the variable in the data set whose value specifies a multiplier for the binary stipple mask (see the LSTIP= argument). The multiplier lengthens the dashes in the base mask. For example, if the multiplier is 2, a stipple mask that specifies 4-pixel dashes and 4-pixel spaces will generate link lines with 8-pixel dashes and spaces.

Used by: DS2CONST, DS2TREE

LTIP=*variable-name*

specifies the name of the variable in the data set that provides the text that is displayed in the pop-up data tips windows for links.

Used by: DS2CONST, DS2TREE

LTIPFMT=*user-defined-format-name*

specifies the name of a user-defined SAS format that is applied to the values in the variable specified in the LTIP= argument to configure those values for display in the pop-up data tips window. Note that the SAS format does not change any values in the data set. The formatted values are applied to the diagram only.

Used by: DS2CONST, DS2TREE

LTO=*variable-name*

specifies the name of the variable whose values identify the nodes at the ends of link lines. The LTO variable values must be coordinated with the values of the variables that are named in the LFROM and NID arguments.

This argument is required.

Used by: DS2CONST

LVALUE=*variable-name*

specifies the name of the variable whose values determine the weights of the link lines, which determines the color and relative thickness of link lines. The variable values must be real numbers. The link weights are used with the MINLNKWT= argument (see below) and the SCLNKWT= argument (see “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545) to control the display of link lines. The LVALUE= argument is valid only when the value of the DATATYPE= argument is ASSOC.

Used by: DS2CONST

LWHERE=*subset-expression*

specifies a WHERE clause that subsets the link data for display in the diagram. If the expression contains any special characters (for example, % or &), include %NRBQUOTE in the expression to process those characters correctly. The following example shows how to correctly specify INT%:

```
LWHERE=%NRBQUOTE(value="Int%")
```

See also the NWHERE argument.

Used by: DS2CONST

LWIDTH=*variable-name*

specifies the name of the variable in the data set that determines the width of the link lines.

For DS2CONST: When this argument is not specified, the width is determined by the LVALUE argument. This argument is valid for DS2CONST only when the value of the DATATYPE argument is ASSOC.

Used by: DS2CONST, DS2TREE

MINLNKWT=*minimum-link-weight*

specifies the initial minimum link weight, which determines which links are initially displayed. The initial diagram show only those links that have weights that are greater than or equal to the minimum weight. In the Constellation Applet, a scroll bar allows the Web user to change the minimum link weight to change the number of links that are displayed. Selecting the browser’s Refresh option restores the initial minimum link weight that is specified in the MINLNKWT argument. Link weights are determined by the LVALUE argument. This argument is valid only when the value of the DATATYPE argument is ASSOC.

Used by: DS2CONST

NACTION=*variable-name*

specifies the name of the variable in the nodes data set that provides the menu text that is displayed when the Web user selects a node with the right mouse button. Selecting this menu option text displays the URL that is associated with that node in the NURL= argument. This argument overrides the ACTION= argument (see “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545). The default menu option text is **Open URL**.

Used by: DS2CONST, DS2TREE

NCOLFMT=*SAS-format-name*

specifies the name of a user-defined SAS format that converts the values in the variable named in the NCOLOR= argument to valid HTML color names. Note that the data in the data set is not altered; the formatted value is used in the hierarchical tree rather than the data value.

Used by: DS2CONST, DS2TREE

NCOLOR=variable-name

specifies the variable in the nodes data set that determines the background color of the nodes, using HTML 3.2 color names or 6-digit hexadecimal RGB values . If the variable does not contain valid HTML color names, then you can use the `NCOLFMT=argument` to convert those values to the HTML color names. See also the `NCOLVAL=` and `NVALUE=` arguments.

Used by: DS2CONST, DS2TREE

NCOLVAL=variable-name

specifies the name of the variable in the nodes data set that determines the color mapping for the nodes. This argument is valid only when the `DATASET=` argument is set to `ASSOC`, and only when the value of the `COLORMAP=` argument is `Y`. If this argument is not specified, then the node color is determined by the `LVALUE=` argument.

Used by: DS2CONST

NDATA=SAS-data-set-name

specifies the SAS data set that contains the node data.

This argument is required.

Used by: DS2CONST, DS2TREE

NFNTNAME=node-font-variable-name

specifies the name of the variable that determines the text font for the node labels. The variable value can be `SERIF`, `SANSSERIF`, `DIALOG`, `DIALOGINPUT`, or `MONOSPACED`. The default node font is specified by the `FNTNAME=` argument (see “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545).

Used by: DS2CONST, DS2TREE

NFNTSIZE=variable-name

specifies the name of the variable in the nodes data set that determines the size of the text font used for node labels. This font size is expressed in points. This argument overrides the `FNTSIZE=` argument.

Used by: DS2CONST, DS2TREE

NFNTSTYL=node-font-style-variable-name

specifies the name of the variable that determines the font style for the node label. The valid values that can be assigned to the variable are `BOLD`, `ITALIC`, and `PLAIN`.

Used by: DS2CONST, DS2TREE

NID=variable-name

specifies the name of the variable in the nodes data set whose values are to illustrated as the nodes in the diagram. The node ID variable type can be either numeric or character. For the `DS2CONST` macro, the values of the `NID` variable must be coordinated with the values of the `LFROM` and `LTO` variables.

This argument is required.

Used by: DS2CONST, DS2TREE

NLABEL=node-label-variable-name

specifies the name of the variable that represents the node labels. This variable type can be either numeric or character.

Used by: DS2CONST, DS2TREE

NPARENT=*node-parent-variable-name*

specifies the name of the variable that represents the parent nodes. This variable type can be either numeric or character.

Used by: DS2TREE

NPW=*password*

specifies the password that is needed for accessing a password-protected data set. This argument is required if the data set has a READ or PW password. You do not need to specify this argument if the data set has only WRITE or ALTER passwords.

Used by: DS2CONST, DS2TREE

NSCBACK=*variable-name*

specifies the name of the variable in the node styles data set that determines the background color of the nodes. The variable values must be HTML 3.2 color names. The default value is determined by the CNODE= argument (see “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545).

Used by: DS2CONST, DS2TREE

NSCTEXT=*variable-name*

specifies the name of the variable in the node styles data set that provides the colors for the node label text. Valid variable values must be HTML 3.2 color names. The default color is provided by the CATEXT= argument.

Used by: DS2CONST, DS2TREE

NSDATA=*SAS-data-set-name*

specifies the name of the node styles data set.

Used by: DS2CONST, DS2TREE

NSFNTNAM=*variable-name*

specifies the name of the variable in the node styles data set that determines the text font that is to be used for node labels. Valid variable values can be SERIF, SANSSERIF, DIALOG, DIALOGINPUT, or MONOSPACED. This argument overrides the FNTNAME= argument.

Used by: DS2CONST, DS2TREE

NSFNNTSIZ=*variable-name*

specifies the name of the variable in the node styles data set that determines the size of the node label text, in points. This argument overrides the FNTSIZE= argument.

Used by: DS2CONST, DS2TREE

NSFNTSTY=*variable-name*

specifies the name of the variable in the node styles data set that determines the style of the node label text. Valid variable values can be BOLD, ITALIC, or the default value, PLAIN. This argument overrides the FNTSTYL= argument.

Used by: DS2CONST, DS2TREE

NSHAPE=*variable-name*

specifies the name of the variable that determines the shape of the nodes. Valid variable values can be CIRCLE, DIAMOND, NONE, SQUARE, or TRIANGLE. The default value is SQUARE. This argument overrides the NODESHAPE= argument.

Used by: DS2CONST

NSID=*variable-name*

specifies the name of the variable in the node styles data set that represents the nodes.

Used by: DS2CONST, DS2TREE

NSIZE=*variable-name*

specifies the name of the variable that determines the size of the nodes. The values of this variable can be real numbers. Node sizes are determined based on the value of the LAYOUT= argument. When LAYOUT=USER, the values of the NSIZE variable are interpreted as literal pixel measurements. When LAYOUT=AUTO, the values of the NSIZE variable determine the size of the nodes based on the relative size of individual values. The values of the NSIZE variable can be scaled with the SCLNSIZE= argument (see “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545). This argument is valid only when the value of the DATATYPE= argument is ASSOC.

Used by: DS2CONST

NSPW=*password*

specifies the password that is needed to access a password-protected node styles data set. This argument is required if the data set has a READ or PW password. You do not need to specify this argument if the data set has only WRITE or ALTER passwords.

Used by: DS2CONST, DS2TREE

NSTYLE=*variable-name*

specifies the name of the variable that determines the style of the nodes. This variable type can be either numeric or character, and the values must correspond to the node identifiers specified in the NSID= argument.

Used by: DS2CONST, DS2TREE

NSWHERE=*subset-expression*

specifies a WHERE clause that subsets the node styles data set for display in the diagram. If the expression contains any special characters (for example, % or &), then include %NRBQUOTE in the expression to process those characters correctly. The following example shows how to correctly specify INT%:

```
NSWHERE=%NRBQUOTE(value="Int%")
```

Used by: DS2CONST, DS2TREE

NTEXTCOL=*variable-name*

specifies the name of the variable that determines the color of the text for the node labels. Valid variable values must be HTML 3.2 color names.

Used by: DS2CONST, DS2TREE

NTIP=*variable-name*

specifies the name of the variable that provides the data or text that is displayed in the pop-up data tips window.

Used by: DS2CONST, DS2TREE

NTIPFMT=*user-defined-format-name*

specifies the name of a user-defined SAS format that is applied to the data tips variable that is named in the NTIP= argument. Note that the data set is not altered; the formatted value is used only in the diagram.

Used by: DS2CONST, DS2TREE

NURL=*drill-down-URL*

specifies the name of the variable that provides the drill-down URLs for the nodes. These URLs are displayed when the Web user double-clicks on a node or selects the node with the right mouse button and chooses an option from the pop-up menu. Menu text is determined by the NACTION= argument above and by the ACTION= argument in “DS2TREE and DS2CONST Arguments for Diagram Appearance” on page 545. The default menu option text is **Open URL**.

Used by: DS2CONST, DS2TREE

NVALUE=variable-name

specifies the name of the variable that determines the relative node size. This argument is valid only when DATATYPE=ASSOC.

If you do not specify a particular node color using either the NCOLOR or NCOLVAL argument (and if COLORMAP=Y), then this argument also determines a default node color. By default, the largest value of NVALUE is mapped to red, the median value to green, and the lowest value to blue. Values in between result in interpolated colors.

Used by: DS2CONST

NWHERE=subset-expression

specifies a WHERE clause that subsets the nodes data set for display in the diagram. If the expression contains any special characters (for example, % or &), then include %NRBQUOTE in the expression to process those characters correctly. The following example shows how to correctly specify INT%:

```
NWHERE=%NRBQUOTE(value="Int%")
```

See also the LWHERE= argument.

Used by: DS2CONST, DS2TREE

NX=variable-name

NY=variable-name

specify the variables that determine the locations of the centers of the nodes.

These arguments are valid only when the LAYOUT= argument is set to USER.

The values are expressed in pixels. Positive values are measured from the top-left corner of the screen. Negative values are measured from the bottom-right corner of the screen.

Used by: DS2CONST

Arguments for Generating HTML and XML Files

The following arguments determine the name, storage location, and file makeup of Web presentations that run in the Constellation Applet or the Treeview Applet.

HTMLFILE=external-filename

specifies the name and storage location of the HTML output file. If the external file does not exist, then it is created for you. Either this argument, or HTMLFREF=, is required if you specify MAKEHTML=Y. Note: Do not use the HTMLFILE= argument if you use the HTMLFREF= argument.

Used by: DS2TREE, DS2CONST, DS2CSF

HTMLFREF=fileref

specifies the SAS fileref that identifies the name and storage location of the HTML output file. If the external file does not exist, then it is created for you. Either this argument, or HTMLFILE=filename, is required if you specify MAKEHTML=Y.

Note: Do not use the HTMLFREF= argument if you use the HTMLFILE= argument, and do not use a reserved name (see “Reserved Names” on page 566).

Used by: DS2TREE, DS2CONST, DS2CSF

MAKEHTML=Y | N

specifies whether or not an HTML file is to be generated. The default value is Y, which generates the HTML output file. If you specify MAKEHTML=N and MAKEXML=Y, then only an XML file is generated.

Used by: DS2TREE, DS2CONST

MAKEXML=Y | N

specifies whether or not an XML file is to be generated. The default value is Y, which generates the XML output file. If you specify MAKEXML=N and MAKEHTML=Y, then only an HTML file will be generated. Note that under these circumstances, you must specify a value for the XMLURL= argument.

Used by: DS2TREE, DS2CONST

OPENMODE=REPLACE | APPEND

indicates whether the new HTML or XML output or both overwrites the information that is currently in the specified file(s), or if the new output is appended to the end of the existing file(s). The default value is REPLACE. Specify APPEND to add your new HTML-enhanced output to the end of an existing file.

Note: OPENMODE=APPEND is not valid if you are writing your resulting HTML to a partitioned data set (PDS) on z/OS.

Used by: DS2TREE, DS2CONST, DS2CSF

RUNMODE=B | S

specifies whether you are running the DS2TREE macro in batch or server mode. Batch mode (RUNMODE=B, the default) means that you are submitting the DS2TREE macro in the SAS Program Editor or you have included it in a SAS program. Server mode (RUNMODE=S) generates the HTTP header that is required by Application Dispatcher in the SAS/INTRNET software.

Used by: DS2TREE, DS2CONST, DS2CSF

XMLFILE=external-filename

specifies the name and storage location of the XML output file. If the external file does not exist, then it is created for you. This argument, or XMLFREF=, is required if you specify MAKEXML=Y and XMLTYPE=EXTERNAL. Note: Do not use the XMLFILE= argument if you use the XMLFREF= argument.

Used by: DS2TREE, DS2CONST

XMLFREF=fileref

specifies the SAS fileref that identifies the name and storage location of the XML output file. If the external file does not exist, then it is created for you. This argument, or XMLFILE=, is required if you specify MAKEXML=Y and XMLTYPE=EXTERNAL. Note: Do not use the XMLFREF= argument if you use the XMLFILE= argument, and do not use a reserved name (see “Reserved Names” on page 566).

Used by: DS2TREE, DS2CONST

XMLTYPE=INLINE | EXTERNAL

specifies whether the XML output file is to be written to an external file or included inline with the HTML. The default value is INLINE. If you specify EXTERNAL you must also specify a value for either the XMLFILE= or XMLFREF= arguments. This argument is required if you specify MAKEXML=Y.

Used by: DS2TREE, DS2CONST

XMLURL=URL

specifies the URL of the existing file that contains the XML tags that define the node/link diagram. This argument is required if specified XMLTYPE=EXTERNAL.

Used by: DS2TREE, DS2CONST

DS2TREE and DS2CONST Arguments for Diagram Appearance

The following arguments for the DS2TREE and DS2CONST macros specify non-default behavior and appearance of the node/link diagram in the respective applet.

None of the following arguments are required.

ACTION=*text*

specifies the default text that is displayed in a pop-up menu when the Web user selects a node with the right mouse button. Selecting this menu option displays the URL that is associated with that node in the NURL= argument. This argument is overridden by the NACTION= argument (see “DS2TREE and DS2CONST Arguments for Data Definition” on page 537). The ACTION= argument is useful when you want to use a single menu text string for most of the nodes in your diagram. The default menu option text is **Open URL**.

Used by: DS2CONST, DS2TREE

ANGLE=*link-angle*

works with the TREESPAN= argument to determine the direction of growth for the diagram. The ANGLE= argument is valid only when you do not specify the TREEDIR= argument. The TREESPAN= argument defines the angular width of the tree (narrow or wide layout). The TREESPAN angle can be visualized as a V shape, with the starting node positioned at the base of the V. The rest of the nodes are laid out between the spreading arms of the V. The ANGLE= argument specifies the angle of the V shape. By default, the value of the ANGLE= argument is zero (0) and the V shape opens to the right, as if the letter V was rotated 90 degrees clockwise, to the three-o’clock position. Values of the ANGLE= argument that are greater than zero rotate the V shape counterclockwise away from the three-o’clock position. Valid values for the ANGLE= argument range from zero (0) to 360 degrees.

Used by: DS2TREE

BORDER=Y | N

specifies whether or not a border is drawn around the background area. The default value is N.

Used by: DS2CONST, DS2TREE

CATEXT=*default-text-color*

specifies a default color for the text in the diagram, using an HTML 3.2 color name or a 6-digit hexadecimal RGB value. For DS2CONST, this argument is overridden by the FNTNAME= argument (see below) and the NTEXTCOL argument (see “DS2TREE and DS2CONST Arguments for Data Definition” on page 537).

Used by: DS2CONST, DS2TREE

CBACK=*color*

specifies a background color for the Treeview Applet. The value must be a valid HTML 3.2 color name.

Used by: DS2TREE

CHANDLE=*color*

specifies the color of the Collapse/Expand handle on the nodes. The handle is represented by a small plus sign (+) that is prefixed to the label of the node when its subtree is collapsed. The value must be a valid HTML color name.

Used by: DS2TREE

CLINK=*default-link-color*

specifies a default color for the links in the diagram, using an HTML 3.2 color name or a 6-digit RGB value. For DS2CONST, this argument is overridden by the LCOLOR= and LCOLVAL= arguments (see “DS2TREE and DS2CONST Arguments for Data Definition” on page 537).

Used by: DS2CONST, DS2TREE

CNODE=*color*

specifies the node background color. The value must be a valid HTML color name. The value specified here can be overridden by specifying a value of TRUE for the NCOLOR= argument.

Used by: DS2TREE

CNODE=*default-node-color*

specifies a default background color for the nodes, using an HTML 3.2 color name or a 6-digit RGB value. This argument is overridden by the NCOLOR=, NCOLVAL=, NVALUE=, or NSCBACK= arguments (see “DS2TREE and DS2CONST Arguments for Data Definition” on page 537).

Used by: DS2CONST

COLORMAP=N | Y

when the value is N (default), specifies that the Constellation Applet is to use the NCOLOR= and LCOLOR= arguments (see “DS2TREE and DS2CONST Arguments for Data Definition” on page 537) to determine node and link colors rather than using the color map.

Used by: DS2CONST

CSELECT=*color*

specifies a color for nodes that are selected by the mouse or as the result of a node search. The value must be a valid HTML 3.2 color name.

Used by: DS2CONST, DS2TREE

CUTOFF=*detail-percentage*

specifies the percentage of the nodes that will be displayed with node labels. After the percentage has been reached, nodes are drawn as rectangles. The size of those rectangles decreases as the distance from the starting node increases. Valid values range from 0.0 to 1.0 (The decimal value is mapped to a percentage from 0% to 100%). The default value is 1.0. See also the DEPTH argument.

Used by: DS2CONST, DS2TREE

DEPTH=*max-path-length*

specifies a whole number greater than zero that determines the maximum number of links that are to be displayed in the node/link diagram. Paths whose lengths exceed the limit are truncated. This argument affects only the initial display of the diagram. Nodes that are initially hidden can become visible as a user selects nodes and navigates around the diagram.

Note that this value is ignored if the value of the CUTOFF= argument is 1.0. There is no default value for this argument.

Used by: DS2TREE

DRILTARG=*target-window-or-frame*

specifies the HTML target or the name of the browser window or frame where drill-down URLs are displayed. The default behavior is to open a new browser window and reuse it for subsequent drill-down requests. Specifically, the default value is _BLANK, which is one of several reserved names for targets in HTML. The value can also be the name of a window or frame in the Web presentation.

Used by: DS2CONST, DS2TREE

DUPCHECK=TRUE | FALSE

specifies whether or not the applet will check for duplicate node IDs. The default value is FALSE. When set to TRUE, this argument will cause the applet to update an ID if a duplicate ID is found, instead of creating a new node with the same ID. This enables you to collect node information from different locations in the data set.

Used by: DS2TREE

FACTOR=*fish-eye-distortion-factor*

specifies the distortion factor for the fish-eye lens. The distortion factor determines the amount that the central region of the display is to be expanded (or zoomed). The value specified must be greater than or equal to 1.0. The default value is 1.0, which represents the lowest amount of distortion. This argument is valid only when the value of the FISHEYE= argument is Y. The maximum effective value (beyond which no further distortion is visible) is variable depending upon the number of nodes in the diagram.

Used by: DS2CONST, DS2TREE

FISHEYE=Y | N

indicates whether or not the diagram is to be displayed with the fish-eye distortion, which displays the central region of the diagram at a specified size and displays the rest of the diagram as if it were mapped onto a ball, with the nodes and links disappearing over a curved horizon. The Web user can move the diagram past the central region by scrolling or searching for nodes. The amount of distortion used in the fish-eye lens is determined by the FACTOR= argument. The default value is Y.

Used by: DS2CONST, DS2TREE

FNTNAME=*default-node-label-font*

specifies the default text font for node labels. Valid values can be SERIF, SANSSERIF, DIALOG, DIALOGINPUT, or MONOSPACED. This argument is overridden by the NFNTNAME or NSFNTNAM= arguments (see “DS2TREE and DS2CONST Arguments for Data Definition” on page 537).

Used by: DS2CONST, DS2TREE

FNTSIZE=*node-font-size*

specifies the size of the node label text font, in points. This argument is overridden by the NFNTSIZE= argument.

Used by: DS2CONST, DS2TREE

FNTSTYL=*node-font-style*

specifies the text font style for node labels. Valid values are BOLD, ITALIC, and PLAIN. PLAIN is the default value. This argument is overridden by the NFNTSTYL= argument.

Used by: DS2CONST, DS2TREE

IBACKLOC=*image-URL*

specifies a URL for the image that you want to use in the background of the diagram. See also the IBACKPOS= argument.

Used by: DS2CONST, DS2TREE

IBACKPOS=CENTER | SCALE | TILE | POSITION

specifies how to display the background image in the IBACKLOC= argument. Specify one of the following options:

CENTER

centers the image in the browser window without resizing the image.

SCALE

resizes the image to fit the browser window.

TILE

fills the browser window by replicating the image at its original size.

POSITION

positions the image without resizing at the values specified by the IBACKX= and IBACKY= arguments.

Used by: DS2CONST, DS2TREE

IBACKURL=*background-drilldown-URL*

specifies the URL that is displayed when you click on the background image. This argument is valid only when the value of the IBACKPOS= argument is POSITION. If you are including the Powered by SAS logo, then you must use this argument to link the image to the SAS Web site.

Used by: DS2CONST, DS2TREE

IBACKX=*corner-coordinate*

IBACKY=*corner-coordinate*

specifies the x (horizontal) and y (vertical) pixel coordinates of the upper left-hand corner of the background image. Positive values are measured from the upper-left corner of the background area. Negative values are measured from the lower-right corner of the background area. These values are valid only if the value of the IBACKPOS= argument is POSITION. Always specify both the IBACKX= and IBACKY= arguments.

Used by: DS2CONST, DS2TREE

NODEBDR=LINE | NONE | FILL | OUTLINE

specifies the appearance of the node border line, using one of the following values:

LINE

show solid border lines around the nodes.

NONE

show no border lines or background.

FILL

show background but no border lines.

OUTLINE

show a border line and background. This is the default value.

Used by: DS2TREE

NODESEP=*character(s)*

specifies which character(s) should be used to separate the selected nodes in the return value for the various getSelectedNodes methods. The default separator is a semicolon (;). If the getSelectedNodesIds method is called, and a JavaScript method is not specified with the SELIFUNC= argument, then the selected node IDs are returned as a single string separated by the NODESEP= character(s).

Used by: DS2CONST, DS2TREE

NODESHAP=*shape*

specifies the shape of the nodes. Valid values can be CIRCLE, DIAMOND, NONE, SQUARE, or TRIANGLE. The default value is SQUARE. This argument is overridden by the NSHAPE= argument (see “DS2TREE and DS2CONST Arguments for Data Definition” on page 537).

Used by: DS2CONST

RBSIZING=Y | N

the default value N indicates that size information from the resource bundle is not to be used for sizing the two dialog boxes that can be invoked from the pop-up menu that appears when a user right-mouse-clicks on a diagram. The two dialog boxes are the About dialog box and the Mouse Help dialog box.

Specify Y for this argument for languages other than English. If you specify Y, then the height and the width of the dialog box frames are read in from the resource bundle. This allows translators to set appropriate heights and widths for

the frames in the resource bundle, based on the length of the message strings in each language.

Used by: DS2CONST, DS2TREE

SCLNKWT=Y | N

when the value is Y (default), specifies that the link weight values are to be scaled into the range of 0–1, which corresponds to 0–100%. When SCLNKWT=Y, the scroll bar in Constellation Applet displays a percentage of the range of the link weights. When SCLNKWT=N, the link weights are not scaled and the scroll bar reflects the actual link weight data values. These values are real numbers that are specified in the LVALUE= argument (see “DS2TREE and DS2CONST Arguments for Data Definition” on page 537). The SCLNKWT= argument is valid only when the value of the DATATYPE= argument is ASSOC. Note that the range of link weights (maximum minus minimum) must be greater than 2 when SCLNKWT=N. Otherwise, the scroll bar will not correctly map the link weights.

Used by: DS2CONST

SCLWIDTH=Y | N

when the value is Y (default), indicates that the link width values are to be scaled into the range of 0–1. Specifying N indicates that the link widths are already scaled into that range. This argument is valid only when the value of the DATATYPE= argument is ASSOC.

Used by: DS2CONST

SCNSIZE=Y | N

when the value is Y (default), indicates that the node size values are to be scaled into the range of 0–1. Specifying N indicates that the node sizes are already scaled into that range. This argument is valid only when the value of the DATATYPE= argument is ASSOC. Node sizes are specified with the NSIZE= argument (see “DS2TREE and DS2CONST Arguments for Data Definition” on page 537).

Used by: DS2CONST

SELIFUNC=*JavaScript-method-name*

specifies the name of the JavaScript method that will be used by the `getSelectedNodesIds` method. The `getSelectedNodesIds` method first collects all the IDs for the selected nodes. Then, if a method is specified with the SELIFUNC= argument, the `getSelectedNodesIds` method calls that method and passes to that method all of the selected node IDs as arguments. If there is no method specified with the SELIFUNC= argument, then the `getSelectedNodesIds` method concatenates all of the IDs into a single string, separates the individual IDs with the value that is specified by the NODESEP= argument, and returns the string. The value for this argument is case-sensitive.

Used by: DS2CONST, DS2TREE

SELLFUNC=*JavaScript-method-name*

specifies the JavaScript method that will be used by the `getSelectedNodesLabels` method. The `getSelectedNodesLabels` method first collects all the labels for the selected nodes. Then, if a method is specified with the SELLFUNC= argument, the `getSelectedNodesLabels` method calls that method and passes to that method all of the selected node labels as arguments. If there is no method specified with the SELLFUNC= argument, then the `getSelectedNodesLabels` method concatenates all of the labels into a single string, separates the individual labels with the value that is specified by the NODESEP= argument, and returns the string. The value for this argument is case-sensitive.

Used by: DS2CONST, DS2TREE

SELUFUNC=JavaScript-method-name

specifies the JavaScript method that will be used by the `getSelectedNodesURLs` method. The `getSelectedNodesURLs` method first collects all the URLs for the selected nodes. Then, if a method is specified with the `SELUFUNC=` argument, the `getSelectedNodesURLs` method calls that method and passes to that method all of the selected node URLs as arguments. If there is no method specified with the `SELUFUNC=` argument, then the `getSelectedNodesURLs` method concatenates all of the URLs into a single string, separates the individual URLs with the value specified by the `NODESEP=` argument, and returns the string. The value for this argument is case-sensitive.

Used by: DS2CONST, DS2TREE

SHOWLINKS=Y | N

specifies whether initially to display all arc lines between nodes. Specifying `N` suppresses all arc lines. The default value is `Y`.

Note: This argument affects only the *initial* display. A viewer can subsequently control which arc lines are displayed by right-mouse clicking and selecting a **Show links** option from the pop-up menu. Δ

Used by: DS2CONST

SPREAD=angular-factor

specifies the angular spreading factor for the layout of the diagram. The value specified must be greater than or equal to 1.0. The default value is 1.25.

Used by: DS2TREE

TIPS=Y | N

indicates whether or not pop-up data tips are displayed when the cursor is positioned over nodes or links or both. The default value is `Y`.

Used by: DS2CONST, DS2TREE

TIPTYPE=TRACKING | STATIONARY

when the value is `TRACKING` (default), indicates that the pop-up data tips windows are to move with the cursor while the cursor moves within the area of a single node or link.

Used by: DS2CONST

TREEDIR=C | D | L | R | U

determines the growth direction of the node/link diagram using the following values.

C | CIRCULAR

grows the tree in a circular pattern. This is the default value.

D | DOWN

grows the tree from top to bottom using center alignment.

L | LEFT

grows the tree from left to right and top to bottom.

R | RIGHT

grows the tree from right to left and top to bottom.

U | UP

grows the tree from the bottom up using center alignment.

If the value of the `TREEDIR=` argument is `UP` or `DOWN`, then the value of the `TREESPAN=` argument is used to set the angular width of the diagram. The starting node is aligned horizontally in the center of the applet. The diagram

grows out of the starting node based on the angular width specified in the TREESPAN= argument. The wider the angle, the wider the layout of the diagram.

The TREEDIR= argument overrides the ANGLE= argument.

Used by: DS2TREE

TREESPAN=*angular-diagram-width*

specifies the angular width of the diagram in degrees. Valid values must be greater than zero and less than 360. The default value is 60. For details, see the TREEDIR= and ANGLE= arguments.

Used by: DS2TREE

ZOOM=*starting-percentage*

specifies the zoom value that is used for the initial display of the diagram. After the initial display, the Web user can change the zoom percentage using the slider-bar beneath the diagram on the Web page. Selecting the Refresh button on the browser runs the applet and restores the initial zoom setting. The default value is 100 percent. The initial diagram can be scaled up with a value greater than 100 or scaled down with a value less than 100.

Used by: DS2CONST

Arguments for Page Formatting

The following arguments format the HTML output file. The rendering of some of these arguments may vary in certain browsers. Several of the following arguments apply only to certain macros, as noted in the descriptions of the arguments.

The BGTYPE=, BRTITLE=, CENTER=, CTEXT=, and DOCTYPE= arguments apply to the entire page for the current invocation of the macro. If you append data to an existing HTML page, then the HTML formatting will not change. You may want to use these arguments only when you replace, rather than append, HTML files.

BDCLASS=*body-stylesheet-name*

specifies the name of the stylesheet that is to be applied to the body of the HTML output file.

Used by: DS2TREE, DS2CONST, DS2CSF.

BG=*color-or-image*

specifies the background color or image, based on the value of the BGTYPE= argument. The color can be specified as an HTML 3.2 color name or as a 6-digit hexadecimal RGB value. When BGTYPE=IMAGE, this argument specifies a background image, using a path or a URL, relative or absolute.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM.

BGTYPE=NONE | COLOR | IMAGE

specifies the background type, using one of the following values:

NONE

causes the applet to display its default background color. This is the default value.

COLOR

specifies that the value of the BG= argument must be an HTML 3.2 color name or hexadecimal RGB value.

IMAGE

specifies that the value of the BG= argument must be the path or URL pointing to an image file that will be displayed in the background of the applet window.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM.

BRTITLE=*browser-window-title*

specifies the text that appears in the title bar of the browser window. By default, no title is displayed.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM.

CENTER= Y | N

specifies whether or not the graph or diagram is centered in the browser window. The default value is N.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM.

CTEXT=*default-text-color*

specifies a default text color that replaces the default text color in the browser. Other color arguments can be used to override this new default. The color can be specified as an HTML 3.2 color name or as a six-digit hexadecimal RGB value.

Used by: DS2TREE, DS2CONST, DS2CSF.

DOCTYPE=*DOCTYPE-tag*

generates the following DOCTYPE tag by default, which specifies HTML version 3.2:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
```

To use a different DOCTYPE tag, specify the entire contents of the tag as the value of the DOCTYPE= argument, including the angle brackets.

If you specify DOCTYPE="", then no DOCTYPE tag is generated in the HTML output file.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM.

ENCODE=Y | N

when the value is Y (default), replaces the angle bracket characters (< and >) in SAS TITLE and FOOTNOTE lines with the HTML character entities (> and <) respectively. Specifying ENCODE=N causes the browser to interpret the angle brackets as parts of HTML tags. For example, you would use ENCODE=N if you wanted to use the following TITLE statement:

```
title '<FONT COLOR="red">Out of Range Data</FONT>';
```

This argument is supported by the DS2CONST, DS2TREE, and DS2CSF macros.

Used by: DS2TREE, DS2CONST, DS2CSF.

PAGEPART=ALL | HEAD | BODY | FOOT

specifies which part or parts of the HTML page are to be written into the HTML output file. This argument is helpful when are appending HTML output to the end of an existing HTML file, or when you are using separate files for the head, body, and foot of your Web page.

ALL

writes the entire HTML file, including metagraphics codes for the META2HTM macro or the XML tags for the DS2CONST, DS2TREE, and DS2CSF macros. This is the default value. Do not use this value if you are appending an existing HTML file.

HEAD

writes the HTML header information and metagraphics codes (for META2HTM) or XML (for DS2CONST and DS2TREE) into the HTML file. The header information consists of the HEAD and BODY tags. HTML footer information is not included.

BODY

writes only the metagraphics codes (for META2HTM) or XML tags (for DS2CONST, DS2TREE, and DS2CSF) into the HTML output file. No head or foot information is generated in the HTML output file.

FOOT

writes metagraphics codes or XML tags and the </BODY> and </HTML> tags to conclude the HTML file.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM.

SASPOWER=*logo-image-file*

specifies the path or URL, relative or absolute, to the image file of the SAS Powered logo. In the HTML file, the image appears at the bottom of the page. Selecting the image displays the SAS home page. By default, the logo is omitted. To obtain the logo image file, see <http://www2.sas.com/dispatcher/index.html>. See also the SPCLASS= argument.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM.

SEPCLASS=*page-separator-stylesheet*

specifies the path or URL, relative or absolute, to the style sheet that is used for the page separator. If the value of the SEPTYPE= argument is RULE, then the value of the SEPCLASS= argument is used on the CLASS attribute of the HTML tag <HR>. If the value of the SEPTYPE= argument is IMAGE, then the value of SEPCLASS= argument is used on the CLASS attribute of the HTML tag .

Used by: DS2TREE, DS2CONST, DS2CSF.

SEPLOC=*separator-image*

specifies the path or URL, relative or absolute, to the image that you want to use as the separator between the graphs in your presentation. This argument is valid only if the value of the SEPTYPE= argument is IMAGE.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM.

SEPTYPE= IMAGE | NONE | RULE

specifies the type of separator that is used between multiple applets in your presentation. The valid values are defined as follows:

IMAGE

specifies separate graphs using the image specified in the SEPLOC= argument.

NONE

specifies not to use a separator between applets.

RULE

inserts a line between applets. This is the default.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM.

SPCLASS=*logo-stylesheet-name*

specifies the name of the style sheet class that is to be used for the Powered by SAS logo.

Used by: DS2TREE, DS2CONST, DS2CSF.

Arguments for Stylesheets

DS2CONT, DS2TREE, DS2CSF, and META2HTM enable the following arguments for style sheet specifications in the HTML output file. See also the BDCLASS=, SEPCLASS=, and SPCLASS= arguments in “Arguments for Page Formatting” on page 552.

Style sheet arguments reference style information in one of two ways. Most of the arguments specify parameters in the HTML LINK tag:

```
<LINK HREF="lqtr98.css" TYPE="text/css" REL="stylesheet">
```

Use these arguments when you do not want to enter your style information directly into your HTML file when you create that file.

Other arguments embed the style information into the header of the HTML file. Use these arguments when you want to collect style information from multiple style sheets. The end result must create a complete STYLE tag in your HTML file.

You can combine LINK tag arguments with arguments that embed style information, but you cannot use the same ordinal number in two arguments. For example, you can specify the arguments SSHREF1= and SSFILE2=, but you cannot specify SSHREF1= and SSFILE1=.

The following arguments link to two different style sheets and include text comments for each stylesheet.

```
ssfile1=comments1.txt,      /* embeds text          */
sshref2=/style/style1.css, /* links to stylesheet  */
sstype2=text/css,         /* parameters for style sheets */
ssrel2=stylesheet,
ssfile3=comments2.txt,      /* embeds text          */
sshref4=/style/style2.css, /* link to stylesheets  */
sstype4=text/css,
ssrel4=stylesheet,
```

SSFILE1–SSFILE5=*file-specification*

embeds in the HTML file the entire contents of the specified file.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

SSFREF1–SSFREF5=*fileref*

embeds in the HTML file the entire contents of the file that is referenced by the SAS fileref.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

SSHREF1–SSHREF5=*style-sheet-URL*

specifies the URL of the stylesheet in the HREF= attribute of the LINK tag. If you specify a relative URL, it must be relative to the location of the HTML output file.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

SSMEDIA1–5=*media*

specifies the media for which the style sheet was designed. The value is applied to the MEDIA= attribute of the LINK tag. The default value is SCREEN. Examples of other valid MEDIA values include BRAILLE for tactile feedback devices, and HANDHELD for small-screen devices.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

SSREL1–5=*relationship*

specifies the REL= attribute of the LINK tag, which describes the relationship from the linked file to the HTML file. The value of this tag is generally STYLESHEET. The arguments SSREL1–5= can also be used with the arguments SSREV1–5 to link HTML pages in a series. For example, the SSREL1= argument can specify the next document in the series, and the SSREV2= argument can specify the reverse relationship, which would be the previous document in the series. Both arguments, SSRELn= and SSREVn=, can appear in the same LINK tag.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

SSREV1–5=relationship

specifies the REV= attribute of the LINK tag, which describes the relationship from the HTML file to the linked file. See the SSREL1–5= argument for details.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

SSTITLE1–5=title-of-linked-page

specifies the TITLE= attribute of the LINK tag. The TITLE= attribute provides a title for the referenced page. Use this argument when you are using the SSRELn= and SSREVN= arguments to specify next and previous links in a series of Web pages.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

SSTYPE1–5=stylesheet-type

specifies the TYPE= attribute of the LINK tag. For cascading style sheets, this value usually is TEXT/CSS. For JavaScript style sheets, this value is generally TEXT/JAVASCRIPT.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

Arguments for the SAS TITLE and FOOTNOTE Tags

The following arguments determine the content and appearance of the SAS TITLE and FOOTNOTE tags in the HTML output file.

FCLASS=footnote-style-sheet-name**TCLASS=title-style-sheet-name**

specify the name of the style sheet class that is to be used for the SAS TITLE or FOOTNOTE.

Used by: DS2TREE, DS2CONST, DS2CSF.

FCOLOR=footnote-text-color**TCOLOR=title-text-color**

specify the color of the text in the SAS TITLE or FOOTNOTE, using an HTML 3.2 color name or a six-digit hexadecimal RGB value.

Used by: DS2TREE, DS2CONST, DS2CSF.

FFACE=footnote-text-font**TFACE=title-text-font**

specify a text font for the SAS TITLE or FOOTNOTE. Valid values are browser-specific.

Used by: DS2TREE, DS2CONST, DS2CSF.

FSIZE=*n* | +*n* | –*n***TSIZE=*n* | +*n* | –*n***

specify the size of the text font that is to be used for the SAS TITLE or FOOTNOTE, where *n* is an integer. Valid values are browser-specific depending on how the browser handles the SIZE attribute on the FONT tag.

Used by: DS2TREE, DS2CONST, DS2CSF.

FTAG=tag-string**TTAG=tag-string**

specify a text string that the macro translates into one or more tags that will enclose the SAS TITLE or FOOTNOTE.

The default value is as follows:

```
PREFORMATTED + HEADER 3
```

Used by: DS2TREE, DS2CONST, DS2CSF.

For each possible value of the TTAG= and FTAG= arguments, the following table shows the HTML tags that are generated by the macro for the SAS TITLE and FOOTNOTE lines (the corresponding end tags are generated automatically):

TTAG or FTAG Value	HTML Tag or Tags Enclosing the SAS TITLE or SAS FOOTNOTE
NO FORMATTING	(none)
STRONG	
EMPHASIS	
HEADER 1	<H1>
HEADER 2	<H2>
HEADER 3	<H3>
HEADER 4	<H4>
HEADER 5	<H5>
HEADER 6	<H6>
PREFORMATTED TEXT	<PRE>
CITATION TEXT	<CITE>
COMPUTER CODE TEXT	<CODE>
KEYBOARD INPUT TEXT	<KBD>
LITERAL TEXT	<SAMP>
VARIABLE TEXT	<VAR>
BOLD	
ITALICIZED TEXT	<I>
UNDERLINE TEXT	<U>
TYPEWRITER	<TT>
BIG TEXT	<BIG>
SMALL TEXT	<SMALL>
STRIKE OUT TEXT	<STRIKE>
DEFINING INSTANCE TEXT	<DFN>
PREFORMATTED + STRONG	<PRE>
PREFORMATTED + EMPHASIS	<PRE>
PREFORMATTED + HEADER 1	<PRE><H1>
PREFORMATTED + HEADER 2	<PRE><H2>
PREFORMATTED + HEADER 3	<PRE><H3>
PREFORMATTED + HEADER 4	<PRE><H4>
PREFORMATTED + HEADER 5	<PRE><H5>
PREFORMATTED + HEADER 6	<PRE><H6>
PREFORMATTED + CITATION	<PRE><CITE>

TTAG or FTAG Value	HTML Tag or Tags Enclosing the SAS TITLE or SAS FOOTNOTE
PREFORMATTED + COMPUTER CODE	<PRE><CODE>
PREFORMATTED + KEYBOARD INPUT	<PRE><KBD>
PREFORMATTED + LITERAL	<PRE><SAMP>
PREFORMATTED + VARIABLE	<PRE><VAR>
PREFORMATTED + BOLD	<PRE>
PREFORMATTED + ITALICIZED	<PRE><I>
PREFORMATTED + TYPEWRITER	<PRE><TT>
PREFORMATTED + UNDERLINE	<PRE><U>
PREFORMATTED + BIG	<PRE><BIG>
PREFORMATTED + SMALL	<PRE><SMALL>
PREFORMATTED + STRIKE OUT	<PRE><STRIKE>
PREFORMATTED + DEFINING INSTANCE	<PRE><DFN>
STRONG + EMPHASIS	
STRONG + ITALICIZED	<I>
STRONG + CITATION	<CITE>
STRONG + COMPUTER CODE	<CODE>
STRONG + KEYBOARD INPUT	<KBD>
STRONG + LITERAL	<SAMP>
STRONG + VARIABLE	<VAR>
STRONG + TYPEWRITER	<TT>
STRONG + BIG	<BIG>
STRONG + SMALL	<SMALL>
EMPHASIS + CITATION	<CITE>
EMPHASIS + COMPUTER CODE	<CODE>
EMPHASIS + KEYBOARD INPUT	<KBD>
EMPHASIS + LITERAL	<SAMP>
EMPHASIS + VARIABLE	<VAR>
EMPHASIS + TYPEWRITER	<TT>
EMPHASIS + BIG	<BIG>
EMPHASIS + SMALL	<SMALL>
BOLD + EMPHASIS	
BOLD + ITALICIZED	<I>
BOLD + CITATION	<CITE>
BOLD + COMPUTER CODE	<CODE>

TTAG or FTAG Value	HTML Tag or Tags Enclosing the SAS TITLE or SAS FOOTNOTE
BOLD + KEYBOARD INPUT	<KBD>
BOLD + LITERAL	<SAMP>
BOLD + VARIABLE	<VAR>
BOLD + TYPEWRITER	<TT>
BOLD + BIG	<BIG>
BOLD + SMALL	<SMALL>
ITALICIZED + CITATION	<I><CITE>
ITALICIZED + COMPUTER CODE	<I><CODE>
ITALICIZED + KEYBOARD INPUT	<I><KBD>
ITALICIZED + LITERAL	<I><SAMP>
ITALICIZED + VARIABLE	<I><VAR>
ITALICIZED + TYPEWRITER	<I><TT>
ITALICIZED + BIG	<I><BIG>
ITALICIZED + SMALL	<I><SMALL>
STRONG + EMPHASIS + BIG	<BIG>
STRONG + CITATION + BIG	<CITE><BIG>
STRONG + COMPUTER CODE + BIG	<CODE><BIG>
STRONG + KEYBOARD INPUT + BIG	<KBD><BIG>
STRONG + LITERAL + BIG	<SAMP><BIG>
STRONG + VARIABLE + BIG	<VAR><BIG>
STRONG + TYPEWRITER + BIG	<TT><BIG>
EMPHASIS + CITATION + BIG	<CITE><BIG>
EMPHASIS + COMPUTER CODE + BIG	<CODE><BIG>
EMPHASIS + KEYBOARD INPUT + BIG	<KBD><BIG>
EMPHASIS + LITERAL + BIG	<SAMP><BIG>
EMPHASIS + VARIABLE + BIG	<VAR><BIG>
EMPHASIS + TYPEWRITER + BIG	<TT><BIG>
BOLD + EMPHASIS + BIG	<BOLD><BIG>
BOLD + ITALICIZED + BIG	<BOLD><I><BIG>
BOLD + CITATION + BIG	<BOLD><CITE><BIG>
BOLD + COMPUTER CODE + BIG	<BOLD><CODE><BIG>
BOLD + KEYBOARD INPUT + BIG	<BOLD><KBD><BIG>
BOLD + LITERAL + BIG	<BOLD><SAMP><BIG>
BOLD + VARIABLE + BIG	<BOLD><VAR><BIG>
BOLD + TYPEWRITER + BIG	<BOLD><TT><BIG>

TTAG or FTAG Value	HTML Tag or Tags Enclosing the SAS TITLE or SAS FOOTNOTE
ITALICIZED + CITATION + BIG	<I><CITE><BIG>
ITALICIZED + COMPUTER CODE + BIG	<I><CODE><BIG>
ITALICIZED + KEYBOARD INPUT + BIG	<I><KBD><BIG>
ITALICIZED + LITERAL + BIG	<I><SAMP><BIG>
ITALICIZED + VARIABLE + BIG	<I><VAR><BIG>
ITALICIZED + TYPEWRITER + BIG	<I><TT><BIG>
STRONG + EMPHASIS + SMALL	<SMALL>
STRONG + ITALICIZED + SMALL	<I><SMALL>
STRONG + CITATION + SMALL	<CITE><SMALL>
STRONG + COMPUTER CODE + SMALL	<CODE><SMALL>
STRONG + LITERAL + SMALL	<SAMP><SMALL>
STRONG + VARIABLE + SMALL	<VAR><SMALL>
STRONG + TYPEWRITER + SMALL	<TT><SMALL>
EMPHASIS + CITATION + SMALL	<CITE><SMALL>
EMPHASIS + COMPUTER CODE + SMALL	<CODE><SMALL>
EMPHASIS + KEYBOARD INPUT + SMALL	<KBD><SMALL>
EMPHASIS + LITERAL + SMALL	<SAMP><SMALL>
EMPHASIS + TYPEWRITER + SMALL	<TT><SMALL>
BOLD + EMPHASIS + SMALL	<BOLD><SMALL>
BOLD + ITALICIZED + SMALL	<BOLD><I><SMALL>
BOLD + CITATION + SMALL	<BOLD><CITE><SMALL>
BOLD + COMPUTER CODE + SMALL	<BOLD><CODE><SMALL>
BOLD + KEYBOARD INPUT + SMALL	<BOLD><KBD><SMALL>
BOLD + LITERAL + SMALL	<BOLD><SAMP><SMALL>
BOLD + VARIABLE + SMALL	<BOLD><VAR><SMALL>
BOLD + TYPEWRITER + SMALL	<BOLD><TT><SMALL>
ITALICIZED + CITATION + SMALL	<I><CITE><SMALL>
ITALICIZED + COMPUTER CODE + SMALL	<I><CODE><SMALL>
ITALICIZED + KEYBOARD INPUT + SMALL	<I><KBD><SMALL>

TTAG or FTAG Value	HTML Tag or Tags Enclosing the SAS TITLE or SAS FOOTNOTE
ITALICIZED + LITERAL + SMALL	<I><SAMP><SMALL>
ITALICIZED + VARIABLE + SMALL	<I><VAR><SMALL>
ITALICIZED + TYPEWRITER + SMALL	<I><TT><SMALL>

Arguments for Character Transcoding

The following arguments allow you to specify a character set or convert character data to the corresponding Unicode Numeric Character Reference (NCR).

CHARSET=*char-set-name*

specifies the character set name that will be written into the META tag of the HTML output file. For information on available character set names, see <http://www.iana.org/assignments/character-sets>.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

TRANLIST=*transcoding-list-name*

specifies the name and location of an existing transcoding list, either user-defined or from SAS. The transcoding list name must be a four-level name, and the fourth level must be SLIST, as in the following example:

```
TRANLIST=SASHELP.HTMLGEN.IDENTITY.SLIST
```

This argument is required if you are implementing character transcoding.

SAS provides a number of transcoding lists in the SASHELP.HTMLNLS catalog. For a description of these transcoding lists, and for information on generating your own transcoding lists, see the SAS Web site at <http://support.sas.com/rnd/web/intrnet/format/lang2.html>.

Used by: DS2TREE, DS2CONST, DS2CSF, META2HTM

DS2CSF Arguments for Data Definition

These arguments for the DS2CSF macro define how the data set is to be interpreted by the Rangeview Applet as it generates the critical success factor graph.

DATA=*SAS-data-set-name*

specifies the name of the data set that is to be used to generate the graph. The default data set is the one that was most recently created.

Used by: DS2CSF

LABELS=Y | N

when the value is Y (default), indicates that the graph label and critical success value are displayed beneath the gauge. Specifying LABELS=N suppresses the display of the diagram label and critical success value.

Used by: DS2CSF

PW=password

specifies the password that is needed to access a password-protected data set. This argument is required if the data set has a READ or PW password. It is not required if the data set has a WRITE or ALTER password.

Used by: DS2CSF

VAR=variable-name

specifies the name of the variable in the SAS data set that is to be represented in the graph. The indicated value is the value of this variable in the first observation in the data set.

Used by: DS2CSF

DS2CSF Arguments for Diagram Appearance

The following arguments allow you to configure your range view diagram to suit the needs of your Web presentation.

CBACK=color

specifies a background color for the graph. The value must be a valid HTML 3.2 color name or a six-digit hexadecimal RGB value.

Used by: DS2CSF

CHUB=color

specifies a color for the half-circle area in the bottom middle of the classic diagram. The value must be a valid HTML 3.2 color name or a six-digit hexadecimal RGB value.

Used by: DS2CSF

CINDIC=color

specifies the color of the indicator in both the classic and the digital styles of the diagram. The value must be a valid HTML 3.2 color name or a six-digit hexadecimal RGB value.

Used by: DS2CSF

CLABTXT=color

specifies the color of the text label (and value, in the classic graph) that appear in a bold text font. The value must be a valid HTML 3.2 color name or a six-digit hexadecimal RGB value.

Used by: DS2CSF

CLABVAL=color

specifies the color of the data value labels that appear above the gauge. The value must be a valid HTML 3.2 color name or a six-digit hexadecimal RGB value.

Used by: DS2CSF

COUTLINE=color

specifies the color of the outline that is drawn around the classic or digital gauge. The value must be a valid HTML 3.2 color name or a six-digit hexadecimal RGB value.

Used by: DS2CSF

CSFTYPE=CLASSIC | DIGITAL

when the value is CLASSIC (default), specifies that the graph is to be rendered as an analog gauge with a needle that points to the critical success factor. Specifying a value of DIGITAL displays a diagram that resembles a liquid-crystal display.

Used by: DS2CSF

CVALUE=*color*

specifies the color of the critical success value. This value is shown in a bold text font along with the text label value in the classic graph. In the digital graph, the value appears in a digital font. The value must be a valid HTML 3.2 color name or a six-digit hexadecimal RGB value.

Used by: DS2CSF

DEPTH=TWO_DIMENSION | TWO_AND_A_HALF_DIMENSION

when the value is TWO_AND_A_HALF_DIMENSION (default), specifies a diagram that appears to have a degree of depth.

Used by: DS2CSF

DRILURL=*URL*

specifies the URL, relative or absolute, that is to be displayed when the user selects any part of the diagram.

Used by: DS2CSF

DRILTARG=*target-window-or-frame*

specifies the HTML target or the name of the browser window or frame where drill-down URLs are to be displayed. The default behavior is to open a new browser window and reuse it for subsequent drill-down requests. Specifically, the default value is `_BLANK`, which is one of several reserved names for targets in HTML. The DRILTARG value can also be the name of a window or frame in the Web presentation.

Used by: DS2CSF

HINDIC=*indicator-height-percentage*

specifies the height of the indicator for the classic diagram (see the CSFTYPE= argument). Valid values must be greater than zero and less than 100.

Used by: DS2CSF

INDICTYP=*indicator-shape*

specifies the shape of the indicator for the classic diagram (see CSFTYPE). Valid values can be `ARROW`, `HARPOON`, `LINE`, `NEEDLE`, or `SPEAR`.

Used by: DS2CSF

LABELPOS=*text-label-location*

specifies the location of the text label that identifies the critical success factor. Valid values can be `TOP_LEFT`, `TOP_CENTER`, `TOP_RIGHT`, `BOTTOM_LEFT`, `BOTTOM_CENTER`, or `BOTTOM_RIGHT`.

Used by: DS2CSF

RANGE=*range-name*

specifies the four-level name of the RANGE entry that defines range values and colors. For example:

```
range=sashelp.javagrfsample1.range
```

To define a range entry, you can use the BUILD command:

```
build libname.catname.entryname.RANGE
```

This opens a window that you use to define the range. Use the scrollbar on the window to navigate between segments of the range.

Used by: DS2CSF

VALUEPOS=*CSF-value-location*

specifies the location of the critical success value. Valid values can be TOP_LEFT, TOP_CENTER, TOP_RIGHT, BOTTOM_LEFT, BOTTOM_CENTER, or BOTTOM_RIGHT.

Used by: DS2CSF

WINDIC=*indicator-width*

specifies the width of the critical success indicator, in pixels. The default value is 8.

Used by: DS2CSF

META2HTM Arguments for Saving the HTML File

The META2HTM macro enables the following arguments for generating and saving an HTML file.

Note: All of the following macro arguments are required. \triangle

CAPTURE=ON | OFF

CAPTURE=ON enables the capture of metagraphics output. Set CAPTURE=ON above the SAS/GRAPH procedure that will generate metagraphics for the HTML output file. Set CAPTURE=OFF after you run and quit the SAS/GRAPH procedure.

The following arguments are enabled when CAPTURE=ON or when CAPTURE=OFF: CENTER=, HTMLFILE=, HTMLFREF=, OPENMODE=, and PAGEPART=.

The following arguments are enabled only when CAPTURE=OFF: SASPOWER=, SEPLOC=, SEPCLASS=, and SPCLASS=.

Any other arguments are valid only when CAPTURE=ON.

This argument is required.

Used by: META2HTM

HTMLFILE=*external-filename*

specifies the name and path of the HTML file where the output will be written. If the file you specify does not exist, it is created for you.

This argument is *required* if HTMLFREF is omitted. HTMLFILE= and HTMLFREF= are mutually exclusive arguments.

Used by: META2HTM

HTMLFREF=*fileref*

specifies the fileref that points to the location of the HTML file where the output will be written. If the file you specify does not exist, then it is created for you. Do not use a reserved name. See “Reserved Names” on page 566 for more information.

This argument is *required* if HTMLFILE is omitted. HTMLFREF= and HTMLFILE= are mutually exclusive arguments.

Used by: META2HTM

OPENMODE=APPEND | REPLACE

indicates whether the new HTML output overwrites the information currently in the specified file or if the new output is appended to the end of the existing file. The default value is REPLACE. If you do not want to replace the current contents, then supply APPEND to add your output to the end of an existing HTML file.

Always use OPENMODE=APPEND with the CAPTURE=OFF argument.

OPENMODE=APPEND is not valid if you are writing in the z/OS operating environment.

This argument is required.

Used by: META2HTM

RUNMODE=B | S

specifies whether you are running the macro in batch or server mode. Batch mode (the default) means that you are submitting the META2HTM macro in the Program Editor or you have included it in a SAS program. Server mode is used with Dispatcher Applications in the SAS/INTRNET software. Specifying server mode generates an HTTP header that is required by Application Dispatcher.

This argument is required.

Used by: META2HTM

META2HTM Arguments for Applet Behavior

The META2HTM macro enables the following arguments, which are used to control the behavior of the applet. Many of these arguments specify applet parameters.

CBACK=*hex-color*

specifies the color of the applet control area, using a six-digit hexadecimal RGB value.

Used by: META2HTM

CTIPHILT=*hex-color*

specifies a six-digit hexadecimal RGB value for the color of the graph element that has been selected to display its pop-up data tips window. The default highlight color is red. See also the TIPTYPE= argument.

Used by: META2HTM

PAGECTL=Y | N

when the value is N (default), prevents the display of the page-selection control. For further information, see “Metaview Applet Parameters” on page 475.

Used by: META2HTM

SLIDECTL=Y | N

the default value of N prevents the display of the graph-selection control. For further information, see “Metaview Applet Parameters” on page 475.

Used by: META2HTM

TIPTYPE=HIGHLIGHT | STICK | FIXED_STICK

specifies the appearance of data tips.

HIGHLIGHT

specifies that the text tip is to appear as a disconnected pop-up window. The outline of the selected graph element is highlighted in the color specified in the CTIPHILT= argument. This is the default value.

STICK

specifies that a line will connect the cursor and the selected graph element. The pop-up text tip window appears above the cursor and follows the cursor as it moves within the selected graph element.

FIXED_STICK

specifies that a line will be drawn between the text tip pop-up window and the center of the selected graph element. The text tip window does not move with the cursor.

See also “Metaview Applet Parameters” on page 475.

Used by: META2HTM

ZOOMCTL=Y | N

specifies whether or not the zoom control appears in the applet control area. By default, the control is displayed.

Used by: META2HTM

Reserved Names

Do not use the following names as the value of a macro variable:

Libnames and Filerefs

HTML

CATENT

HTMSS

Global Macro Variables

_htmvp

_htmcap

_htmtitl

_htmwher

Data Sets or Views

WORK._BYGRP

Catalogs

WORK._HTMLG_

SASHELP.HTMLNLS

Catalog Entries

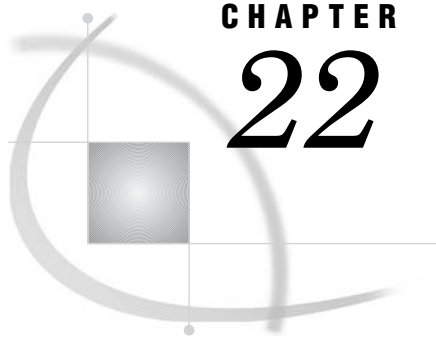
SASHELP.HTMLGEN.DSPROP.SLIST

SASHELP.HTMLGEN.IDENTITY.SLIST

SASHELP.HTMLGEN.OUTPROP.SLIST

SASHELP.HTMLGEN.TABPROP.SLIST

SASHELP.HTMLGEN.TAGS.SLIST



CHAPTER 22

Enhancing Web Output

<i>Enhancing Web Output</i>	567
<i>Adding Data Tips to Web Presentations</i>	568
<i>Data Tips in GIF, JPEG, and PNG Files</i>	568
<i>Data Tips in ACTXIMG and JAVAIMG Images</i>	568
<i>Data Tips in Java and ActiveX Web Presentations</i>	568
<i>Data Tips in Metaview Applet Presentations</i>	569
<i>Data Tips in Treeview Diagrams</i>	569
<i>Data Tips in Constellation Charts</i>	569
<i>Data Tips in Critical Success Factor Diagrams</i>	569
<i>Data Tips in Animated GIFs</i>	570
<i>Adding Data Tips with the HTML= Option</i>	570
<i>Adding Drill-Down Links to Web Presentations</i>	571
<i>Links in GIF, JPEG, and PNG Files</i>	571
<i>Links in ACTXIMG and JAVAIMG Images</i>	571
<i>Links in Java and ActiveX Web Presentations</i>	571
<i>Links in Metaview Applet Presentations</i>	572
<i>Links in Treeview Diagrams</i>	572
<i>Links in Constellation Charts</i>	573
<i>Links in Critical Success Factor Diagrams</i>	573
<i>Links in Animated GIFs</i>	573
<i>Adding Links with the HTML= and HTML_LEGEND= Options</i>	574
<i>Working with Link and Enhancement Variables</i>	574
<i>Assigning Values to Link and Enhancement Variables</i>	575

Enhancing Web Output

When you enhance a Web presentation, you specify additional options, arguments, or parameters to configure and add to the Web presentation that is generated by default. The number and type of enhancements that are available depend on the type of the Web presentation. Presentations that run with Web executables (ActiveX Control or Java applets) enable the largest number of enhancements. Depending upon the type of Web presentation, enhancements include:

- Using ODS styles to enhance the appearance of graphical output. See “Using ODS Styles” on page 488.
- Displaying pop-up text when the cursor is over a portion of the diagram. See “Adding Data Tips to Web Presentations” on page 568.
- Adding hotspots that link to other Web pages. See “Adding Drill-Down Links to Web Presentations” on page 571.

- Adding annotations to Web presentations. See Chapter 17, “Generating Web Output with the Annotate Facility,” on page 499.
- Modifying colors in the presentations. See “Specifying Colors in SAS/GRAPH Programs” on page 92.
- Displaying images as part of a graph. See “Specifying Images in SAS/GRAPH Programs” on page 106.
- Changing text fonts. See “Specifying Fonts in SAS/GRAPH Programs” on page 75.

Adding Data Tips to Web Presentations

You can add pop-up data tips to most SAS/GRAPH Web presentations. The text is displayed when the user’s cursor is over a specified area of a graph. See the following sections for basic information on adding data tips to Web presentations:

- “Data Tips in GIF, JPEG, and PNG Files” on page 568
- “Data Tips in ACTXIMG and JAVAIMG Images” on page 568
- “Data Tips in Java and ActiveX Web Presentations” on page 568
- “Data Tips in Metaview Applet Presentations” on page 569
- “Data Tips in Treeview Diagrams” on page 569
- “Data Tips in Constellation Charts” on page 569
- “Data Tips in Critical Success Factor Diagrams” on page 569
- “Data Tips in Animated GIFs” on page 570.

Data Tips in GIF, JPEG, and PNG Files

For Web output that is generated with ODS and the GIF, JPEG, and PNG device drivers, SAS adds default data tips using the values of fields in the SAS data set. Also, if you specify `DESCRIPTION=` as an option on the SAS/GRAPH procedure, then SAS adds the text of that description as a data tip for the entire graphic. Specify `DESCRIPTION=""` to suppress this default data tip.

You can also add custom data tips to the output of any SAS/GRAPH procedure that supports the `HTML=` option. For this technique, see “Adding Data Tips with the `HTML=` Option” on page 570.

Data Tips in ACTXIMG and JAVAIMG Images

When you specify `DEVICE=ACTXIMG` or `JAVAIMG`, and use `ODS HTML`, SAS adds default data tips using the values of fields in the SAS data set. Also, if you specify `DESCRIPTION=` as an option on the SAS/GRAPH procedure, then SAS adds the text of that description as a data tip for the entire graphic. Specify `DESCRIPTION=""` to suppress this default data tip.

By using the `HTML=` option of a `GBARLINE`, `GCHART`, `GPLOT` (except for `high-low`), or `GRADAR` procedure, you can add custom data tips to graphs created with the `ACTXIMG` device driver. For information, see “Adding Data Tips with the `HTML=` Option” on page 570. SAS/GRAPH does not directly support adding custom data tips to Web presentations created with `DEVICE=JAVAIMG`. You can, however, use any image-map tool available to you to create an image map for the resulting PNG file.

Data Tips in Java and ActiveX Web Presentations

By default, Web presentations created by the `ACTIVEEX` control or `JAVA` applets automatically include data tips using the values of fields in the SAS data set. Use the

TIPS=NONE parameter to suppress data tips. See “Parameter Reference for Java and ActiveX” on page 424.

You can provide your own custom data tips to the output of any SAS/GRAPH procedure that supports the HTML= option. (This feature is supported by the Map applet but is not currently supported by the Graph applet.) For information on this technique, see “Adding Data Tips with the HTML= Option” on page 570.

Note: When you provide data-tip text using HTML=‘ALT=“*variable_name*”’ the Java applets automatically suppress the display of the default data tips. The ActiveX Control, however, adds your custom text to the default text. To suppress the default text in ActiveX presentations, use the TipMode=HTML parameter in an ODS statement. See “Parameter Reference for Java and ActiveX” on page 424 .

For example:

```
ODS HTML parameters=( "TipMode"="HTML" )
```

\triangle

Data Tips in Metaview Applet Presentations

For graphs displayed by the Metaview applet, you can add data tips to the output of any SAS/GRAPH procedure that supports the HTML= option. For more information, see “Adding Data Tips with the HTML= Option” on page 570.

Data Tips in Treeview Diagrams

You can add data tip text to a Treeview diagram by including the text in the SAS data set from which the Treeview is generated. For example, the following data set specifies a different data-tip for each observation:

```
data father_and_sons;
input id $8. name $15. father $8. datatip $30.;
cards;
aaron   Aaron Parker           Data tip for Aaron Parker...
bob     Bob Parker             aaron   Data tip for Bob Parker...
charlie Charlie Parker aaron   Data tip for Charlie Parker...
david   David Parker          aaron   Data tip for David Parker...
edward  Edward Parker         david   Data tip for Edward Parker...
;
run;
```

Use the TIPS, NTIP, and NTIPFMT parameters of DS2TREE (see “DS2TREE and DS2CONST Arguments for Data Definition” on page 537) to control data tip text.

Data Tips in Constellation Charts

You can add data tips to a Constellation diagram by including the text in the SAS data set from which the diagram is generated, as is done for Treeview diagrams as shown in “Data Tips in Treeview Diagrams” on page 569. Use the TIPS, NTIP, and NTIPFMT parameters of DS2CONT (see “DS2TREE and DS2CONST Arguments for Data Definition” on page 537) to control data-tip text.

Data Tips in Critical Success Factor Diagrams

The Critical Success Factor macro DS2CSF does not support adding data tips.

Data Tips in Animated GIFs

SAS/GRAPH does not directly support inserting data tips into animated GIFs.

Adding Data Tips with the HTML= Option

You can add custom data tips to the output of any SAS/GRAPH procedure that supports the HTML= option. Use the HTML option in the following form:

```
HTML='ALT="variable_name"'
```

The following code fragment illustrates the basic technique of adding data tips with the HTML= option.

```
/* initialize a data tip variable in the data set */
length rpt $40;

/* assign values to the link variable */
if Region='Central' then
    rpt='alt="Central region"';
else if Region='South' then
    rpt='alt="Southern region"';
else if Region='West' then
    rpt='alt="Western region"';

/* create a chart that uses the data tip variable */
proc gchart data=regsales;
    vbar3d region / sumvar=sales
    patternid=midpoint
    html=rpt;
run;
```

In this case, the HTML= option identifies the variable RPT as containing ALT= plus the text of data tip to be displayed. The value of RPT, in turn, is set by an IF statement according to the contents of a SAS data set. The maximum length for the value of the variable is 1024 characters, including the characters "ALT=".

The following code uses a variable inside the HTML= variable to substitute the appropriate text for a data tip. It also uses the concatenation operator || to concatenate the string "region" to the name of the region, and the newline character 'OD'x to create a second output line.

```
/* initialize a data tip variable in the data set */
length rpt $40;

/* assign values to the link variable */
rpt='ALT="Region: ' || trim(left(Region)) || 'OD'x ||
    'Second line of text ' || '"';
/* create a chart that uses the data tip variable */
proc gchart data=regsales;
    vbar3d region / sumvar=sales
    patternid=midpoint
    html=rpt;
run;
```


For more on the HTML option, see “Adding Links with the HTML= and HTML_LEGEND= Options” on page 574.

Adding Drill-Down Links to Web Presentations

You can add hotspots to most SAS/GRAPH Web presentations. The hotspots enable a user to select an element of a graph to open a Web page showing either another graph or related information. See the following sections for basic information on adding links to Web presentations:

- “Links in GIF, JPEG, and PNG Files” on page 571
- “Links in ACTXIMG and JAVAIMG Images” on page 571
- “Links in Java and ActiveX Web Presentations” on page 571
- “Links in Metaview Applet Presentations” on page 572
- “Links in Treeview Diagrams” on page 572
- “Links in Constellation Charts” on page 573
- “Links in Critical Success Factor Diagrams” on page 573
- “Links in Animated GIFs” on page 573.

Links in GIF, JPEG, and PNG Files

To add a hotspot link to static images generated with the GIF, JPEG, and PNG device drivers, use the HTML= option or HTML_LEGEND= option or both, with a SAS/GRAPH procedure as described in “Adding Links with the HTML= and HTML_LEGEND= Options” on page 574. The links are implemented in an HTML image map. The elements in the graph, such as bars or pie slices, become selectable hotspots in the Web presentation. In your SAS program, drill-down URLs are defined in a data set variable. SAS generates the image map for you in the HTML output file.

A complete example of hotspots created in this way is shown in “GIF Output with Hotspot Links” on page 452.

Links in ACTXIMG and JAVAIMG Images

To add a hotspot link to an image created with the ACTXIMG device driver and a GBARLINE, GCHART, GPLOT (except for high-low), or GRADAR procedure, use the HTML= option as described in “Adding Links with the HTML= and HTML_LEGEND= Options” on page 574.

SAS/GRAPH does not directly support hotspots for Web presentations created with DEVICE=JAVAIMG. You can, however, create an image map for the resulting PNG file using any image-map tool available to you.

Links in Java and ActiveX Web Presentations

The ActiveX Control and two of the SAS Java applets (Graph and Map) support four implementations of links in Web presentations. Hotspots are not supported for contour diagrams in either the ActiveX Control or the Contour applet.

To specify the drill-down mode, use the DRILLDOWNMODE parameter as follows (for more information, see “Configuring Drill-Down Links with ACTIVEX” on page 392):

```
ODS HTML PARAMETERS=
  ("DRILLDOWNMODE"="LOCAL"|"SCRIPT"|"URL"|"HTML");
```

Local mode (Graph applet only)

responds to drill-down actions by dynamically generating and displaying new graphs based on a subset of the data in the selected graph element. In this mode, the graphic to be displayed is generated by the applet from data it already has (rather than being a pre-existing HTML page that you have created yourself). At each drill-down level, the user can configure the graph type, data subset, variable roles, and colors.

Script mode

calls a JavaScript method that you specify in your SAS/GRAPH program, and passes to that method information on the selected graph element or map region. It is up to you to write the JavaScript to respond to the mouse-click. You can use the data passed to the JavaScript function to determine what part of the diagram was clicked on and, therefore, what URL is appropriate to link to. This is the default drill-down mode for the Map applet.

URL mode

displays URLs that are provided by link variables. The link variables are identified to the graphics procedure with the HTML= option. The URLs identify pre-existing HTML files that you will have created yourself. The drill-down functionality of the URL mode is similar to the drill-down functionality that is provided by the GIF, JPEG, and PNG device drivers.

HTML mode

generates drill-down URLs based on a substitution pattern that you specify in your SAS/GRAPH program. The Graph applet and Map applet complete the URL by inserting the specified data from the graph element that was selected in the drill-down action. An example link specified in HTML mode is the following:

```
ods html file=statepop.htm
  parameters=( "DRILLDOWNMODE"="HTML"
    "DRILLPATTERN"='http://www.state.{&statename}.us' );
```

In this example, the value of the data set variable STATENAME completes the drill-down URL.

Note: The variable must be used in the chart. It is not sufficient that it simply be in the data set. △

Any mode (Graph applet and ActiveX control)

attempts to implement each of the four drill-down modes in succession until a valid Web destination is found. The order of the attempts is Local (Graph applet only), Script, URL, and HTML.

Links in Metaview Applet Presentations

To generate drill-down presentations for the Metaview applet, use either the HTML= or HTML_LEGEND= options or both and an enhancement variable, as introduced in “Adding Links with the HTML= and HTML_LEGEND= Options” on page 574.

Links in Treeview Diagrams

You can add hotspots to Treeview diagrams so that when a user selects a node another Web page is opened. The easiest way to add hotspots is to include the URL to

be linked to in the SAS data set from which the Treeview is generated. For example, the following data set specifies a different URL for each observation:

```
data father_and_sons;
input id $8. name $15. father $8. url $30.;
cards;
aaron  Aaron Parker          http://www.yourdomain.com/aaronparker.html
bob    Bob Parker      aaron  http://www.yourdomain.com/bobparker.html
charlie Charlie Parker aaron  http://www.yourdomain.com/charlieparker.html
david  David Parker    aaron  http://www.yourdomain.com/davidparker.html
edward Edward Parker  david  http://www.yourdomain.com/edwardparker.html
;
run;
```

A simple, but complete example is shown in “Treeview with Hotspots” on page 510.

Links in Constellation Charts

You can add hotspots to Constellation diagrams so that when a user selects a node another Web page is opened. The easiest way to add hotspots is to include the URL to be linked to in the SAS data set from which the diagram is generated, as is done for Treeview diagrams as shown in “Links in Treeview Diagrams” on page 572. A simple example of a Constellation diagram with hotspots is shown in “Constellation Chart with Hotspots” on page 524.

Links in Critical Success Factor Diagrams

You can add a hotspot to a critical success factor diagram created with the DS2CSF macro. However, unlike the Treeview and Constellation macros, you can add only a single hotspot to the diagram, i.e., the diagram as a whole. The following code fragment links to the specified URL when a user clicks anywhere on the diagram. The DRILURL= option specifies the URL to link to, while DRILTARG=_self specifies that the new Web page is to be displayed in the same window as the dial.

```
%ds2csf(data=test,
        var=x,
        htmlfile=u:/public_html/Web_output/csf.html,
        openmode=replace, pagepart=head,
        center=y,
        drilurl=http://www.sas.com,
        archive=rvapplet.jar,
        /* specify the complete url for the jar file          */
        /* if it is not in same directory as the html file   */
        /* archive=http://sww.sas.com/avd/codebase/rvapplet.jar, */
        csftyp=classic, septype=none, cback=#e0e0e0,
        bgtype=color, bg="#e0e0e0",
        ttag=bold + italicized, tcolor="#002288", tsize=5,
        tface="Arial, Helvetica",
        range=sashelp.javagrfsample1.range)
```

Links in Animated GIFs

SAS/GRAPH does not directly support inserting hotspots into animated GIFs. If you want to enable linking from an animated GIF, you must use whatever third-party tools

are available to you. You can also make the entire image a hotspot by including the tag inside an tag.

Adding Links with the HTML= and HTML_LEGEND= Options

The HTML= and HTML_LEGEND= options can be used in a number of statements that generate graphs. These options are used to add drill-down links to Web presentations that are generated with the following device drivers:

- GIF, JPEG, or PNG
- JAVA and ACTIVEX
- JAVAMETA

In these Web presentations, the HTML= and HTML_LEGEND= options identify a variable that provides drill-down URLs. This variable is referred to as a *link variable* because of its use in establishing links.

The HTML= and HTML_LEGEND= options are also used to implement a number of different enhancements to Web presentations that run in the Metaview applet. In this case, the variables that are identified by the HTML= and HTML_LEGEND options are referred to as *enhancement variables* because of their broader use than just establishing links.

Working with Link and Enhancement Variables

To use link or enhancement variables in a Web presentation, you need to define those variables, add data to those variables, and then identify those variables in the HTML= option or HTML_LEGEND option or both.

The following code fragment defines a link variable named RPT and assigns that variable a length of 40 characters:

```
data regsales;
  input Region State Sales;
  length rpt $40
```

Be sure to define your link variable with a length that will be sufficient to contain your URLs (plus the HREF= option). The maximum length is 1024 characters.

The values of the link variable use the following syntax:

```
'HREF=URL<"anchor-name">'
```

This syntax is used in the following example:

```
RPT='href="reports.html#west" ';
```

The following table lists the valid values of the link variable:

Table 22.1 Valid Values of the Link Variable

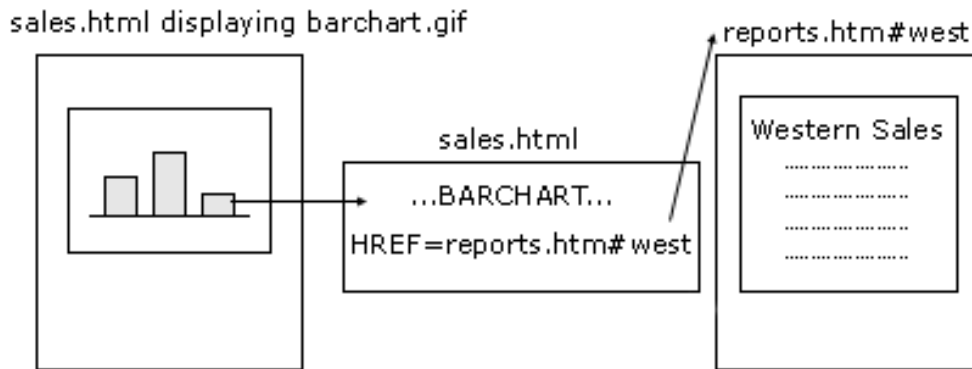
Value Assigned to a Link Destination Variable	Implications
' <code>HREF=<path>reports.html</code> '"	Tells the browser to look in the file reports.html. If <code><path></code> is not provided, the file must be in the same location as the HTML file that initiates the link.
' <code>HREF=<path>reports.html#west</code> '"	For the value that has #west, go to the output with the anchor name west. Users must have access to your file system in order to access the link target.
' <code>HREF="http://www.company.com/web/reports.html"</code> '"	Tells the browser to go to the Web site address http://www.company.com/web and look in the file reports.html.
' <code>HREF="http://www.company.com/web/reports.html#west"</code> '"	For the value that has #west, go to the output with the anchor name west. Users must have access to the Web to follow the links.
' <code>HREF="#west"</code> '"	Go to the target with the anchor name west. The target must be referenced or contained in the same HTML file as the drill-down graph that contains the link. For example, if the drill-down graph is in the file sales.html, then the target output must be referenced or contained in sales.html. Because this target is in the same file as the drill-down graph, this link will work whether the output is viewed within your file system or across the Web.

Assigning Values to Link and Enhancement Variables

The most obvious method of adding these variables to your data set is to manually add them to the desired observations in your data set. This method is not practical or feasible in many cases, in which case you can use IF/THEN statements or variable substitution.

The following picture shows how link variables are assigned to a bar chart. The three bars represent regional sales for a company's central, southern, and western regions.

Figure 22.1 Links in Drill-Down Graphs



Each bar in the chart needs to link to a different anchor tag in an HTML file named reports.html. The anchor names in the linked file are “central,” “south,” and “west.” The following DATA step uses an IF/THEN statement to assign values to the link variable.

```

/* create data set REGSALES */
data regsales;
  length Region $ 8;
  format Sales dollar8.;
  input Region State Sales;
  length rpt $40; /* the link dest. variable */

/* assign HREF values to link dest. variable */
if Region='Central' then
  rpt='HREF="reports.html#central"';
else if Region='South' then
  rpt='HREF="reports.html#south"';
else if Region='West' then
  rpt='HREF="reports.html#west"';

/* create a chart that uses the data tip variable */
proc gchart data=regsales;
  vbar3d region / sumvar=sales
  patternid=midpoint
  html=rpt;
run;
  datalines;
West CA 13636
West OR 18988
West WA 14523
Central IL 18038
Central IN 13611
Central OH 11084
Central MI 19660
South FL 14541
South GA 19022
;

```

The following table shows the values in the data set REGSALES.

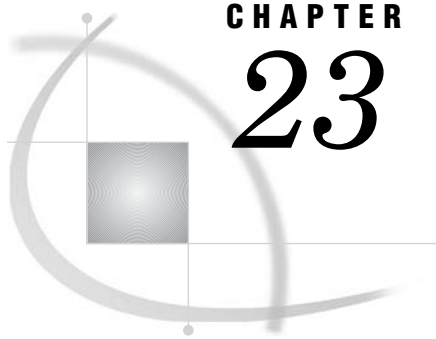
Display 22.1 Values in the REGSALES Data Set

Region	State	Sales	rpt
West	CA	\$13,636	href="reports.html#west"
West	OR	\$18,988	href="reports.html#west"
West	WA	\$14,523	href="reports.html#west"
Central	IL	\$18,038	href="reports.html#central"
Central	IN	\$13,611	href="reports.html#central"
Central	OH	\$11,084	href="reports.html#central"
Central	MI	\$19,660	href="reports.html#central"
South	FL	\$14,541	href="reports.html#south"
South	GA	\$19,022	href="reports.html#south"

To illustrate the use of variable substitution, assume that you are developing for the Metaview applet a presentation that uses the HREF drill-down mode of linking. You need to add an enhancement variable to each observation in the data set. The value of that variable is a URL that has a common base and a filename that is derived from a variable in the data set. The following example defines the base URL, defines an enhancement variable, and uses variable substitution to assign values to the enhancement variable.

```
%let htmlpath=http://webServer1/graph/javameta;
data yearonly;
  set prdsummary(where=( _type_ eq 2 ));
  length htmlvar $200;
  htmlvar='href='||quote("&htmlpath/y"||put(year,4.)||'.htm');
run;
```

In the preceding example, the values of the enhancement variable HTMLVAR are URLs. Each URL combines the base address in HTMLPATH with the value of the YEAR variable and a filetype of HTM.



CHAPTER

23

Troubleshooting Web Output

<i>Troubleshooting Web Output</i>	579
<i>Checking Browser Permissions</i>	582
<i>Using HTML Character Entities</i>	582
<i>Connecting to Web Servers that Require Authentication</i>	583
<i>Removing CLASSPATH Environment Variables</i>	583
<i>Correcting Text Fonts</i>	583
<i>Resolving Colors in Netscape</i>	583
<i>Resolving Differences Between Client and Server Graphs</i>	584

Troubleshooting Web Output

This chapter contains information that you can use to resolve rendering problems on client workstations.

If you or a member of your audience cannot display your presentation, then refer to the following table for solutions.

NOTE: to ensure that software requirements have been met, see “What does your audience need to view the presentation?” on page 380.

Table 23.1 Web Troubleshooting

Symptom	Cause	Remedy
Can't access the HTML file.	Incorrect URL.	Check the URL in the browser.
	Network access denied.	Check operating environment permissions for the HTML file. Check firewall access permissions for Internet clients.
Browser can't display the file.	Browser or Java plug—in may not meet requirements.	Check the requirements. See “What does your audience need to view the presentation?” on page 380.

	ActiveX control may not have been installed or may be out of date.	Install the ActiveX control manually (see “Manually Installing the ActiveX Control” on page 389). Consider updating the presentation to prompt users to install the control (see “Prompting for Installation of the ActiveX Control” on page 390).
	User attempting to run the ActiveX control in a browser other than Internet Explorer.	Switch to the required version of the Internet Explorer Web browser.
	User has not been authenticated for that browser and that Web page.	Check to see if authentication is needed, and then authenticate. See “Connecting to Web Servers that Require Authentication” on page 583.
	Browser doesn’t recognize the file as HTML.	Ensure that the type of the HTML file is correctly specified. Ensure that the DOCTYPE and MIME tags are correctly formatted.
	Browser permissions too restrictive.	Check browser permissions. See “Checking Browser Permissions” on page 582.
Browser displays blank page.	Browser cannot access the referenced image file.	If <i>not</i> running an applet or control, check the image file at the location specified in the HTML file.
	Browser cannot run the applet or control.	For Java, ensure that the HTML file is correctly referencing the Java plug-in and SAS Java archive. See “Specifying the Location of Control and Applet Files (CODEBASE= and ARCHIVE= Options)” on page 422. Check browser permissions for running Java scripts. See “Checking Browser Permissions” on page 582. In the UNIX operating environment, remove any CLASSPATH environment variables. See “Removing CLASSPATH Environment Variables” on page 583. Open the browser’s Java Console and trace the source of the error.

Graph is distorted.	NOGTITLE or NOGFOOTNOTE option on the ODS statement is not properly reclaiming space.	Use the HPOS= or ASPECT= GOPTION to restore the correct aspect. Use hardware fonts.
Browser displays popup message Error: Not enough virtual memory to produce plot.	Client RAM is insufficient for rendering.	Generate a new graph using a smaller data set or a simpler graph. If using PROC GMAP, consider using PROC GREDUCE.
Graph is not rendering as specified by the ODS graph style.	A style attribute may not be enabled for your ODS destination.	Ensure that the attribute is enabled for your ODS destination. For example, the URL attribute is not enabled for the PS destination. Refer to the table of style attributes for the STYLE statement of the TEMPLATE procedure in <i>SAS Output Delivery System: User's Guide</i> .
	A style attribute may be overridden by a global option, global statement option, procedure option, or statement option.	Specify the minimum options needed for your graph, for example: options reset=all device=activex;
In ActiveX, the user gets the message There is a pending reboot for this machine...	<ol style="list-style-type: none"> 1 Virus-scanning software may be interfering with the installation of the control. 2 Other instances of the control might be running. 	<ol style="list-style-type: none"> 1 Turn off any virus-scanning software before installing the control. 2 Be sure to close all instances of Internet Explorer before installing the control.
Text font is incorrect.	Java font is defined differently.	Change browser fonts or change the SAS/GRAPH program. See "Correcting Text Fonts" on page 583.
Text in browser shows incorrect characters.	Browser misinterpreting special characters.	Replace special characters with character entities. See "Using HTML Character Entities" on page 582.
Graph in browser differs from graph in SAS.	A graphics option or global statement may be unsupported or partially supported for that applet or control. See also "Resolving Differences Between Client and Server Graphs" on page 584.	Refer to the descriptions for the options you are using and to Appendix 1, "Summary of ActiveX and Java Support," on page 1507 for information on whether a statement or option is supported.

	A default value in the applet or control is overriding a default option value.	Specify a value for the option rather than relying on the default. See “Resolving Differences Between Client and Server Graphs” on page 584.
	GPLOT lines drawn in reverse order on the client.	This change was made intentionally to maintain the integrity of plots drawn with the AREAS= option.
In ActiveX, black-and-white image is not displayed	ActiveX does not enable 8-bit grayscales images.	Convert the image to 24-bit monochrome.
Graph loses attributes after graph type is changed in the Web browser.	Some attribute loss is inherent in graph type changes.	Select the Refresh button in the Web browser to restore the original graph.
Changes made through the Data Options dialog cause the graph to revert to its original view.	The graph discards subsetting information if you make changes through the Data Options dialog.	Make any changes needed through the Data Options dialog before subsetting the graph.
Colors wrong in Netscape.	Netscape is running without its own color map.	Run Netscape with the install option. See “Resolving Colors in Netscape” on page 583.

Checking Browser Permissions

Access permissions vary from browser to browser, but some form of access control is enforced in most browsers. To check your permissions, open the browser’s Preferences or Internet Options window. Then look for the advanced options. Use your browser’s help system and contact your system support representative as needed to ensure that the browser permissions allow the following:

- Stylesheets
- Java
- JavaScripts
- Java Console

In the Security tab of the Internet Explorer’s Internet Options window, make sure that the selected Web content zone enables access to the Web presentation.

Using HTML Character Entities

If a special character in your Web presentation does not resolve in the browser, that character may need to be changed to a character entity in the source file or in the SAS program. A character entity is a standardized string of characters that represents a special character. The browser recognizes the string and replaces it with the special character when it is formatting the display. One common character entity is `>`. This entity represents the greater-than symbol (`<`).

Lists of standard character entities are provided in HTML reference books and in HTML references on the Worldwide Web.

For presentations that run in the Constellation, Treeview, and Rangeview applets, the macros DS2CONST, DS2TREE, and DS2CSF enable the ENCODE argument, which you can use to automatically replace or not replace angle brackets (“<” and “>”) in TITLE and FOOTNOTE statements.

Connecting to Web Servers that Require Authentication

If you are unable to run a Java applet or install the ActiveX control, then you may be trying to access a Web server that requires authentication. To resolve this problem, access a different file on that server and enter your user ID and password. Redisplaying your Web presentation should now allow you to access that Web server.

Removing CLASSPATH Environment Variables

In the UNIX operating environment, if the Java applet does not run after you have verified that your Java archive is correctly specified, then you should remove any CLASSPATH environment variables that have been set. The Java archive files contain all the required classes to run the applets. Your CLASSPATH may point to old versions of the required classes (for example, for use with the webAF software). This can cause the applets to fail to load. Most applications allow you to specify a CLASSPATH at startup, by using a startup option. This is often safer for running multiple clients than using the environment variables.

Correcting Text Fonts

If your presentation displays an incorrect text font on a given client computer, then the cause may be that the client computer maps a Java logical font name such as Courier to a different physical font set. If the logical font is not mapped to any physical font, Java uses a default font. To correct the problem, edit your HTML file or your SAS/GRAPH program to specify a different logical font name.

For programs that use the JAVA or JAVAMETA device drivers, or that use the macros DS2CONST, DS2TREE, or DS2CSF, specify one of these logical font names: Courier, Dialog, DialogInput, Helvetica, Monospaced, Serif, SansSerif, or TimesRoman. These names are case-sensitive. If you specified them in uppercase letters in your SAS program, this could be the source of your problem.

The JAVA device driver can use the physical font set Lucida, which is provided in the Java plug-in. If your Java presentation shows unexpected fonts, switching to the Lucida font should clear up the problem. For further information on font specifications, see “Specifying Fonts in SAS/GRAPH Programs” on page 75.

Resolving Colors in Netscape

In the UNIX operating environment, the Netscape browser can run out of available screen colors and begin using default colors such as black. To alleviate this problem, run the Netscape program with the INSTALL option, as follows:

```
netscape -install
```

Running the Netscape program with the INSTALL option generates a separate color map for that browser.

Resolving Differences Between Client and Server Graphs

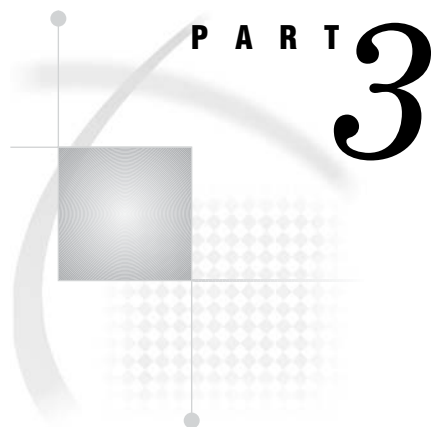
A *client* graph is rendered on the client's (recipient's) system using data sent from the server. The user may be able to manipulate and interact with the graph. Client graphs are intended for display on the Web. Graphs generated with the JAVA and ACTIVEX device drivers, for example, are client graphs.

A *server* graph is rendered on the server (the machine where the SAS session is running). Interaction with server graphs is limited to features defined by the server such as drill-down links and pop-up data tips. A server graph may or may not be intended for display on the web. Graphs generated with the GIF, JPEG, and PNG device drivers, for example, are server graphs.

Because of technological differences between SAS, Java, and ActiveX, client graphs may differ from server graphs even if the graphs are generated with the same SAS procedure code. In addition, graphs generated with Java may differ from graphs generated with ActiveX. The graphs may differ in appearance, in the default values used for certain options, or in the availability of certain features.

For example, differences between client and server graphs may occur if you are using a global statement or procedure option that is not enabled for that applet or control. Most global statement and procedure options are fully supported by the client device drivers. Exceptions are identified in the procedure and statement documentation and summarized in Appendix 1, "Summary of ActiveX and Java Support," on page 1507.

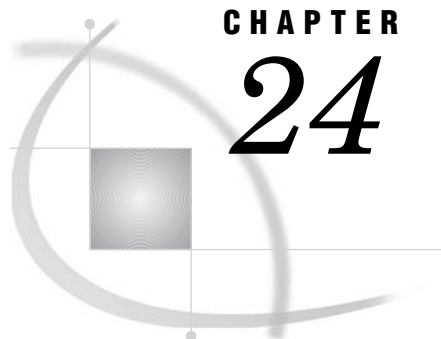
In certain cases, differences between client and server graphs can occur when an applet or control overrides the default value of a procedure option. To resolve this issue, specify a value for the option rather than relying on the default. For example, consider a bubble plot that is being displayed in the Graph applet. The default bubble size is 5. The Graph applet overrides that default with a larger bubble size. To apply a bubble size of 5, specify BSIZE=5 in the BUBBLE statement, rather than relying on the default value of the BSIZE= option.



The Annotate Facility

Chapter 24 **Using Annotate Data Sets** 587

Chapter 25 **Annotate Dictionary** 613



CHAPTER

24

Using Annotate Data Sets

<i>Overview</i>	587
<i>Enhancing Existing Graphs</i>	588
<i>Creating Custom Graphs</i>	588
<i>Creating Annotate Graphics</i>	589
<i>About the Annotate Data Set</i>	589
<i>Structure of An Annotate Data Set</i>	589
<i>Annotate Variables</i>	591
<i>Annotate Functions</i>	594
<i>About Annotate Graphics</i>	595
<i>Graphics Elements</i>	595
<i>Coordinates</i>	596
<i>Coordinate Systems</i>	596
<i>Ranges for Cells</i>	598
<i>Internal Coordinates</i>	598
<i>Attribute Variables</i>	599
<i>Creating an Annotate Data Set</i>	599
<i>Using the DATA Step</i>	600
<i>Using Annotate Macros in the DATA Step</i>	600
<i>Effect of Missing Values</i>	601
<i>Producing Graphics Output from Annotate Data Sets</i>	601
<i>Including Annotate Graphics with Procedure Output</i>	601
<i>Producing Only Annotate Graphics Output</i>	601
<i>Using the Annotate Variables for Web Output</i>	602
<i>Annotate Processing Details</i>	602
<i>Order in Which Graphics Elements Are Drawn</i>	602
<i>Controlling the Processing with the WHEN Variable</i>	602
<i>Using BY-Group Processing with the Annotate Facility</i>	603
<i>Using the LIFO Stack</i>	603
<i>Debugging</i>	604
<i>Examples</i>	604
<i>Labeling Cities on a Map</i>	604
<i>Labeling Subgroups in a Vertical Bar Chart</i>	607
<i>Drawing a Circle of Stars</i>	609

Overview

The Annotate facility enables you to generate a special data set of graphics commands from which you can produce graphics output. This data set is referred to as an *Annotate data set*. You can use it to generate custom graphics or to enhance graphics

output from many SAS/GRAPH procedures, including GCHART, GCONTOUR, GMAP, GPLOT, GPRINT, GRADAR, GSLIDE, and G3D.

Enhancing Existing Graphs

The Annotate facility enhances output from SAS/GRAPH procedures by adding graphics elements to the output. For example, you can

- label points on a map using map coordinates
- label bars on horizontal and vertical bar charts
- label points on a plot
- create a legend for a three-dimensional graph.

Figure 24.1 on page 588 shows GMAP procedure output annotated with stars and labels at selected cities.

Figure 24.1 Annotate Graphics Applied to a Map



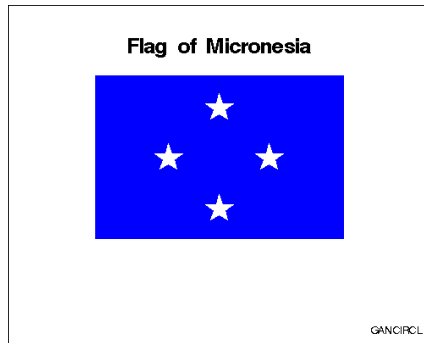
The program that creates this output is in “Labeling Cities on a Map” on page 604.

Creating Custom Graphs

You can also use an Annotate data set to create custom graphics. For example, you can use Annotate graphics commands to

- create various types of graphs (including pie charts, bar charts, and plots)
- draw graphics elements such as lines, polygons, arcs, symbols, and text.

Figure 24.2 on page 589 is an example of a custom graph that uses Annotate commands to draw the graphic elements.

Figure 24.2 Custom Graphics Using Only Annotate Commands

The program that creates this output is in “Drawing a Circle of Stars” on page 609.

Creating Annotate Graphics

In order to create and use Annotate graphics, you must first understand the structure and functioning of the Annotate data set. For this information see “About the Annotate Data Set” on page 589. Once you understand the way the data set works, you can follow these three steps to create Annotate graphics:

- 1 Determine what you want to draw, and where (location) and how (coordinate system) you want to position it on the graphics output. (See “About Annotate Graphics” on page 595.)
- 2 Build an Annotate data set of graphics commands using the Annotate variables and functions. (See “Creating an Annotate Data Set” on page 599.)
- 3 Submit a SAS/GRAPH procedure to produce the graphics output. (See “Producing Graphics Output from Annotate Data Sets” on page 601.)

About the Annotate Data Set

In an Annotate data set, each observation represents a command to draw a graphics element or to perform an action. The graphic elements drawn by these commands can be added to SAS/GRAPH output or displayed with the GANNO or GSLIDE procedure as a custom graphic.

The observations in an Annotate data set use a set of predefined Annotate variables. The values of the variables in the observation determine what is done and how it is done. To create these observations, you assign values to the variables either explicitly with a DATA step or implicitly with Annotate macros. See “Creating an Annotate Data Set” on page 599.

The following sections describe the items in an Annotate data set and explain how SAS/GRAPH software uses the commands in an Annotate data set to create graphics elements.

Structure of An Annotate Data Set

Output 24.1 is an example of an Annotate data set called TRIANGLE. The observations in this data set contain the commands that create a text label, move to a

point in the output, and draw a triangle. (The DATA step that creates TRIANGLE is shown in “Using the DATA Step” on page 600.)

Output 24.1 Listing of the Annotate Data Set TRIANGLE

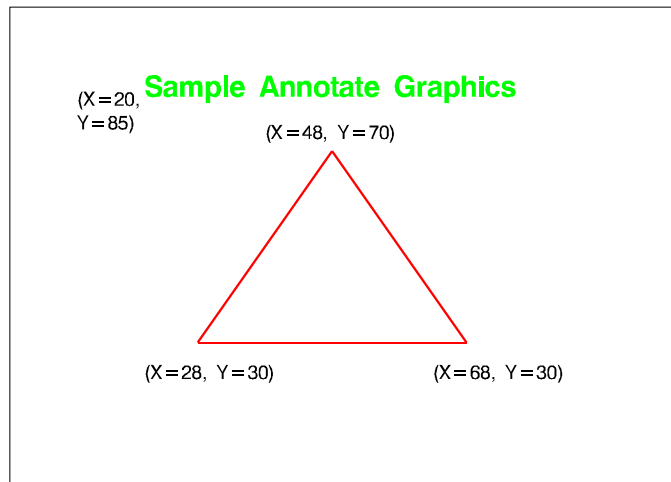
OBS	FUNCTION	X	Y	HSYS	XSYS	YSYS	STYLE	COLOR	POSITION	SIZE	LINE	TEXT
1	label	20	85	3	3	3	swissb	green	6	6.0	.	Sample Annotate Graphics
2	move	28	30	3	3	3	swissb	green	6	6.0	.	Sample Annotate Graphics
3	draw	68	30	3	3	3	swissb	red	6	0.8	1	Sample Annotate Graphics
4	draw	48	70	3	3	3	swissb	red	6	0.8	1	Sample Annotate Graphics
5	draw	28	30	3	3	3	swissb	red	6	0.8	1	Sample Annotate Graphics

Note: A blank denotes a missing value for a character variable. A ‘.’ denotes a missing value for a numeric variable. \triangle

Each observation in this data set contains complete instructions for drawing a graphic or moving to a position to draw a graphic. The value of the FUNCTION variable determines what the observation does. Other variables control how the function is performed. This list describes each observation in the TRIANGLE and the task it performs:

- 1 Create a label. This instruction draws a green label at position 20,85 (in X,Y coordinates). The value of the FUNCTION variable (LABEL) tells the program *what* to do. The values of the coordinate variables X and Y combined with the values of the coordinate system variables HSYS, XSYS, and YSYS tell *where* to do it. The values of the attribute variables STYLE, COLOR, TEXT, POSITION, and SIZE tell *how* to do it. These variables specify the font (SWISSB), the color and text of the label, the position of the label in relation to X and Y (centered on the point), and the size of the text.
- 2 Go to the starting point for the triangle. The value of the FUNCTION variable (MOVE) tells the program to go to the point specified by X and Y. This is the only instruction in the observation. Notice that the values of the variables specified for the first observation persist but are not used because they have no effect on the MOVE function.
- 3 Draw the first line of the triangle. The value of the FUNCTION variable (DRAW) tells the program to draw a line from the current point (the one specified by MOVE in the second observation) to the new point specified by X and Y. The value of the COLOR variable changes to red.
- 4 Draw the second line of the triangle.
- 5 Draw the third line of the triangle.

Figure 24.3 on page 591 shows the green title and the red triangle produced by the TRIANGLE data set and displayed with the GANNOChapter 26, “The GANNO Procedure,” on page 707 procedure. Notes on the figure in black contain the X and Y coordinates of the graphics elements.

Figure 24.3 Annotate Output from the TRIANGLE Data Set

Annotate Variables

Annotate variables have predefined names. In each observation, the Annotate facility looks only for variables with those names. Other variables can be present, but they are ignored. Conceptually, there are three types of variables:

an action variable	tells <i>what</i> to do. The only action variable is FUNCTION, which specifies what graphics element to draw (graphics primitive) or what action to take (programming function).
positioning variables	tell <i>where</i> to do it. The positioning variables specify the point at which to draw the graphics element.
attribute variables	tell <i>how</i> to do it. The attribute variables specify the characteristics of the graphics element (for example, color, size, line style, text font).

There is also an HTML variable, which provides linking information when you want to use the annotate data set to generate a drill-down graph that can be viewed in a Web browser.

Table 24.1 on page 591 lists all Annotate variables, grouped by task, and briefly describes each one. See “Annotate Variables” on page 642 for a complete description of each variable.

Table 24.1 Summary of Annotate Variables

Task Group	Variable	Description
Variable that defines an action	FUNCTION	specifies a drawing or programming action; Table 24.2 on page 594 describes these actions.
Positioning variables that determine coordinate values	GROUP	uses the value of the GCHART GROUP= option in place of X or Y
	MIDPOINT	uses the value of the GCHART MIDPOINT= option in place of X or Y

Task Group	Variable	Description
	SUBGROUP	uses the value of the GCHART SUBGROUP= option in place of X or Y
	X	specifies a numeric horizontal coordinate
	Y	specifies a numeric vertical coordinate
	Z	specifies a numeric third dimensional coordinate; used with G3D procedure only
	XC	specifies a horizontal character coordinate; only used with data coordinate systems 1, 2, 7, 8
	YC	specifies a vertical character coordinate; only used with data coordinate systems 1, 2, 7, 8
Positioning variables that contain internal coordinates	XLAST, YLAST	contain the X and Y coordinates of the last nontext function
	XLSTT, YLSTT	contain the X and Y coordinates of the last text function
Positioning variables that specify coordinate systems	HSYS	specifies type of units for the SIZE variable
	XSYS	specifies coordinate system for X or XC coordinates
	YSYS	specifies coordinate system for Y or YC coordinates
	ZSYS	specifies coordinate system for Z coordinate (G3D procedure only)
Attribute variables	ANGLE	angle of text label or starting angle of a pie slice
	CBORDER	colored border around text or symbol
	CBOX	colored box behind text or symbol
	COLOR	color of a graphics primitive
	LINE	line type to use in drawing or special control over pies and bars
	POSITION	placement and alignment for text strings
	ROTATE	angle at which to place individual characters in a text string or the delta angle (sweep) of a pie slice
	SIZE	size of an aspect of a graphics primitive; depends on FUNCTION variable (for TEXT, height of characters; for PIE, pie slice radius; for DRAW, line thickness; and so on)
	STYLE	font or pattern for a graphics element, depends on the FUNCTION variable
	TEXT	text to use in a label, symbol, or comment
	WHEN	whether a graphics element is drawn before or after procedure graphics output
Web variable	HTML	specifies link information for a drill-down graph

See Figure 24.4 on page 593 for a table that shows you which Annotate functions are used with which Annotate variables.

Annotate Functions

The FUNCTION variable accepts a set of predefined values (functions) that perform both graphics tasks and programming tasks.

The graphics functions draw the graphics elements that are illustrated in “Graphics Elements” on page 595.

The programming functions control the internal coordinates, manipulate the LIFO stack, and help you debug an Annotate data set. These programming functions are discussed in “Internal Coordinates” on page 598, “Using the LIFO Stack” on page 603, and “Debugging” on page 604.

Table 24.2 on page 594 summarizes the tasks that are performed by the Annotate functions. See “Annotate Functions” on page 615 for a complete description of the FUNCTION variable and its values.

Table 24.2 Summary of Graphics Tasks Performed by Annotate Functions

Task Group	If you want to...	Use this function...
Graphics tasks	begin to draw a polygon (starting point) and, optionally, specify a fill color and pattern	POLY
	continue drawing a polygon (additional vertex) and, optionally, specify an outline color of the polygon	POLYCONT
	draw a line from the current (X,Y) position (see MOVE and TXT2CNTL)	DRAW
	draw a point	POINT
	draw a rectangle from the current (X,Y) position (see MOVE and TXT2CNTL); optionally, fill with a pattern	BAR
	draw a symbol	SYMBOL
	draw line from (XLAST, YLAST) coordinates to (XLSTT, YLSTT) coordinates	DRAW2TXT
	draw pie slice, circle, or arc	PIE
	draw text	LABEL
	move to the specified point (X,Y)	MOVE
Programming tasks	put a frame around the area defined by XSYS and YSYS, optionally, fill with a pattern	FRAME
	insert a comment in the data set (no action); documentation aid	COMMENT
	copy (XLAST, YLAST) coordinates to (XLSTT, YLSTT) coordinates	CNTL2TXT
	copy (XLSTT, YLSTT) coordinates to (XLAST, YLAST) coordinates	TXT2CNTL
	exchange LSTT and LAST coordinates	SWAP
	get coordinates of a point on a pie slice outline	PIEXY
	get values for LAST and LSTT coordinates from LIFO stack	POP
	put current values of LAST and LSTT coordinates onto LIFO stack	PUSH

Task Group	If you want to...	Use this function...
	set pie radius and coordinates for center; does not draw a pie	PIECNTR
	turn on trace of previous values and LIFO stack	DEBUG

See Figure 24.4 on page 593 for a table that shows you which Annotate functions work with which Annotate variables.

About Annotate Graphics

When you create Annotate graphics, you specify these things:

- what to draw (graphics elements)
- where to draw those elements (the coordinates of the position on the output)
- how to draw (characteristics of the element such as size or color).

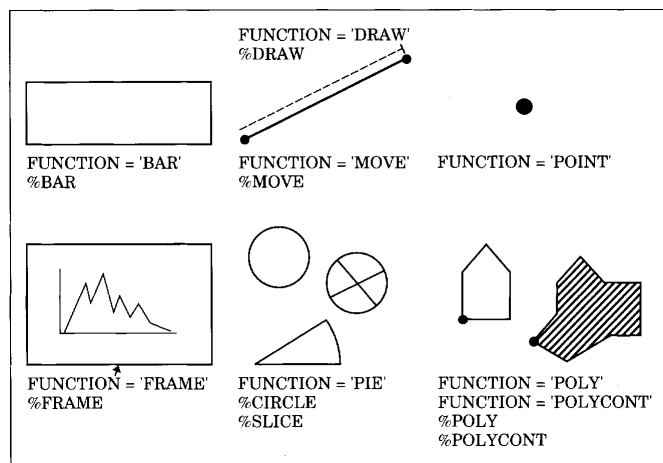
The following sections describe the components of the graphics output that are produced by an Annotate data set.

Graphics Elements

In an Annotate data set, the FUNCTION variable determines the graphics element that is drawn.

The particular graphics elements that you can draw are shown in Figure 24.5 on page 595 along with the value of the FUNCTION variable or Annotate macro that draws them.

Figure 24.5 Annotate Graphics Elements



You can control the position of graphics elements in the following ways:

- explicitly, using coordinates that you supply.
- dependently, based on the location of features in the SAS/GRAPH output. For example, when you use the GCHART procedure, you can label the parts of a

subgrouped vertical bar chart by using the SUBGROUP variable in your Annotate data set. The Annotate facility enables you to label subgroups without having to specify the actual coordinates of the subgroup bar.

- dependently, based on values that are supplied from other data sets. For example, you can label the ending point of a plot line in the GPLOT procedure by extracting the value of the last point in the sorted input data set.

Coordinates

Coordinates specify where to put graphics elements. These variables can contain coordinate values:

- X, Y, and sometimes Z are used for numeric coordinates.
- XC and YC are used for character coordinates.
- GROUP, MIDPOINT, and SUBGROUP can be used when you annotate output from procedures such as GCHART. Use these variables to specify coordinates for horizontal or vertical bar charts.

Coordinates are interpreted in terms of a coordinate system in order to identify a precise location in the graphics output.

Coordinate Systems

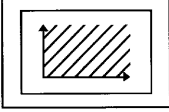
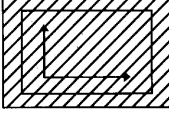
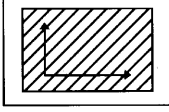
A coordinate system determines how coordinates are interpreted. You specify a coordinate system to use for each dimension, using the XSYS, YSYS, and ZSYS variables (for X, Y, and Z, respectively). Use ZSYS to annotate graphics output only from the G3D procedure.

You also specify a coordinate system for the SIZE variable using the HSYS variable. HSYS takes the same kinds of values as XSYS, YSYS, and ZSYS. The SIZE variable specifies the size of a graphics element, such as the width of lines (for example, FRAME), the radius of pie slices (for example, PIE, PIECNTR, and PIEXY), or the height of text (for example, LABEL and SYMBOL).

These are the important components of the Annotate coordinate systems:

- *Area*: Each coordinate system refers to one of three drawing areas: data area, procedure output area, and graphics output area. Coordinates are measured from a different origin for each area; they also have different limits. Figure 24.6 on page 597 shows the areas on the graphics output and the coordinate systems that use them.

Figure 24.6 Areas and Their Coordinate Systems

Area	Unit	Coordinate System	
	Data	Absolute	Relative
		%	1
	Graphics Output Area	Absolute	Relative
		%	3
	Procedure Output Area	Absolute	Relative
		%	5
		Cells	C

- *Units:* The units for a coordinate system are based on one of the following:
 - data values (for data coordinate systems). The range of values depends on the range of data expressed along the axes of the graph.
 - cells (for coordinate systems for the procedure output area or graphics output area). The range of values depends on the type of area. See “Ranges for Cells” on page 598.
 - percentages of the total area available, that is, percent of the data area, or percent of the procedure output area, or percent of the graphics output area.
- *Placement:* The placement of a coordinate can be absolute or relative. Absolute coordinates name the exact location for a graphics element in the graphics output. Relative coordinates name the location with respect to another graphics element in the output.

Table 24.3 on page 597 describes the coordinate system values for the XSYS, YSYS, ZSYS, and HSYS variables.

Table 24.3 Coordinate System Values for XSYS, YSYS, ZSYS, and HSYS Variables

Type of Coordinates	Area	Units	Range	Value for XSYS, YSYS, ZSYS, HSYS
Absolute	data	%	0-100% of axis	1' *
	data	values	minimum to maximum of axis	2' *
	graphics output area	%	0-100% of graphics output area	3'
	graphics output area	cells	0 to limit of graphics output area	4'
	procedure output area	%	0-100% of procedure output area	5'
	procedure output area	cells	0 to limit of procedure output area	6'
Relative	data	%	0-100% of axis	7' *

Type of Coordinates	Area	Units	Range	Value for XSYS, YSYS, ZSYS, HSYS
	data	values	minimum to maximum of axis	8' *
	graphics output area	%	0-100% of graphics output area	9'
	graphics output area	cells	0 to limit of graphics output area	A'
	procedure output area	%	0-100% of procedure output area	B'
	procedure output area	cells	0 to limit of procedure output area	C'

*Coordinate systems 1, 2, 7, and 8 are not valid with block, pie or star charts in the GCHART procedure or surface, prism or block maps with the GMAP procedure.

Ranges for Cells

The available range for coordinate systems that are measured in cells differs by area:

graphics output area

The range of cells that are available for the graphics output area depends on the device and the number of rows and columns that are set by the HPOS= and VPOS= graphics options or by the PCOLS and LCOLS device parameters.

procedure output area

As with the graphics output area, the range of cells available for the procedure output area depends on the device and the number of rows and columns set by the HPOS= and VPOS= graphics options or by the PCOLS and LCOLS device parameters. However, the procedure output area is sized *after* areas for titles and footnotes are allocated and is reduced accordingly. If you specify that the legend appear outside of the axis area, the procedure output area also decreases by the size of the legend.

See “Procedure Output and the Graphics Output Area” on page 34 for descriptions of the procedure output area and the graphics output area.

Internal Coordinates

The Annotate facility maintains two pairs of internal coordinates that are stored in internal variables:

- coordinates of the last graphics element drawn or the coordinates from the last move are stored in the variables XLAST and YLAST
- coordinates of the last text drawn are stored in the variables XLSTT and YLSTT.

Many functions use these internal coordinates as a starting point, relying on the coordinates that are specified with the function as an ending point. For example, in the BAR function, the (XLAST, YLAST) coordinate pair is used for the lower left corner; the position defined by the X and Y variables is used for the upper-right corner. (For details, see “BAR Function” on page 615.) These internal variables can also provide default coordinates if X, XC, Y, or YC contains a missing value.

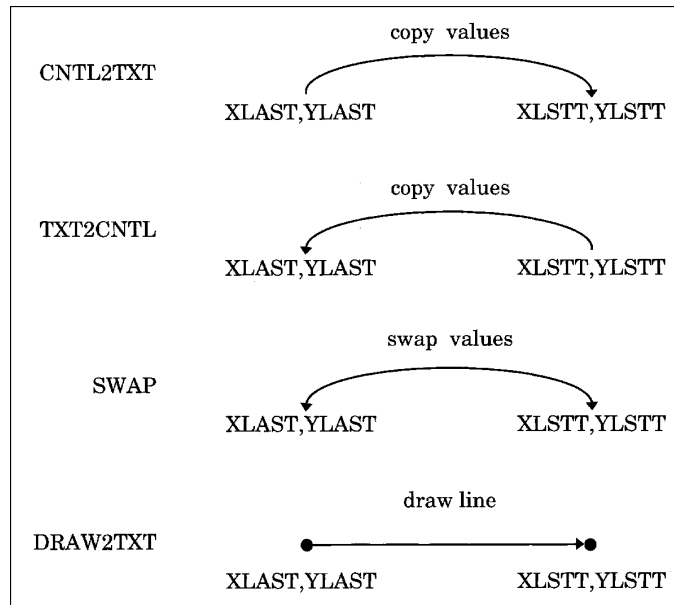
The internal coordinates are automatically updated by some of the Annotate functions. The text functions, LABEL and SYMBOL, update the (XLSTT, YLSTT) variables. The BAR, DRAW, MOVE, PIE, and POINT functions update the (XLAST, YLAST) variables.

You cannot explicitly assign a value to XLAST, YLAST, XLSTT, or YLSTT because they are internal variables. For example, you cannot make this assignment:

```
xlast=50;
```

However, you can use several functions to directly manipulate the values of the internal coordinates. The functions are shown in Figure 24.7 on page 599.

Figure 24.7 Programming Functions That Manipulate System Variables



For a complete description, see “Annotate Internal Coordinates” on page 678.

Attribute Variables

Attribute variables control the appearance of the graphics elements. Each function uses only a subset of these variables. See Table 24.1 on page 591 for a list of attribute variables.

What an attribute variable controls often depends on the graphics element to which it applies. For example, the SIZE variable controls the width of a line when it is used with FUNCTION='DRAW', but it controls the text height when it is used with FUNCTION='LABEL'.

For a complete description of the attribute variables and the aspect of the graphics elements that they control, see “Annotate Variables” on page 642.

Creating an Annotate Data Set

Once you have determined what you are going to draw and how you want it to appear in the output, you need to build an Annotate data set. Although there are many ways to create SAS data sets, the most commonly used method for creating Annotate data sets is with a DATA step that uses either

- assignment statements that you explicitly output as separate observations
- Annotate macros, which implicitly assign values to Annotate variables.

Most of the examples in this documentation use a DATA step with assignment statements. For more information on creating SAS data sets, see *SAS Language Reference: Concepts*.

Using the DATA Step

When you use the SAS DATA step with assignment statements, each statement provides a value for an Annotate variable. After you have assigned all of the variable values for an observation, you must use an OUTPUT statement to write the observation to the data set. For example, the following statements create the TRIANGLE data set shown in Output 24.1:

```
data triangle;

    /* declare variables */
    length function style color $ 8 text $ 25;
    retain hsys xsys ysys '3';

    /* create observation to draw the title */
    function='label'; x=20; y=85; position='6';
    text='Sample Annotate Graphics';
    style='swissb'; color='green'; size=6;
    output;

    /* create observations to draw the triangle */
    function='move'; x=28; y=30; output;
    function='draw'; x=68; y=30; size=.8; line=1;
    color='red'; output;
    function='draw'; x=48; y=70; output;
    function='draw'; x=28; y=30; output;
run;

proc ganno annotation=triangle;
run;
quit;
```

Notice that a RETAIN statement sets the values of the HSYS, XSYS, and YSYS variables. RETAIN statements are useful when you want to select the values for variables that are required for many functions and the value is the same for all of them.

The SIZE, LINE, and COLOR variables are included with only the first DRAW function. Using this method to create the data set, the values set in the first DRAW function carry over to subsequent DRAW functions.

The PROC GANNO takes as input the annotate data set “triangle” created by the previous DATA step and creates the output shown in Figure 24.3 on page 591.

Using Annotate Macros in the DATA Step

A set of Annotate macros is provided in the SAS sample library. You can use macro calls in a DATA step to create observations in an Annotate data set. You can also use Annotate macros and explicit variable assignments together in the same DATA step. For complete information, see “Annotate Macros” on page 679 and “Using Annotate Macros” on page 697.

Effect of Missing Values

Annotate data sets follow the same rules for missing values as any other SAS data set. (See *SAS Language Reference: Concepts* for information on the effect of missing values in a data set.)

Variables that have a missing value use a default value. For example, if the COLOR variable has a missing value, then the first color in either the colors list that is defined by the COLORS= graphics option, if specified, or the device's default colors list is used. If the FUNCTION variable has a missing value, LABEL is used. If the X variable is missing, the value of the XLSTT internal coordinate is used for text functions and the XLAST internal coordinate is used for nontext functions. See “Annotate Variables” on page 642 for the default value of each Annotate variable.

You probably should not depend on this effect when you create an Annotate data set. If the data set is structured so that observations depend on prior observations setting attributes for them, then you may have extra work to do if you change the order of observations later.

Sometimes missing values are required to produce the desired results. If you have calculated the coordinates of a point and have the values stored in (XLAST,YLAST) or (XLSTT,YLSTT), you can force Annotate to use the internal coordinates by supplying missing values for the X and Y variables. See “Annotate Internal Coordinates” on page 678 for details on using the (XLAST,YLAST) and (XLSTT,YLSTT) internal coordinates.

Producing Graphics Output from Annotate Data Sets

You can display Annotate graphics in two ways:

- annotate output from a SAS/GRAPH procedure by assigning the Annotate data set to the PROC statement or the action statement, or both.
- display only the Annotate graphics by assigning the Annotate data set to either the GANNO or GSLIDE procedure.

Including Annotate Graphics with Procedure Output

To annotate SAS/GRAPH procedure output, you must include the ANNOTATE= option in the appropriate statement in the procedure. ANNOTATE= must name the Annotate data set that you have already created. If you want the Annotate graphics to apply to all graphs produced by a procedure, you should include ANNOTATE= in the PROC statement. If you want the Annotate graphics to apply only to the graph produced by an action statement within the procedure, include ANNOTATE= in the action statement. You can specify Annotate data sets in both places.

When you annotate a SAS/GRAPH procedure, the Annotate graphics are displayed and stored as part of the graphics output that the procedure produces.

Producing Only Annotate Graphics Output

To produce Annotate graphics without other procedure output, use the GANNO procedure or the GSLIDE procedure:

- The GANNO procedure produces graphics output consisting only of Annotate graphics. See Chapter 26, “The GANNO Procedure,” on page 707Chapter 26, “The GANNO Procedure,” on page 707 for information on displaying or storing Annotate graphics.

- The GSLIDE procedure can also produce graphics output consisting only of Annotate graphics. In addition, you can enhance the graphics output with TITLE, NOTE, and FOOTNOTE statements. See Chapter 44, “The GSLIDE Procedure,” on page 1277 for details.

Using the Annotate Variables for Web Output

Most of the annotate variables can be used in programs that generate output for the Web. For more information on the annotate functions and variables, see the Chapter 25, “Annotate Dictionary,” on page 613. For information on using annotate data sets in Web output, see Chapter 17, “Generating Web Output with the Annotate Facility,” on page 499.

Annotate Processing Details

Order in Which Graphics Elements Are Drawn

When a procedure uses an Annotate data set, it reads and interprets the observations one at a time, starting with the first observation and proceeding to the last. The order of the observations in the data set determines the order in which the graphics elements are generated. If the coordinates of two graphics elements overlap, the graphics element produced by an earlier observation can be overwritten by any graphics elements that are produced by subsequent observations. As a result, graphics elements can overlay each other and they can also overlay or be overlaid by procedure output.

CAUTION:

Overlay behavior is device-dependent. Most terminals, cameras, and some printers demonstrate overlay behavior because the process of drawing updates pixels as each graphics element is drawn. Plotters do not overlay the graphics elements internally before plotting; they draw graphics elements on top of each other on the paper. The area where graphics elements overlap shows one color bleeding through the color that overlays it. To ensure that one graphics element overlays another, use the WHEN variable. \triangle

Controlling the Processing with the WHEN Variable

The WHEN variable determines the order in which observations in an Annotate data set are processed. It determines if observations are processed *before* or *after* output that is produced by a SAS/GRAPH procedure. This means that Annotate graphics can be overlaid by procedure output or can overlay procedure output. By default, Annotate graphics are drawn before the procedure output.

In effect, you can have two sets of Annotate graphics elements that are generated for the same output:

- Annotate graphics drawn before procedure output (the default, WHEN='B').
- Annotate graphics drawn after procedure output (WHEN='A').

Within each set, graphics elements are drawn in the order that they appear in the Annotate data set and overlay each other as appropriate (on devices that demonstrate overlay behavior). For details, see the description of the WHEN variable on “WHEN Variable” on page 666.

Using BY-Group Processing with the Annotate Facility

You can use the Annotate facility with procedures that use BY statements to annotate each graph that is generated with a BY statement. The Annotate graphics for each graph are generated depending on the value of the BY variable. To use BY-group processing with the Annotate facility, your program must meet the following conditions:

- Both the input data set for the procedure and the Annotate data set must contain the same BY variable.
- The BY variable must be defined as the same type (character or numeric) and length in both data sets.
- If a label or format is associated with a BY variable in one data set, the same label or format has to be associated with it in the other data set.
- Both data sets must be sorted by the BY variable.
- The ANNOTATE= option must be specified in an action statement in the procedure. If you specify the ANNOTATE= option in the PROC statement, the Annotate graphics are used for all graphs that are generated by the procedure rather than for unique values of the BY variable.

See “BY Statement” on page 141 for details.

Using the LIFO Stack

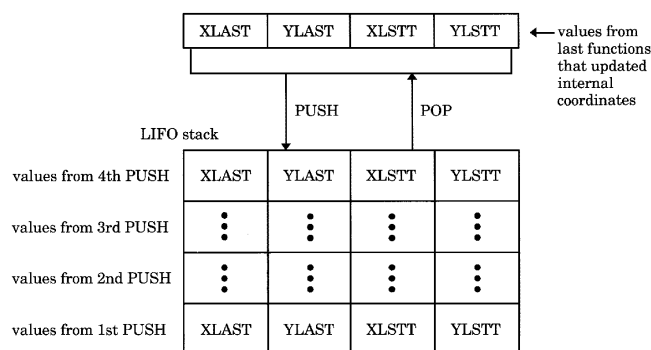
The FUNCTION variable supports several programming functions that manipulate the internal coordinates and provide other utility operations. Several of these functions use the LIFO stack to track and set variable values.

The *LIFO* (last-in-first-out) *stack* is a storage area where you can keep internal coordinate values for later use. It is useful when you want to save the current values of (XLAST,YLAST) and (XLSTT,YLSTT) and use them with functions later in the DATA step.

You store and retrieve values from the stack using the PUSH and POP functions. The PUSH function copies the current values of XLAST, YLAST, XLSTT, and YLSTT onto the stack. The POP function copies values from the stack into XLAST, YLAST, XLSTT, and YLSTT.

LIFO stacks manage the stored data so that the last data stored in the stack is the first data removed from the stack. This means that a POP function retrieves the values most recently stored with a PUSH function. Figure 24.8 on page 603 illustrates how PUSH and POP functions work together.

Figure 24.8 Using PUSH and POP to Store and Retrieve Coordinate Values



See also “Internal Coordinates” on page 598.

Debugging

You can print your Annotate data set with the PRINT procedure. This is an easy way to examine the Annotation that you have specified or to debug your program. For example, a listing such as the one in Output 24.1 provides complete information about the value that you specify for each variable in every observation.

For more complex problems, the DEBUG function enables you to display the values of Annotate variables and internal coordinates before and after a function is submitted. The values are written to the SAS log.

If there is an error in your Annotate data set, one or more diagnostic messages are printed in the SAS log:

- If an error is found in preprocessing, this message appears:

```
NOTE: ERROR DETECTED IN ANNOTATE= libref.dataset
```

- If an error is found as an observation is being read, this message appears:

```
PROBLEM IN OBSERVATION number-message
```

where *message* is the text of the error message.

- If the error limit of 20 errors is reached at any point during processing of the data set, a termination message similar to this one appears:

```
ERROR LIMIT REACHED IN ANNOTATE PROCESS
```

```
20 TOTAL ERRORS
```

For an explanation of common diagnostic messages, refer to the Help facility.

Examples

The following examples show how to annotate graphics that are created with SAS/GRAPH procedures and how to build custom graphics:

- “Labeling Cities on a Map” on page 604
- “Labeling Subgroups in a Vertical Bar Chart” on page 607
- “Drawing a Circle of Stars” on page 609

Other examples that use Annotate data sets are as follows:

- Example 1 on page 710 (and others in that chapter)
- Chapter 44, “The GSLIDE Procedure,” on page 1277
- “Drawing a Circle of Stars” on page 609

Labeling Cities on a Map

Features:

Annotate function:	SYMBOL
Annotate variables:	HSYS
	POSITION

	SIZE
	TEXT
	WHEN
	X and Y
	XSYS
	YSYS
<i>Sample library member:</i>	GANCITY

Figure 24.9 Map with Labeled Cities



This example labels a map of the continental United States with the location and names of three cities. The GMAP procedure draws a map of the U.S. and an Annotate data set adds the stars and labels.

The DATA step that creates the Annotate data set gets the x and y coordinates of the cities to be labeled from the MAPS.USCITY data set. Because MAPS.USCITY stores projected coordinates in the X and Y variables, the DATA step does not need to reassign the variable values. Also because X and Y contain data values (the map data set coordinates), the XSYS and YSYS variables specify coordinate system 2, absolute data values. However, the HSYS variable that controls text height uses coordinate system 3, percent of the graphics output area.

See Example 4 on page 1180 for an example of labeling a map using map coordinates in units of latitude and longitude.

See Chapter 35, “The GMAP Procedure,” on page 995 for more information on using map data sets

Note: If the libref MAPS is automatically assigned at your site to the SAS data library containing the Institute-supplied map data sets, you can omit the LIBNAME statement. △

Assign the libref MAPS, if necessary, and set the graphics environment.

```
libname maps 'SAS-data-library';
options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swissb htitle=6 htext=3;
```

Subset the U.S. map data set by omitting Alaska and Hawaii.

```
data lower48;
  set maps.us;
  if state ne stfips('AK');
  if state ne stfips('HI');
run;
```

Create the Annotate data set, CITYSTAR. CITYSTAR contains the commands that draw a star and a label at each of the three cities. Setting WHEN to A draws the annotation after the map.

```
data citystar;
  length function style color $ 8 position $ 1
         text $ 20;
  retain xsys ysys '2' hsys '3'
         when 'a';
```

Include the values of selected variables from MAPS.USCITY. X and Y contain projected coordinates; CITY contains names; STATE contains FIPS codes. Because there are several Atlantas, a STATE value is necessary.

```
set maps.uscity(keep=x y city state);
if (city='Atlanta' and state=13)
  or city='Chicago'
  or city='Seattle';
```

Create the observation that draws the star. The text string V is the character code for the star figure in the MARKER font assigned by the STYLE variable.

```
function='symbol'; style='marker'; text='V'; color='red'; size=5;
output;
```

Create the observation that labels the city. TEXT is assigned the value of CITY. The font is SWISSB. SIZE uses the units assigned by HSYS so text height is 5 percent of the height of the graphics output area. POSITION 8 places the label directly below the city location.

```
function='label'; style='swissb'; text=city; color='green';
size=5; position='8'; output;
run;
```

Define the title and footnote for the map.

```
title 'Distribution Center Locations';
footnote font=swiss j=r 'GANCITY';
```

Define patterns for the map areas. MEMPTY colors only the state borders.

```
pattern value=mempty color=blue repeat=49;
```

Generate the map and assign the annotate data set to the CHORO statement.

```
proc gmap data=lower48 map=lower48;
  id state;
  choro state / annotate=citystar discrete nolegend;
run;
quit;
```

Labeling Subgroups in a Vertical Bar Chart

Features:

Annotate function: LABEL (default)

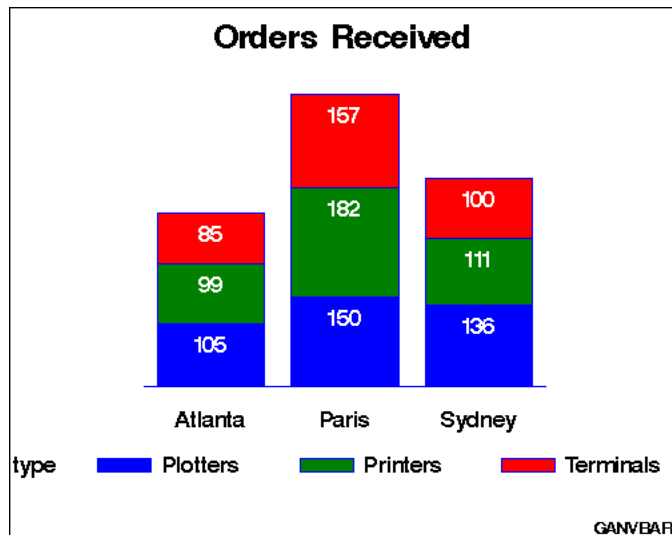
Annotate variables: MIDPOINT

POSITION

SUBGROUP

Sample library member: GANVBAR

Figure 24.10 Bar Chart with Labeled Subgroups



This example shows how to label subgroups in a vertical bar chart that is generated by the GCHART procedure. Each bar represents total orders for a city and is

subgrouped by the type of order. The Annotate facility labels each subgroup with the number of orders for that category. The coordinates that position the subgroup labels are derived from the values of the GCHART procedure variables CITY (the chart (or midpoint) variable) and TYPE (the subgroup variable). These variables are assigned to the corresponding Annotate variable.

See Chapter 29, “The GCHART Procedure,” on page 773 for more information on creating bar charts.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(blue green red) ctext=black htitle=6
          ftitle=swissb htext=4 ftext=swiss;
```

Create the data set SOLD.

```
data sold;
  length type $ 10;
  input city $ units type $ ;
  datalines;
Atlanta  99 Printers
Atlanta 105 Plotters
Atlanta  85 Terminals
Paris    182 Printers
Paris   150 Plotters
Paris   157 Terminals
Sydney  111 Printers
Sydney  136 Plotters
Sydney  100 Terminals
;
run;
```

Create the Annotate data set, BARLABEL. The MIDPOINT variable uses the values of the chart variable CITY to provide the X coordinate for the subgroup labels. The SUBGROUP variable uses the values of the variable TYPE to provide the Y coordinate that vertically positions the labels in the bar. Because no function is specified, the data set uses the default function, LABEL. The POSITION value **E** places the labels just below the top of each subgroup bar.

```
data barlabel;
  length color style $ 8;
  retain color 'white' when 'a' style 'swissb'
          xsys ysys '2' position 'E' size 4 hsys '3';
  set sold;
  midpoint=city;
  subgroup=type;
  text=left(put(units,5.));
run;
```

Define the title and footnote.

```

title 'Orders Received';
footnote h=3 j=r 'GANVBAR';

```

Define axis characteristics. AXIS1 suppresses the vertical axis. AXIS2 drops the midpoint axis label.

```

axis1 label=none major=none minor=none style=0
      value=none;
axis2 label=none;

```

Generate a vertical bar chart and assign the Annotate data set to the VBAR statement.

```

proc gchart data=sold;
  vbar city / type=sum
          sumvar=units
          subgroup=type
          width=17
          raxis=axis1
          maxis=axis2
          annotate=barlabel;

run;
quit;

```

Drawing a Circle of Stars

Features:

Annotate function:	BAR
	CNTL2TXT
	FRAME
	LABEL
	MOVE
	PIECNTR
	PIEXY
	SYMBOL
Annotate variables:	COLOR
	HSYS, XSYS, YSYS
	LINE
	STYLE
	TEXT
	XLAST and YLAST
	XLSTT and YLSTT

Sample library member: GANCIRCL

Figure 24.11 Stars Positioned in a Circle with GANNO



This example shows how to use an Annotate data set to draw a flag that is composed of a rectangle and four stars. The stars are positioned by placing them on an imaginary circle. The program uses the `PIECNTR` and `PIEXY` functions to find the points on the circle and the `CNTL2TXT` programming function to transfer coordinate values. It also processes Annotate assignment statements in a `DO` loop. The `GANNO` procedure displays the Annotate graphics.

Set the graphics environment.

```
goptions reset=global cback=white colors=(black);
```

Create the Annotate data set, FLAG. `XSYS`, `YSYS`, and `HSYS` specify coordinate system 3, absolute size of the graphics output area.

```
data flag;
  length function style color $ 8 text $ 30;
  retain xsys ysys hsys '3';
```

Draw a frame. The `FRAME` function uses the default color `BLACK` to draw a frame around the graphics output area specified by the `XSYS` and `YSYS` variables.

```
function='frame'; output;
```

Draw the footnote. The `LABEL` function draws the text specified in the `TEXT` variable. `X` and `Y` explicitly position the footnote on the graphics output area.

```
function='label'; x=92; y=5; text='GANCIRCL';
  style='swiss'; size=3; position='5'; output;
```


Draw the title. The values of FUNCTION, POSITION, and COLOR remain the same because no new values are assigned.

```
x=50; y=90; text='Flag of Micronesia';
      style='swissb'; size=6; output;
```

Draw the background. MOVE specifies the lower left corner of the rectangle that forms the flag. BAR draws the rectangle using the values of X and Y for the upper right corner. The LINE value of 3 fills the figure with the specified color.

```
function='move'; x=20; y=30; output;
function='bar'; x=80; y=80; color='blue';
      line=3; style='solid'; output;
```

Draw the circle of stars. The DO loop repeats the processing instructions defined by the nested assignment statements, placing a star every 90 degrees around the circle. To increase the number of stars, reduce the size of the angle between them and adjust the ending angle.

```
do star_ang=0 to 270 by 90;
```

The PIECNTR function is set to the center of the rectangle. PIEXY calculates a point on the arc based on the value of STAR_ANG and updates the internal coordinates XLAST and YLAST.

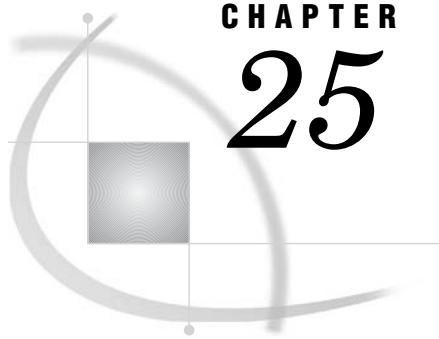
```
function='piecntr'; x=50; y=55; size=15; output;
function='piexy'; size=1; angle=star_ang; output;
```

The programming function CNTL2TXT copies the values of XLAST and YLAST to the text-handling coordinates XLSTT and YLSTT. Assigning missing values to X and Y forces the SYMBOL function to use the values of XLSTT and YLSTT to position the star. The text string V is the character code for the star figure in the MARKER font assigned by the STYLE variable.

```
function='cntl2txt'; output;
function='symbol'; style='marker'; text='V';
      angle=0; color='white'; size=10; x=.; y=.;
      output;
end;
run;
```

Use the GANNO procedure to process the Annotate data set and generate the graphics output.

```
proc ganno annotate=flag;
run;
quit;
```

CHAPTER 25

Annotate Dictionary

<i>Annotate Dictionary Overview</i>	614
<i>Annotate Functions</i>	615
<i>BAR Function</i>	615
<i>CNTL2TXT Function</i>	617
<i>COMMENT Function</i>	619
<i>DEBUG Function</i>	620
<i>DRAW Function</i>	620
<i>DRAW2TXT Function</i>	621
<i>FRAME Function</i>	622
<i>IMAGE Function</i>	625
<i>LABEL Function</i>	626
<i>MOVE Function</i>	627
<i>PIE Function</i>	628
<i>PIECNTR Function</i>	630
<i>PIEXY Function</i>	631
<i>POINT Function</i>	633
<i>POLY Function</i>	634
<i>POLYCONT Function</i>	635
<i>POP Function</i>	638
<i>PUSH Function</i>	639
<i>SWAP Function</i>	639
<i>SYMBOL Function</i>	640
<i>TXT2CNTL Function</i>	641
<i>Annotate Variables</i>	642
<i>ANGLE Variable</i>	642
<i>CBORDER Variable</i>	643
<i>CBOX Variable</i>	644
<i>COLOR Variable</i>	645
<i>FUNCTION Variable</i>	646
<i>GROUP Variable</i>	647
<i>HSYS Variable</i>	649
<i>HTML Variable</i>	651
<i>IMGPATH Variable</i>	652
<i>LINE Variable</i>	652
<i>MIDPOINT Variable</i>	654
<i>POSITION Variable</i>	656
<i>ROTATE Variable</i>	659
<i>SIZE Variable</i>	660
<i>STYLE Variable (Fonts)</i>	661
<i>STYLE Variable (Images)</i>	662
<i>STYLE Variable (Patterns)</i>	662

<i>SUBGROUP Variable</i>	664
<i>TEXT Variable</i>	666
<i>WHEN Variable</i>	666
<i>X Variable</i>	667
<i>XC Variable</i>	668
<i>XSYS Variable</i>	670
<i>Y Variable</i>	673
<i>YC Variable</i>	673
<i>YSYS Variable</i>	674
<i>Z Variable</i>	676
<i>ZSYS Variable</i>	676
<i>Annotate Internal Coordinates</i>	678
<i>XLAST, YLAST Variables</i>	678
<i>XLSTT, YLSTT Variables</i>	678
<i>Annotate Macros</i>	679
<i>%ANNOMAC Macro</i>	679
<i>%BAR, %BAR2 Macros</i>	679
<i>%CENTROID Macro</i>	680
<i>%CIRCLE Macro</i>	681
<i>%CNTL2TXT Macro</i>	681
<i>%COMMENT Macro</i>	682
<i>%DCLANNO Macro</i>	682
<i>%DRAW Macro</i>	683
<i>%DRAW2TXT Macro</i>	683
<i>%FRAME Macro</i>	684
<i>%LABEL Macro</i>	685
<i>%LINE Macro</i>	686
<i>%MAPLABEL Macro</i>	686
<i>%MOVE Macro</i>	687
<i>%PIEXY Macro</i>	688
<i>%POLY, %POLY2 Macro</i>	688
<i>%POLYCONT Macro</i>	689
<i>%POP Macro</i>	690
<i>%PUSH Macro</i>	690
<i>%RECT Macro</i>	691
<i>%SCALE Macro</i>	692
<i>%SCALET Macro</i>	693
<i>%SEQUENCE Macro</i>	694
<i>%SLICE Macro</i>	695
<i>%SWAP Macro</i>	696
<i>%SYSTEM Macro</i>	696
<i>%TXT2CNTL Macro</i>	697
<i>Using Annotate Macros</i>	697
<i>Macro Structure</i>	697
<i>Making the Macros Available</i>	697
<i>Annotate Macro Task Summary</i>	698
<i>Annotate Error Messages</i>	699

Annotate Dictionary Overview

The Annotate facility enables you to generate a special data set of graphics commands from which you can produce graphics output. This data set is referred to as an Annotate data set. You can generate a complete graph using an Annotate data set in

conjunction with the Chapter 26, “The GANNO Procedure,” on page 707 or Chapter 44, “The GSLIDE Procedure,” on page 1277 procedures, or you can apply an Annotate data set to graphics that were generated with procedures such as Chapter 29, “The GCHART Procedure,” on page 773, Chapter 30, “The GCONTOUR Procedure,” on page 885, and Chapter 35, “The GMAP Procedure,” on page 995, among others.

In addition, SAS/GRAPH supports the following procedures on the client using Java or ActiveX: GCHART, GCONTOUR, GMAP, GPLOT, GRADAR, and G3D.

In an Annotate data set, each observation represents a command to draw a graphics element or perform an action. The observations use a set of predefined “Annotate Variables” on page 642. “Annotate Functions” on page 615 determine what is to be done with each observation. “Annotate Macros” on page 679 simplify the process of drawing a graphics element. “Annotate Error Messages” on page 699 are sent to the SAS log.

For usage information and example programs, refer to “Using Annotate Macros” on page 697 and Chapter 24, “Using Annotate Data Sets,” on page 587.

Annotate Functions

In an Annotate data set, the value of the FUNCTION variable specifies what action the observation performs. Annotate functions act in conjunction with Annotate variables that determine where and how to perform the action. Many of these variables are function-dependent, that is, what they do depends on the function they are used with. For example, with the LABEL function the STYLE variable specifies a font; with the BAR function, STYLE specifies a pattern.

This section describes all of the values of the FUNCTION variable. For each function it

- describes the function’s action.
- notes whether the function updates the internal coordinate variables XLAST, YLAST and XLSTT, YLSTT.
- describes how other Annotate variables behave with the function. For a complete description of each variable, see “Annotate Variables” on page 642.

For a summary of drawing and programming tasks performed by the FUNCTION variable, see Table 24.2 on page 594 .

The variables that are available for use with each function are listed in Figure 24.4 on page 593.

BAR Function

Draws a rectangle whose lower-left corner is defined by the internal variables (XLAST, YLAST) and whose upper-right corner is defined by the specified X, Y variable pair. You can define the color of the fill, the fill pattern, and the edge lines to be drawn.

Updates: XLAST, YLAST

Syntax

FUNCTION='BAR';

Associated Variables

COLOR=*'color'*

specifies the color of either the interior of the bar or the outline of the bar. *Color* can be any SAS/GRAPH color name. The part of the bar affected depends on the value of the **STYLE** variable. If **STYLE** specifies a pattern or fill, the **COLOR** variable determines the color of the interior. If **STYLE** specifies an empty pattern, the **COLOR** variable determines the color of the outline of the bar.

GROUP=*group-value*

MIDPOINT=*midpoint-value*

SUBGROUP=*subgroup-value*

specify coordinates for HBAR and VBAR charts from the GCHART procedure. Use these variables only with the data coordinate systems 1, 2, 7, and 8.

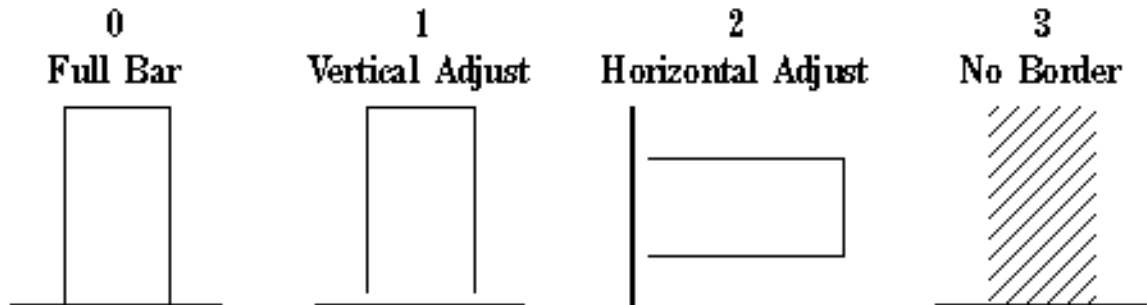
HTML=*'link-string'*

specifies the text that defines the link for drill-down.

LINE=0...3

specifies the direction in which to adjust the outline of the bar. Use **LINE** values 1 and 2 to offset a particular bar from an axis or adjoining area. The following figure illustrates **LINE** values.

Figure 25.1 LINE Values for Bars



SIZE=*thickness*

specifies a line thickness for the rectangle

STYLE=*'fill-pattern'*

specifies the pattern that fills the bar. *Fill-pattern* can be the following bar and block patterns:

SOLID a solid fill.

S

EMPTY an empty fill.

E

style<*density*> a shaded pattern:

style can be R | X | L

density can be 1...5

WHEN='B' | 'A'

specifies when to draw the bar in relation to other procedure output. See “WHEN Variable” on page 666.

X=*horizontal-coordinate*

Y=*vertical-coordinate*

Z=*depth-coordinate*

XC='character-type-horizontal-coordinate'

YC='character-type-vertical-coordinate'

define the upper-right corner of a bar (rectangle) whose lower-left corner is (XLAST,YLAST). Use the Z variable only when you are annotating output from the G3D procedure. Figure 25.2 on page 617 illustrates the use of these coordinates.

XSYS='coordinate-system'

specifies the coordinate system for the X or XC variable. The XC variable can be used only with XSYS='2'. See “XSYS Variable” on page 670 for an explanation of *coordinate-system*.

YSYS='coordinate-system'

specifies the coordinate system for Y or YC variable. The YC variable can only be used with YSYS='2'. See “YSYS Variable” on page 674 for an explanation of *coordinate-system*.

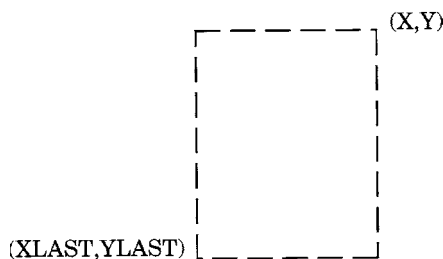
ZSYS='coordinate-system'

specifies the coordinate system for the Z variable. See “ZSYS Variable” on page 676 for an explanation of *coordinate-system*.

Details

Figure 25.2 on page 617 shows how the XLAST, YLAST, and X, Y variables define the diagonal corners of the bar. With character data, the XC and YC variables are used in place of the X and Y variables. The values of the XLAST and YLAST variables are usually initialized with a MOVE function or another function that updates the XLAST and YLAST pair. When the XC variable is used, set XSYS='2'. When the YC variable is used, set YSYS='2'.

Figure 25.2 Points Used to Construct a Bar



CNTL2TXT Function

Copies the values of the internal coordinates stored in the variable pairs (XLAST, YLAST) to (XLSTT, YLSTT).

Updates: XLSTT, YLSTT

Syntax

FUNCTION='CNTL2TXT';

Details

You can use CNTL2TXT to calculate the position of labels on a graph. For example, the following DATA step uses CNTL2TXT to position a pie slice label in the center of the arc and just beyond the arc itself, as shown in Figure 25.5 on page 619.

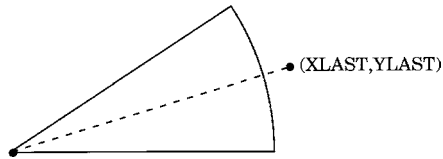
First, use the PIE function to draw the pie slice:

```
data pielabel;
  retain xsys ysys '3';
  length function style $ 8;
  function='pie'; size=20; x=30; y=30;
  style='empty'; rotate=45; output;
```

Then use the PIEXY function to calculate a point outside of the arc as shown in Figure 25.3 on page 618.

```
/* find a point that is half of the arc (rotate*.5) */
/* and is 4 units beyond the radius (size=1.1) */
function='piexy'; angle=rotate*.5; size=1.1; output;
```

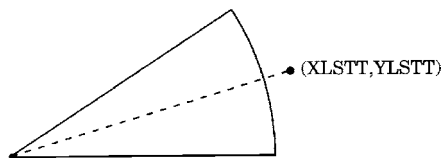
Figure 25.3 Position Calculated with the PIEXY Function



At this point, the XLAST and YLAST variables contain the coordinates of the point that is calculated by PIEXY. However, (XLAST, YLAST) cannot be used directly by text functions. Use CNTL2TXT to copy the coordinates in (XLAST, YLAST) to the XLSTT and YLSTT variables, which text functions can use. Figure 25.4 on page 618 shows the results.

```
function='cntl2txt'; output;
```

Figure 25.4 Coordinates after Using the CNTL2TXT Function



Now you can use the LABEL function to write the label as shown in Figure 25.5 on page 619. Specify missing values for the X and Y variables to force LABEL to use the XLSTT and YLSTT variables instead of the X and Y variables.

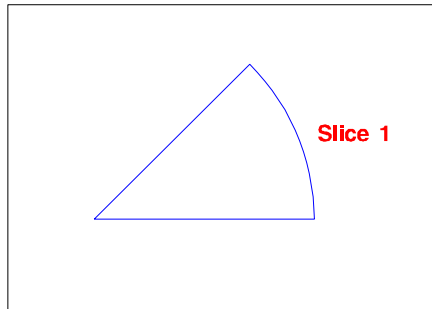
```

/* write the label 'Slice 1' and position it to      */
/* the right of the point stored in XLSTT and YLSTT */
function='label'; text='Slice 1'; angle=0; rotate=0;
position='6'; style='swissb'; size=4; x=.; y=.;
output;
run;

/* draw the Annotate graphics */
proc ganno anno=pielabel;
run;
quit;

```

Figure 25.5 Labeled Pie Slice



COMMENT Function

Inserts comments within the Annotate data set. The observations generated by the COMMENT function are ignored when the data set is processed.

Syntax

```
FUNCTION='COMMENT';
```

Associated Variables

TEXT=*'text-string'*
specifies the comment to write to the data set.

DEBUG Function

Writes the values of internal coordinates and Annotate variables to the SAS log before and after processing the next command (unless it is DEBUG) in the Annotate DATA step.

Syntax

```
FUNCTION='DEBUG';
```

DRAW Function

Draws a line in the graphics output from the (XLAST, YLAST) coordinates to the (X, Y) coordinates specified in the function.

Updates: XLAST, YLAST

Syntax

```
FUNCTION='DRAW';
```

Associated Variables

COLOR='color'

specifies the color of the line that is being drawn. *Color* can be any SAS/GRAPH color name.

GROUP=group-value

MIDPOINT=midpoint-value

SUBGROUP=subgroup-value

specify coordinates for HBAR and VBAR charts from the GCHART procedure. Use these variables only with the data coordinate systems 1, 2, 7, and 8.

HSYS='coordinate-system'

specifies the coordinate system for the SIZE variable. See “HSYS Variable” on page 649 for an explanation of *coordinate-system*.

LINE=1...46

specifies the line type of the line that is being drawn. See “Specifying Line Types” on page 207 for an illustration of the line types.

SIZE=line-thickness

specifies the thickness of the line that is being drawn. The units depend on the value of the HSYS variable. For example, if HSYS='3', the SIZE variable is in units of percent of the graphics output area. If HSYS='4', the SIZE variable is in units of cells of the graphics output area.

As the thickness of the line increases, it may be impossible to center around a given coordinate. For example, if you specify a thickness of value 2 and HSYS='4',

the first line is drawn at the (X, Y) coordinates. The second is drawn slightly above the first. The exact amount varies by device, but it is always one pixel in width. A thickness of value 3 produces one line above, one line at, and one line below the (X, Y) coordinate position. See Figure 25.6 on page 621 for examples of line thicknesses.

Figure 25.6 Sample Line Thicknesses Used with the SIZE Variable



WHEN='B' | 'A'

specifies when to draw the line in relation to other procedure output. See “WHEN Variable” on page 666.

X=*horizontal-coordinate*

Y=*vertical-coordinate*

Z=*depth-coordinate* (PROC G3D only)

XC=*'character-type-horizontal-coordinate'*

YC=*'character-type-vertical-coordinate'*

specify the endpoint of a line drawn from (XLAST, YLAST) to (X,Y).

XSYS=*'coordinate-system'*

specifies the coordinate system for the X or XC variable. The XC variable can be used only with XSYS='2'. See “XSYS Variable” on page 670 for an explanation of *coordinate-system*.

YSYS=*'coordinate-system'*

specifies the coordinate system for the Y or YC variable. The YC variable can be used only with YSYS='2'. See “YSYS Variable” on page 674 for an explanation of *coordinate-system*.

ZSYS=*'coordinate-system'*

specifies the coordinate system for the Z variable (PROC G3D only). See “ZSYS Variable” on page 676 for an explanation of *coordinate-system*.

DRAW2TXT Function

Draws a line from (XLAST, YLAST) to (XLSTT, YLSTT) without updating any of those variables.

Syntax

FUNCTION='DRAW2TXT';

Associated Variables

COLOR=*'color'*

specifies the line color. *Color* can be any SAS/GRAPH color name.

HSYS=*'coordinate-system'*

specifies the coordinate system for the SIZE variable. See “HSYS Variable” on page 649 for an explanation of *coordinate-system*.

LINE=1...46

specifies the line type of the line that is being drawn. See “Specifying Line Types” on page 207 for an illustration of the line types.

SIZE=*line-thickness*

specifies the thickness of the line that is being drawn. See “DRAW Function” on page 620 for details.

WHEN=*'B' | 'A'*

specifies when to draw the line in relation to generation of the procedure output. See “WHEN Variable” on page 666.

Details

DRAW2TXT is useful for underlining text.

DRAW2TXT does not update the (XLAST, YLAST) or (XLSTT, YLSTT) coordinates; neither can it interrupt a POLYCONT sequence.

FRAME Function

Draws a border around the portion of the display area defined by the XSYS and YSYS variables. Optionally specifies a background color for the framed area.

Syntax

FUNCTION=*'FRAME'*;

Note: The FRAME function is not supported for client-side annotate with Java. \triangle

Associated Variables

COLOR=*'color'*

specifies the frame color and, if the STYLE variable is specified, fills the interior of the frame. *Color* can be any SAS/GRAPH color name.

HSYS=*'coordinate-system'*

specifies the coordinate system for the SIZE variable. See “HSYS Variable” on page 649 for an explanation of *coordinate-system*.

Note: The HSYS variable is not supported for client-side output with ActiveX. \triangle

HTML=*'link-string'*

specifies the text that defines the link for drill-down.

LINE=1...46

specifies the line type with which to draw the frame. See “Specifying Line Types” on page 207 for an illustration of the line types.

SIZE=*line-thickness*

specifies the thickness of the line with which to draw the frame. See “DRAW Function” on page 620 for details.

Note: The SIZE variable is not supported for client-side output with ActiveX. Δ

STYLE=*'fill-pattern'*

specifies the pattern that fills the area that is bounded by the frame. *Fill-pattern* can be the following bar and block patterns:

SOLID a solid fill.
S

EMPTY an empty fill.
E

style<*density*> a shaded pattern:
 style can be R | X | L
 density can be 1...5

See also the discussion of fill patterns for bars and blocks in VALUE= on page 171.

WHEN='B' | 'A'

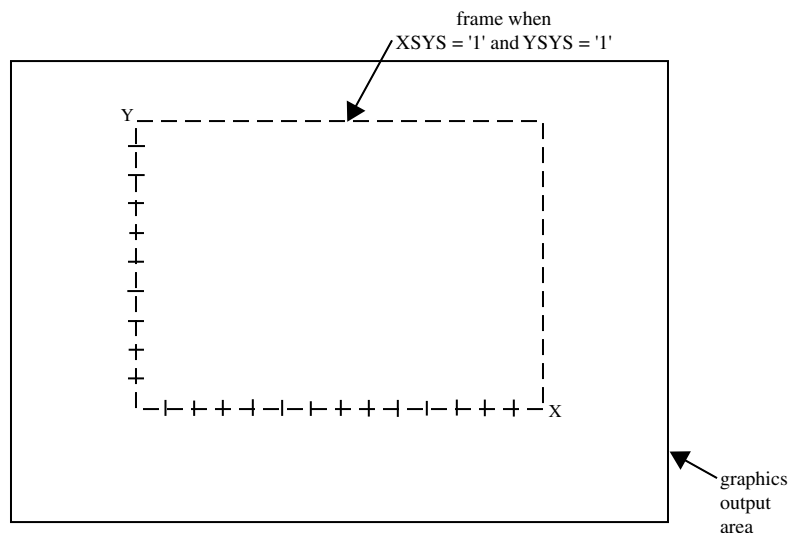
specifies when to draw the frame in relation to other procedure output. See “WHEN Variable” on page 666

XSYS=*'coordinate-system'*

YSYS=*'coordinate-system'*

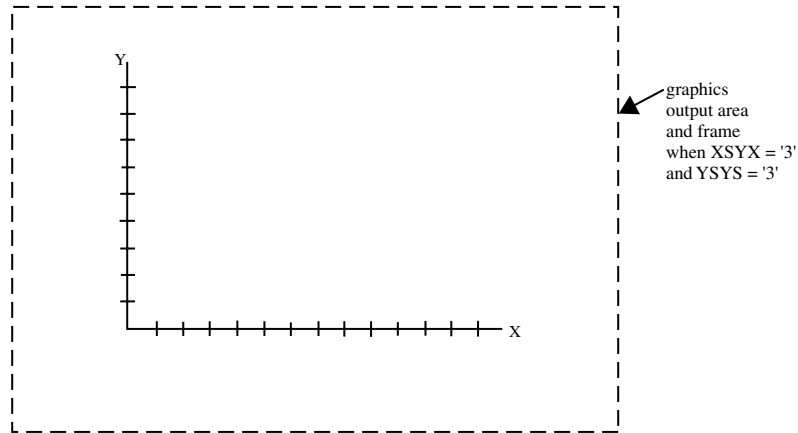
define the area to be enclosed by the frame. For example, if XSYS='1' and YSYS='1', the frame encloses the axis area as shown in Figure 25.7 on page 623. See “XSYS Variable” on page 670 and the YSYS variable on “YSYS Variable” on page 674 for an explanation of *coordinate-system*.

Figure 25.7 Frame Created When XSYS='1' and YSYS='1'



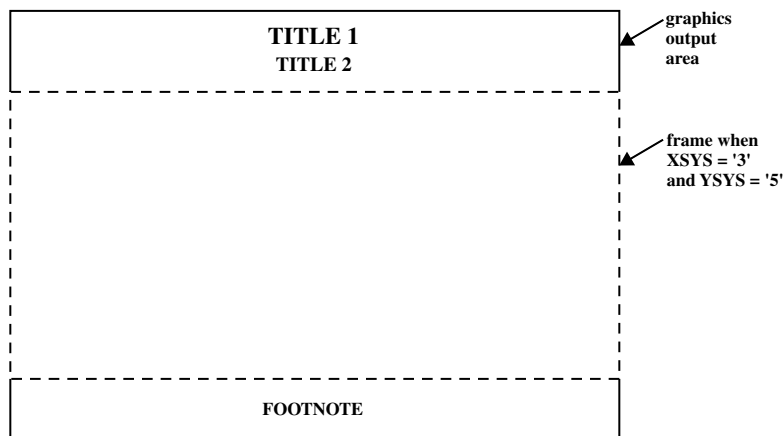
If `XSYS='3'` and `YSYS='3'`, the frame encloses the entire graphics output area, as shown in Figure 25.8 on page 624.

Figure 25.8 Frame Created When `XSYS='3'` and `YSYS='3'`



The values for `XSYS` and `YSYS` do not have to be the same. If `XSYS='3'` and `YSYS='5'`, the frame encloses the entire width of the graphics output area; however, vertically, the frame only encloses the procedure output area as shown in Figure 25.9 on page 624.

Figure 25.9 Frame Created When `XSYS='3'` and `YSYS='5'`



See “`XSYS` Variable” on page 670 and “`YSYS` Variable” on page 674 for an explanation of these variables and the areas that they affect.

Details

Use `FRAME` to simulate the `CBACK=` graphics option on devices (such as plotters) that do not support that option. For devices that do support the `CBACK=` graphics option, `FRAME` works in addition to that option. `FRAME` does not alter the (`XLAST`, `YLAST`) coordinates. See “`CBACK`” on page 266 for more information on `CBACK=`.

IMAGE Function

Displays an image in the graphics output from the current (X,Y) coordinates to the (X, Y) coordinates that are associated with the IMGPATH variable.

Updates: XLAST, YLAST

Syntax

FUNCTION='IMAGE';

Associated Variables

HTML=*'link-string'*

specifies the text that defines the link for drill-down.

IMGPATH=*fileref* | *'external-file'*

specifies the image file to be displayed in the graphics output. The syntax of external file specifications varies across operating environments.

STYLE = 'TILE' | 'FIT';

specifies how the image is to be applied to fill the specified area of the graphics output. The default value of TILE replicates the image to fill the area. The FIT value stretches a single instance of the image to fill the area.

X=*horizontal-coordinate*;

specifies the horizontal coordinate that determines the size of the image displayed in the graphics output.

Y=*vertical-coordinate*;

specifies the vertical coordinate that determines the size of the image displayed in the graphics output.

Z=*depth-coordinate*;

specifies the depth coordinate for 3D output.

ZSYS=*'coordinate-system'*

specifies the coordinate system for the Z variable. See “ZSYS Variable” on page 676 for an explanation of *coordinate-system*.

Details

The following example shows how the IMAGE function adds a single stretched instance of an image to the graphics output, beginning at the current coordinates and ending at the specified coordinates:

```
x=10; y=5; function='move' output;
x=35; y=15; imgpath='/images/gifs/picture.gif';
style='fit';
function='image'; output;
```

LABEL Function

Places text in the graphics output. Associated variables can control the color, size, font, base angle, and rotation of the characters displayed.

Updates: XLSTT, YLSTT

Syntax

FUNCTION='LABEL';

Associated Variables

ANGLE=0...360

specifies the baseline angle of the character string with respect to the horizontal. The pivot point is at (X, Y), and the rotation is in a counterclockwise direction.

CBORDER='color' | 'CTEXT'

draws a colored border around the text. *Color* can be any SAS/GRAPH color name.

CBOX='color' | 'CBACK'

draws a solid, colored box behind the text. *Color* can be any SAS/GRAPH color name.

COLOR='color'

specifies the color of the text. *Color* can be any SAS/GRAPH color name.

GROUP=*group-value*

MIDPOINT=*midpoint-value*

SUBGROUP=*subgroup-value*

specify coordinates for HBAR and VBAR charts from the GCHART procedure. Use these variables only with the data coordinate systems 1, 2, 7, and 8.

HSYS='coordinate-system'

specifies the coordinate system for the SIZE variable. See "HSYS Variable" on page 649 for an explanation of *coordinate-system*.

HTML='link-string'

specifies the text that defines the link for drill-down.

POSITION='text-position' | '0'

controls the text string placement and alignment. *Text-position* can be one of the characters 1 through 9, A through F, <, +, or >. Invalid or missing values default to POSITION='5'. POSITION should always be a character variable of length 1. For details, see "POSITION Variable" on page 656.

ROTATE=*rotation-angle*

specifies the rotation angle of each character in the string. It is equivalent to the ROTATE= option in the FOOTNOTE, NOTE, and TITLE statements.

SIZE=*height*

specifies the height of the text string. The SIZE variable units are based on the value of the HSYS variable.

STYLE=*font* | "*hardware-font-name*" | 'NONE'

specifies the font with which to draw the text that is specified by the TEXT variable. See "STYLE Variable (Fonts)" on page 661 for a description of the various font specifications.

TEXT=*text-string*'

specifies the text to be written. *Text-string* can be up to 200 characters. Define the TEXT variable with sufficient length to contain all of the characters in your text string. If you need longer strings, use separate observations and POSITION='0' to continue the text.

WHEN='B' | 'A'

specifies when to draw the text strings in relation to other procedure output. See "WHEN Variable" on page 666

X=*horizontal-coordinate*

Y=*vertical-coordinate*

Z=*depth-coordinate* (PROC G3D only)

XC=*character-type-horizontal-coordinate*'

YC=*character-type-vertical-coordinate*'

specify the start point of the text string. The Z variable can be used only with the G3D procedure. Optionally, you can modify the placement of the text string with the POSITION variable.

XSYS=*coordinate-system*'

specifies the coordinate system for the X or XC variable. Use the XC variable only with XSYS='2'. See "XSYS Variable" on page 670 for an explanation of *coordinate-system*.

YSYS=*coordinate-system*'

specifies the coordinate system for the Y or YC variable. Use the YC variable only with YSYS='2'. See "YSYS Variable" on page 674 for an explanation of *coordinate-system*.

ZSYS=*coordinate-system*'

specifies the coordinate system for the Z variable. See "ZSYS Variable" on page 676 for an explanation of *coordinate-system*.

MOVE Function

Moves the drawing pointer to a specific location without drawing a line.

Updates: XLAST, YLAST

Syntax

FUNCTION='MOVE';

Associated Variables

GROUP=*group-value*

MIDPOINT=*midpoint-value*

SUBGROUP=*subgroup-value*

specify coordinates for HBAR and VBAR charts from the GCHART procedure. Use these variables only with the data coordinate systems 1, 2, 7, and 8.

WHEN='B' | 'A'

specifies when to perform the move in relation to other procedure output. See also “WHEN Variable” on page 666.

X=horizontal-coordinate

Y=vertical-coordinate

Z=depth-coordinate (PROC G3D only)

XC='character-type-horizontal-coordinate'

YC='character-type-vertical-coordinate'

specify the coordinates to which the pen is to be moved. The Z variable can only be used with the G3D procedure.

XSYS='coordinate-system'

specifies the coordinate system for the X or XC variable. Use the XC variable only with XSYS='2'. See “XSYS Variable” on page 670 for an explanation of *coordinate-system*.

YSYS='coordinate-system'

specifies the coordinate system for the Y or YC variable. Use the YC variable only with YSYS='2'. See “YSYS Variable” on page 674 for an explanation of *coordinate-system*.

ZSYS='coordinate-system'

specifies the coordinate system for the Z variable. See “ZSYS Variable” on page 676 for an explanation of *coordinate-system*.

Details

Use MOVE to prepare for a DRAW command, a BAR command, or programming functions.

PIE Function

Draws pie slices in the graphics output.

Updates: XLAST, YLAST to coordinates for center of the slice.

Syntax

FUNCTION='PIE';

Associated Variables

ANGLE=starting-angle

specifies the starting angle of the slice arc. The default is 0.00 (horizontal) if the ANGLE variable is not specified for the first slice. After the first slice, the default is the ending angle of the slice arc just drawn if ANGLE=. (missing). Therefore, you can specify consecutive pie slices more easily by omitting the start and end calculations that are otherwise required. If you want the next slice to start at an

angle that is different from the ending angle of the previous slice, you must specify a value for the ANGLE variable.

COLOR=*'color'*

specifies the color of the pie slice, if a pattern is specified in the STYLE variable. If you specify STYLE='EMPTY', the COLOR variable also specifies the outline color of the pie slices. *Color* can be any SAS/GRAPH color name.

GROUP=*group-value*

MIDPOINT=*midpoint-value*

SUBGROUP=*subgroup-value*

specify coordinates for HBAR and VBAR charts from the GCHART procedure. Use these variables only with the data coordinate systems 1, 2, 7, and 8.

HSYS=*'coordinate-system'*

specifies the coordinate system for the SIZE variable. See "HSYS Variable" on page 649 for an explanation of *coordinate-system*.

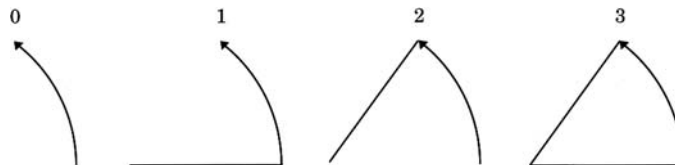
HTML=*'link-string'*

specifies the text that defines the link for drill-down.

LINE=0...3

specifies which slice line (or lines) to draw. See Figure 25.10 on page 629 for line values and their actions. LINE=0 draws only the outside of the arc and enables you to draw a circle.

Figure 25.10 LINE Values Used with the PIE Function



ROTATE=*rotation-angle*

specifies the angle of rotation or the delta angle of the slice arc. The default is 0.00.

For example, if you specify these statements, the slice arc that is drawn begins at 90 degrees (vertical) and ends at 135 degrees (90+45):

```
function='pie'; angle=90; rotate=45; output;
```

The ANGLE variable is internally updated to the end value, 135 degrees. The value is modified only internally. If a second PIE is used and the ANGLE variable contains a missing value, the start angle is assumed to be the previous end, or 135 degrees. The arc continues from that point.

If you specify the previous statements and then specify these statements, the slice begins at 135 degrees (the end angle from the previous slice) and extends another 45 degrees to the end point, 180 degrees.

```
function='pie'; angle=.; rotate=45; output;
```

This action repeats for every missing angle in the sequence.

SIZE=*radius*

specifies the radius of the circle being drawn. The SIZE variable uses units that are determined by the HSYS variable.

STYLE=*'fill-pattern'*

specifies the value of the pattern that fills the pie slices. *Fill-pattern* can be the following pie patterns:

PSOLID a solid fill.

PS

PEMPTY an empty fill.

PE

Pdensity<*style*<*angle*>> a shaded pattern:

density can be 1...5

style can be X | N

angle can be 0...360

For example, if **STYLE**=*'P5N15'*, a pie slice with a fill of parallel lines is produced. The fill uses the heaviest density to draw the lines, and the parallel lines are drawn at a 15-degree angle from perpendicular to the radius of the pie slice. See also the discussion of fill patterns for pie and star charts in **VALUE**= on page 174.

WHEN=*'B' | 'A'*

specifies when to draw the pie slice in relation to other procedure output. See “**WHEN Variable**” on page 666.

X=*horizontal-coordinate*

Y=*vertical-coordinate*

Z=*depth-coordinate* (PROC G3D only)

XC=*'character-type-horizontal-coordinate'*

YC=*'character-type-vertical-coordinate'*

define the center of the slice. The pivot point for all slices is the point referenced by X, Y, and Z (with PROC G3D only). The first **PIE** command that is issued sets the center at the (X,Y) value. If subsequent values for X and Y are missing, the coordinates of the center point are used.

XSYS=*'coordinate-system'*

specifies the coordinate system for the X or XC variable. Use the XC variable only with **XSYS**=*'2'*. See “**XSYS Variable**” on page 670 for an explanation of *coordinate-system*.

YSYS=*'coordinate-system'*

specifies the coordinate system for the Y or YC variable. Use the YC variable only with **YSYS**=*'2'*. See “**YSYS Variable**” on page 674 for an explanation of *coordinate-system*.

ZSYS=*'coordinate-system'*

specifies the coordinate system for the Z variable. See “**ZSYS Variable**” on page 676 for an explanation of *coordinate-system*.

See Also

“**CNTL2TXT Function**” on page 617

PIECNTR Function

Sets new center and radius values for later use by the **PIEXY** function but does not draw an arc.

Updates: XLAST, YLAST

Syntax

FUNCTION='PIECNTR';

Associated Variables

GROUP=*group-value*

MIDPOINT=*midpoint-value*

SUBGROUP=*subgroup-value*

specify coordinates for HBAR and VBAR charts from the GCHART procedure. Use these variables only with the data coordinate systems 1, 2, 7, and 8.

HSYS=*'coordinate-system'*

specifies the coordinate system for the SIZE variable. See “HSYS Variable” on page 649 for an explanation of *coordinate-system*.

SIZE=*radius*

specifies the new radius of the pie slice. The new radius is used by a subsequent PIEXY function. The HSYS variable determines the SIZE variable units.

WHEN='B' | 'A'

specifies when to draw the pie slice in relation to other procedure output. See “WHEN Variable” on page 666

X=*horizontal-coordinate*

Y=*vertical-coordinate*

Z=*depth-coordinate* (PROC G3D only)

XC=*'character-type-horizontal-coordinate'*

YC=*'character-type-vertical-coordinate'*

define the center and radius of the slice. All slices are referenced from that center. Use the Z variable only with the G3D procedure.

XSYS=*'coordinate-system'*

specifies the coordinate system for the X or XC variable. Use the XC variable only with XSYS='2'. See “XSYS Variable” on page 670 for an explanation of *coordinate-system*.

YSYS=*'coordinate-system'*

specifies the coordinate system for the Y or YC variable. Use the YC variable only with YSYS='2'. See “YSYS Variable” on page 674 for an explanation of *coordinate-system*.

ZSYS=*'coordinate-system'*

specifies the coordinate system for the Z variable. See “ZSYS Variable” on page 676 for an explanation of *coordinate-system*.

PIEXY Function

Calculates a point on the outline of the slice arc.

Updates: XLAST, YLAST

Syntax

FUNCTION='PIEXY';

Associated Variables

ANGLE=*rotation-angle*

specifies the angle of rotation when moving around the perimeter of a pie. The ANGLE variable determines the angle at which the point is located relative to 0 (the three o'clock position). The default is 0.00.

SIZE=*radius-multiplier*

determines the distance from the center of the slice to the point that is being calculated. The point's distance is the current value of the SIZE variable multiplied by the radius (that is, the SIZE variable) of the previously drawn slice. To position a graphics element inside the pie slice, set the SIZE variable to less than 1; to position it outside of the pie slice, set the SIZE variable to greater than 1. For example, if you specify these statements, the point calculated is 1.1 times the radius (where the radius is taken from the SIZE variable that is used with the previous FUNCTION='PIE' or FUNCTION='PIECNTR' observation).

```
function='piexy'; size=1.1; output;
```

WHEN='B' | 'A'

specifies when to update the internal coordinate pair (XLAST, YLAST) in relation to other procedure output. See "WHEN Variable" on page 666.

Details

PIEXY does not draw anything but places the calculated coordinates of the point in the internal coordinate pair (XLAST, YLAST). Then you can use XLAST and YLAST with other functions to perform other graphics actions, such as labeling pie slices. If you need to use the calculated position for a text function, use the SWAP or CNTL2TXT to put (XLAST, YLAST) into (XLSTT, YLSTT).

PIEXY assumes that a pie slice has been drawn or that FUNCTION='PIECNTR' has been used. Erroneous results can occur if a slice has not been drawn and PIEXY is invoked.

Figure 25.11 on page 632 shows a pie slice that is drawn with the PIE function. Figure 25.12 on page 633 shows a point beyond the arc that was calculated using the PIEXY function.

Figure 25.11 Pie Slice Drawn with the PIE Function

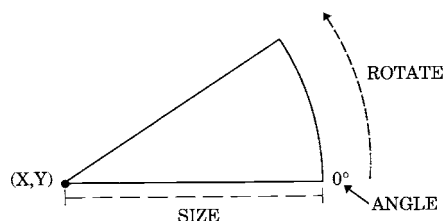
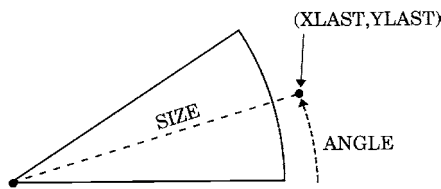


Figure 25.12 Point Calculated with the PEXY Function



See Also

“CNTL2TXT Function” on page 617

POINT Function

Places a single point at the (X, Y) coordinates in the color you specify. The point is one visible pixel in size.

Updates: XLAST, YLAST

Syntax

FUNCTION='POINT';

Associated Variables

COLOR=*color*

specifies the color of the point to be drawn. *Color* can be any SAS/GRAPH color name.

GROUP=*group-value*

MIDPOINT=*midpoint-value*

SUBGROUP=*subgroup-value*

specify coordinates when used with HBAR and VBAR charts from the GCHART procedure. Use these variables only with the data coordinate systems 1, 2, 7, and 8.

WHEN='B' | 'A'

specifies when to draw the point in relation to other procedure output. See “WHEN Variable” on page 666

X=*horizontal-coordinate*

Y=*vertical-coordinate*

Z=*depth-coordinate* (PROC G3D only)

XC=*character-type-horizontal-coordinate*

YC=*character-type-vertical-coordinate*

specify the coordinates of the point that is to be drawn. Use the Z variable only with the G3D procedure.

XSYS=*'coordinate-system'*

specifies the coordinate system for the X or XC variable. Use the XC variable only with XSYS='2'. See “XSYS Variable” on page 670 for an explanation of *coordinate-system*.

YSYS=*'coordinate-system'*

specifies the coordinate system for the Y or YC variable. Use the YC variable only with YSYS='2'. See “YSYS Variable” on page 674 for an explanation of *coordinate-system*.

ZSYS=*'coordinate-system'*

specifies the coordinate system for the Z variable. See “ZSYS Variable” on page 676 for an explanation of *coordinate-system*.

POLY Function

Specifies the beginning point of a polygon. Associated variables can define the fill pattern and color, as well as the line type that outlines the polygon.

Syntax

FUNCTION='POLY';

Associated Variables

COLOR=*'color'*

specifies the color of the interior of the polygon, if a pattern is specified for the STYLE variable. The outline color is specified with the POLYCONT function. *Color* can be any SAS/GRAPH color name.

GROUP=*group-value*

MIDPOINT=*midpoint-value*

SUBGROUP=*subgroup-value*

specify coordinates for HBAR and VBAR charts from the GCHART procedure. Use these variables only with data coordinate systems 1, 2, 7, and 8.

HTML=*'link-string'*

specifies the text that defines the link for drill-down.

LINE=1...46

specifies the line type that outlines the polygon. See “Specifying Line Types” on page 207 for an illustration of the line types.

SIZE=*thickness*

specifies a line thickness for the polygon

STYLE=*'fill-pattern'*

specifies the value of the pattern that fills the polygon. *Fill-pattern* can be the following map patterns:

MSOLID a solid pattern

MS

MEMPTY an empty pattern

ME

Mdensity<*style*<*angle*>> a shaded pattern:

density can be 1...5

style can be X | N

angle can be 0...360.

For example, if STYLE='MSOLID' for the POLY function, the fill area that is drawn by the POLYCONT sequence uses a solid fill. If STYLE='M5N15', the fill area uses a shaded fill of parallel lines. The *fill-pattern* value M5N15 specifies that the lines use the heaviest density, are parallel, and are drawn at a 15-degree angle from the horizontal. See also the discussion of fill patterns for maps in VALUE= on page 173.

WHEN='B' | 'A'

specifies when to begin the polygon in relation to other procedure output. See "WHEN Variable" on page 666

X=*horizontal-coordinate*

Y=*vertical-coordinate*

Z=*depth-coordinate* (PROC G3D only)

XC='character-type-horizontal-coordinate'

YC='character-type-vertical-coordinate'

specify the initial point of the polygon that is being created. Use the Z variable only with the G3D procedure.

XSYS='coordinate-system'

specifies the coordinate system for the X or XC variable. Use the XC variable only with XSYS='2'. See "XSYS Variable" on page 670 for an explanation of *coordinate-system*.

YSYS='coordinate-system'

specifies the coordinate system for the Y or YC variable. Use the YC variable only with YSYS='2'. See "YSYS Variable" on page 674 for an explanation of *coordinate-system*.

ZSYS='coordinate-system'

specifies the coordinate system for the Z variable. See "ZSYS Variable" on page 676 for an explanation of *coordinate-system*.

Details

Use POLY with POLYCONT to define and fill areas in the graphics output. POLY and POLYCONT do not update the (XLAST, YLAST) coordinates.

See Also

"POLYCONT Function" on page 635

POLYCONT Function

Continues drawing a polygon begun with the POLY function. POLYCONT specifies each successive point in the polygon definition.

Syntax

FUNCTION='POLYCONT';

Associated Variables

COLOR=*'color'*

specifies the polygon outline color. *Color* can be any SAS/GRAPH color name. You can specify an outline color only with the first POLYCONT command in the sequence; all subsequent POLYCONT commands ignore the COLOR variable. If you do not specify a color, the POLYCONT function uses the interior color that was specified with the POLY function.

GROUP=*group-value*

MIDPOINT=*midpoint-value*

SUBGROUP=*subgroup-value*

specify coordinates for HBAR and VBAR charts from the GCHART procedure. Use these variables only with the data coordinate systems 1, 2, 7, and 8.

WHEN='B' | 'A'

specifies when to draw the polygon in relation to other procedure output. See “WHEN Variable” on page 666

X=*horizontal-coordinate*

Y=*vertical-coordinate*

Z=*depth-coordinate* (PROC G3D only)

XC=*'character-type-horizontal-coordinate'*

YC=*'character-type-vertical-coordinate'*

specify a point on the outline of the polygon that is being created. Use the Z variable only with the G3D procedure.

XSYS=*'coordinate-system'*

specifies the coordinate system for the X and XC variable. Use the XC variable only with XSYS='2'. See “XSYS Variable” on page 670 for an explanation of *coordinate-system*.

YSYS=*'coordinate-system'*

specifies the coordinate system for the Y and YC variable. Use the YC variable only with YSYS='2'. See “YSYS Variable” on page 674 for an explanation of *coordinate-system*.

ZSYS=*'coordinate-system'*

specifies the coordinate system for the Z variable. See “ZSYS Variable” on page 676 for an explanation of *coordinate-system*.

Details

The polygon definition is terminated by a new POLY command or by any of these functions:

BAR

DRAW

DRAW2TXT

FRAME

LABEL
 MOVE
 PIE
 PIECNTR
 PIEXY
 POINT
 SYMBOL

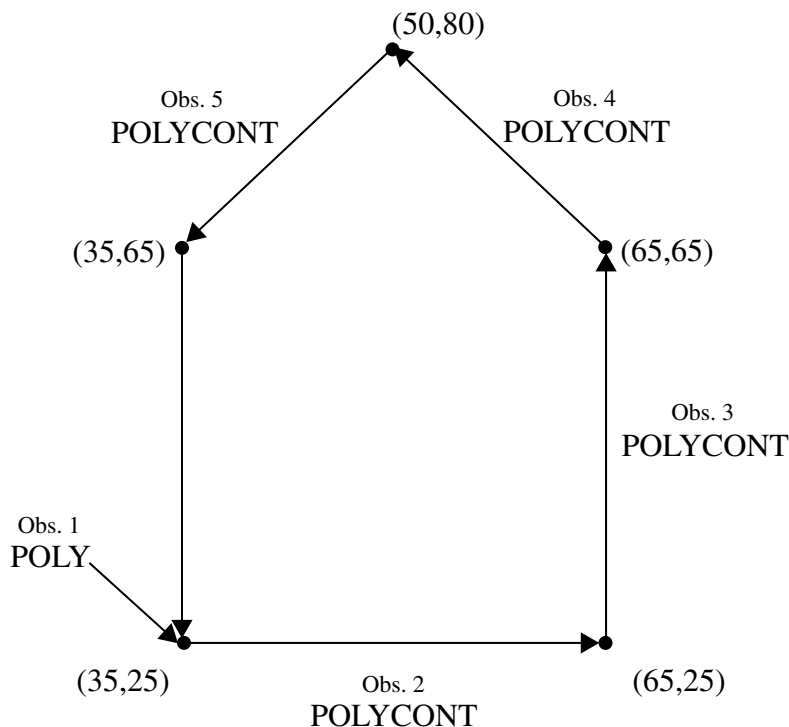
Use POLY and POLYCONT together to draw a polygon. The (X, Y) observation from the POLY function and the last (X, Y) observation from POLYCONT are assumed to connect. Thus, you are not required to respecify the first point. For example, these statements draw a pentagon like the one in Figure 25.13 on page 638:

```

data house;
  retain xsys ysys '3';
  length function $ 8;
  /* start at the lower left corner */
  function='poly'; x=35; y=25; output;
  /* move to the lower right corner */
  function='polycont'; x=65; y=25; output;
  /* move to the upper right corner */
  function='polycont'; x=65; y=65; output;
  /* move to the center top*/
  function='polycont'; x=50; y=80; output;
  /* move to the upper left corner and complete the figure */
  function='polycont'; x=35; y=65; output;
run;

proc ganno anno=house;
run;
quit;

```

Figure 25.13 Pentagon Produced with the POLY and POLYCONT Functions

Missing values for the X and Y variables that are specified with POLYCONT are interpreted differently from the way that they are interpreted with the other functions. Other functions use the missing values to request a default value. POLYCONT interprets a missing value as a discontinuity (that is, a hole) in the polygon. If you are not using the data coordinate system and you specify an X or Y value of -999 in a POLYCONT observation, the default of (XLAST, YLAST) is used. Missing values indicate holes and are handled identically in the Annotate facility and the GMAP procedure. See “Displaying Map Areas and Response Data” on page 1005 for more information on handling missing values.

POP Function

Removes the (XLAST, YLAST) and (XLSTT, YLSTT) values from the LIFO stack and updates the internal coordinate pairs with the retrieved values.

Updates: (XLAST, YLAST) and (XLSTT, YLSTT)

Syntax

FUNCTION='POP';

Details

Use POP when you want to access the values of (XLAST, YLAST) and (XLSTT, YLSTT) that you most recently stored with the PUSH function. See the PUSH function for a description of the LIFO stack.

PUSH Function

Adds current (XLAST, YLAST) and (XLSTT, YLSTT) values to the LIFO stack.

Syntax

FUNCTION='PUSH';

Details

The LIFO (last-in-first-out) stack is a storage area where you can keep internal coordinate values for later use by utility functions without recalculating those values. LIFO stacks manage the stored data so that the last data stored in the stack is the first data removed from the stack.

Use the stack to save the current values of (XLAST, YLAST) and (XLSTT, YLSTT) and use them with functions later in the DATA step. You store and retrieve these values from the stack with the PUSH and POP functions. The PUSH function copies the current values of XLAST, YLAST, XLSTT, and YLSTT onto the stack. The POP function copies values from the stack into XLAST, YLAST, XLSTT, and YLSTT.

SWAP Function

Exchanges values of (XLAST, YLAST) with (XLSTT, YLSTT) and vice versa.

Updates: (XLAST, YLAST) and (XLSTT, YLSTT)

Syntax

FUNCTION='SWAP';

Details

Use SWAP when you want to use both the (XLAST, YLAST) and (XLSTT, YLSTT) coordinates for text and nontext functions, respectively.

SYMBOL Function

Places symbols in the graphics output. Associated variables can specify the color, font, and height of the symbols displayed.

Updates: XLSTT, YLSTT

Syntax

FUNCTION='SYMBOL';

Associated Variables

CBORDER='color' | 'CTEXT'

draws a colored border around the text. *Color* can be any SAS/GRAPH color name.

CBOX='color' | 'CBACK'

draws a solid, colored box behind the text. *Color* can be any SAS/GRAPH color name.

COLOR='color'

specifies the symbol color. *Color* can be any SAS/GRAPH color name. The COLOR variable behaves in the same way as the COLOR= option in the SYMBOL statement. See COLOR= on page 185 for details

GROUP=*group-value*

MIDPOINT=*midpoint-value*

SUBGROUP=*subgroup-value*

specify coordinates for HBAR and VBAR charts from the GCHART procedure. Use these variables only with the data coordinate systems 1, 2, 7, and 8.

HSYS='coordinate-system'

specifies the coordinate system for the SIZE variable. See "HSYS Variable" on page 649 for an explanation of *coordinate-system*.

HTML='link-string'

specifies the text that defines the link for drill-down.

SIZE=*height*

specifies the height of the symbol that is being drawn, using units determined by the HSYS variable. The SIZE variable is equivalent to the HEIGHT= option in the SYMBOL statement. See HEIGHT= on page 187 for details.

STYLE='font' | "*hardware-font-name*" | 'NONE';

specifies the font that is used to draw the symbol that is specified by the TEXT variable. See "STYLE Variable (Fonts)" on page 661 for a description of the various font specifications.

When the STYLE variable is used with the SYMBOL function, it behaves the same as the FONT= option in the SYMBOL statement. By default, no font is specified and the symbol that is specified by the TEXT variable is taken from the special symbol table. If you use STYLE to specify a symbol font, such as Marker, the string that is assigned by the TEXT variable is the character code for a symbol. If you use STYLE to specify a text font, such as Swiss, the string assigned by the TEXT variable is displayed as text. See FONT= on page 186 for details.

TEXT='special-symbol' | 'text-string';

specifies the symbol to be displayed. *Special-symbol* can be up to eight characters long. Values for *special-symbol* are those described in the VALUE= option of the SYMBOL statement and are illustrated in VALUE= on page 199.

For client-side rendering using ActiveX, the following values are supported: plus, X, star, square, diamond, triangle, dot, circle, ", #, \$, %, =. If a symbol is not supported, a plus sign (+) is drawn instead.

For client-side rendering using Java, the following values are supported: plus, X, star, square, diamond, triangle, dot (draws a circle), circle, *, +, >. If a symbol is not supported, a plus sign (+) is drawn instead.

If you also specify a text font with the STYLE variable, you can specify a text string that is displayed as the symbol. The maximum length for *text-string* is 200 characters.

When the TEXT variable is used with the SYMBOL function, it behaves the same as the VALUE= option in the SYMBOL statement. See VALUE= on page 199 for details.

WHEN='B' | 'A'

specifies when to draw the symbols in relation to other procedure output. See “WHEN Variable” on page 666

Y=*vertical-coordinate*

Z=*depth-coordinate* (PROC G3D only)

XC='character-type-horizontal-coordinate'

YC='character-type-vertical-coordinate'

specify the point at which the symbol is placed. Use the Z variable only with the G3D procedure.

XSYS='coordinate-system'

specifies the coordinate system for the X or XC variable. Use the XC variable only with XSYS='2'. See “XSYS Variable” on page 670 for an explanation of *coordinate-system*.

YSYS='coordinate-system'

specifies the coordinate system for the Y or YC variable. Use the YC variable only with YSYS='2'. See “YSYS Variable” on page 674 for an explanation of *coordinate-system*.

ZSYS='coordinate-system'

specifies the coordinate system for the Z variable. See “ZSYS Variable” on page 676 for an explanation of *coordinate-system*.

Details

SYMBOL is similar to the LABEL function with these exceptions:

- SYMBOL draws symbols. If you do not specify a font, SYMBOL can use the symbols found in Figure 7.21 on page 202.
- The text cannot be rotated or angled.
- The text string cannot be longer than eight characters.
- The text string is always centered with respect to *x* and *y*.

TXT2CNTL Function

Copies the values (XLSTT, YLSTT) to (XLAST, YLAST), replacing previous values of (XLAST, YLAST).

Syntax

FUNCTION='TXT2CNTL';

Details

TXT2CNTL allows nontext functions to use the ending position of a text string as a starting or ending point.

Annotate Variables

When an Annotate data set is processed, the Annotate facility looks at the values of specific variables in order to draw graphics. This section describes all of the Annotate variables in alphabetical order. Not all variables are used with all functions. Refer to the description of the individual functions in “Annotate Functions” on page 615 for more information about how each variable is used with each function. For a summary of Annotate variables and their uses, see Table 24.1 on page 591.

ANGLE Variable

Specifies the angle at which the graphics output is drawn.

Type: numeric

Default: function dependent

Syntax

ANGLE=0...360;

Functions

The ANGLE variable is function dependent.

If function is...	then the ANGLE variable specifies...
LABEL	the baseline angle of the character string with respect to the horizontal. With the LABEL function, the pivot point is at (X,Y) and the direction of rotation is counterclockwise. The default value is 0.
PIE	the starting angle of the slice arc, measured counterclockwise. The default for the first PIE function is ANGLE=0 (horizontal, or 3:00 position), or is the ending point of the arc of the previous slice. Specify a value for the ANGLE variable if you want the next slice to start at an angle that is different from the edge of the previous slice, or if you want the first slice to start at an angle other than horizontal.
PIEXY	the angle that works with the SIZE variable to establish the new XLAST, YLAST point relative to the last pie element established with the PIE or PIECNTR functions. The angle is measured counterclockwise starting at the 3:00 position. The default value is 0.

CBORDER Variable

Draws a colored border around text or symbols.

Type: character

Length: 8 for color codes and up to 64 for color names

See also: CBOX

Syntax

CBORDER=*color* | 'CTEXT';

color

specifies the color that fills the box. The *color* value can be any SAS/GRAPH color name. See Chapter 6, "SAS/GRAPH Colors and Images," on page 91 for more information about specifying colors.

Specifying a null value for the *color* value (CBOX=' ') cancels the CBOX variable.

CTEXT

draws the border in the same color as the text or symbol. The text color is determined by (1) the COLOR variable or (2) the CTEXT=graphics option or (3) the first color in the colors list.

Functions

You can use the CBORDER variable with these functions:

LABEL

SYMBOL

Details

Once you have specified CBORDER, it remains in effect for all subsequent observations that use the LABEL or SYMBOL function and draws a border around all text or symbols. To turn off the border for subsequent text or symbols, specify CBORDER=' '.

To fill the area defined by CBORDER, use the CBOX variable in conjunction with CBORDER.

CBOX Variable

Draws a solid box behind the text or symbol and fills the box with the specified color.

Type: character

Length: 8 for color codes and up to 64 for color names

See also: CBORDER

Syntax

```
CBOX='color' | 'CBACK';
```

color

specifies the color that fills the box. *Color* is any SAS/GRAPH color name. See Chapter 6, “SAS/GRAPH Colors and Images,” on page 91 for more information about specifying colors.

Specifying a null value for *color* (CBOX=' ') cancels the CBOX variable.

CBACK

fills the box with the same color as the background color of the graph. The background color is either (1) the color specified by the CBACK= graphics option or (2) the default background color for the device.

Functions

You can use the CBOX variable with these functions:

LABEL

SYMBOL

Details

Once you have specified CBOX, it remains in effect for all subsequent observations that use the LABEL or SYMBOL function.

The color of the text or symbol within the box is controlled by the COLOR variable.

By default, the solid box has no border. To add a colored border to the box, use the CBORDER variable in conjunction with CBOX.

COLOR Variable

Specifies the color used by the function.

Type: character

Length: 8 for color codes and up to 64 for color names

Default:

- 1 first color in colors list of the COLORS= graphics option
- 2 first color in device's default colors list.

Syntax

COLOR=*color*;

color

specifies any SAS/GRAPH color name. See Chapter 6, "SAS/GRAPH Colors and Images," on page 91 for more information about specifying colors.

Functions

The COLOR variable is function dependent.

If function is...	then the COLOR variable specifies...
BAR	the color that outlines and, optionally, fills the bar if a pattern is specified in the STYLE (patterns)"STYLE Variable (Patterns)" on page 662 variable. If no pattern is specified, the <i>color</i> value is applied only to the outline of the bar.
DRAW, DRAW2TXT	the color of the line.
FRAME	the color of the outline of the frame. If a fill pattern is specified, <i>color</i> also determines the color of the inside of the frame.
LABEL	the color of the text.
PIE	the color for the pie slice if a pattern is specified with the STYLE (patterns)"STYLE Variable (Patterns)" on page 662 variable. If no pattern is specified, <i>color</i> determines the color of the outline of the pie slice.
POINT	the color of the point.
POLY	the fill color for the interior of the polygon if a pattern is specified with the STYLE variable. If the STYLE variable is missing or EMPTY, <i>color</i> is ignored. Use the POLYCONT function to specify the outline color.
POLYCONT	the color that outlines the polygon when used with the first POLYCONT function. COLOR is ignored for subsequent POLYCONT functions in the POLYCONT sequence.
SYMBOL	the color that draws the symbol.

FUNCTION Variable

Specifies a graphics command or programming function for the Annotate facility to perform.

Type: character

Length: 8

Default: LABEL

Syntax

FUNCTION='function-name';

function-name

specifies the name of an Annotate function. The *function-name* value can be any of the following.

BAR	draws and, optionally, fills a rectangle.
CNTL2TXT, DRAW2TXT	copies (XLAST, YLAST) to (XLSTT, YLSTT), overwriting the previous values of (XLSTT, YLSTT).
COMMENT	places comments in your data set. The observation is ignored when the data set is processed.
DEBUG	writes the values of all Annotate variables to the SAS log before and after the next observation.
DRAW	draws a line in the graphics output.
FRAME	draws a border around the area defined by XSYS and YSYS and specifies a background color for the framed area .
IMAGE	displays an image in the graphics output from the current (X,Y) coordinates to the coordinates that are associated with the IMGPATH variable.
LABEL	draws text and is the default for the FUNCTION variable.
MOVE	moves to the specified point (does not draw a line).
PIE	draws a pie slice, arc, or circle that can be filled.
PIECNTR	sets new center and radius values. The PIEXY function can use this information in a later observation.
PIEXY	returns the coordinates of a point on a pie slice. Other functions can use this information in a later observation.
POINT	draws a point.
POLY	begins drawing a polygon (first vertex). Use the POLYCONT function in successive observations to supply the remaining vertices.
POLYCONT	continues drawing a polygon.
POP	gets values from the LIFO stack and changes the current value of (XLAST, YLAST) and (XLSTT, YLSTT) to those values.

PUSH	puts the current values for (XLAST, YLAST) and (XLSTT, YLSTT) in the LIFO stack.
SWAP	exchanges the values of (XLAST, YLAST) and (XLSTT, YLSTT).
SYMBOL	draws a symbol. See Figure 7.21 on page 202 for a list of the symbols.
TXT2CNTL	copies the values (XLSTT, YLSTT) to (XLAST, YLAST), overwriting the previous values of (XLAST, YLAST).

All other variables in the observation that contain the function act as parameters for the action. For a detailed description of each function and the Annotate variables that can be used in conjunction with it, see “Annotate Functions” on page 615.

GROUP Variable

Positions graphics elements on the bars of a vertical or horizontal bar chart drawn using the GROUP= option in the GCHART procedure.

Type: Numeric or character; must match the type of the GROUP= variable used in the GCHART procedure.

Length: Should match the length of GROUP= variable in the GCHART procedure.

Default: none

Restriction: Used only with vertical or horizontal bar charts produced by the GCHART procedure.

Syntax

GROUP=*group-value*;

group-value

references value(s) of the variable that is identified by the GROUP= option in the GCHART procedure either as a variable name or as an explicit data value.

Group-value can be one of the following:

group-variable the name of a group variable.

group-data-value a specific numeric data value.

'*group-data-value*' a specific character data value.

To annotate all the bars in a horizontal or vertical bar chart, specify a variable name. To annotate a bar chart for a specific value of the GROUP variable, specify a specific value.

Functions

You can use the GROUP variable only with the data coordinate systems 1, 2, 7, and 8, and with these functions:

BAR

DRAW

LABEL

MOVE

PIE

PIECNTR

POINT

POLY

POLYCONT

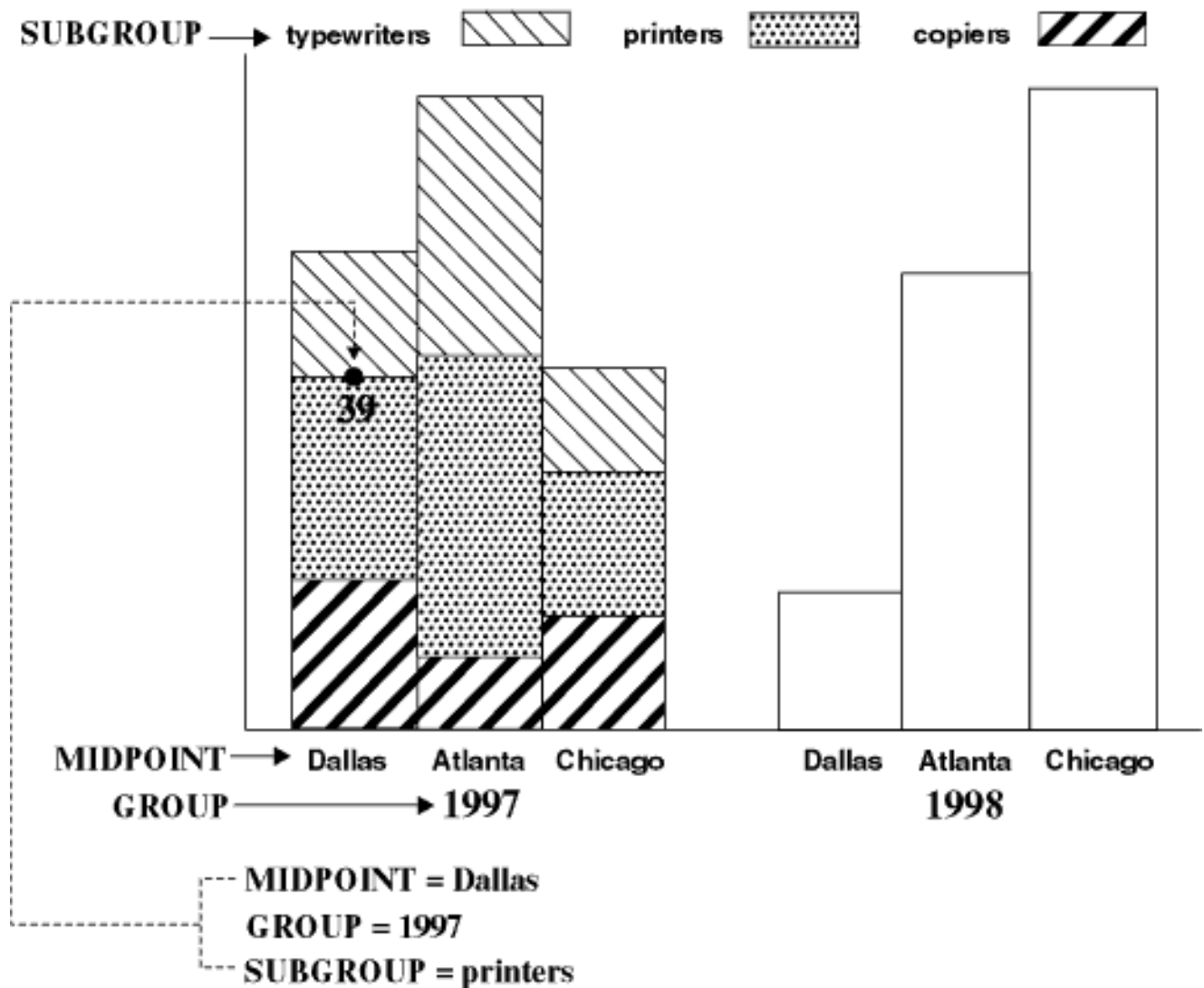
SYMBOL

Details

Using the GROUP variable is similar to using the X and Y variables with data system coordinates to position graphics elements in a vertical or horizontal bar chart.

Figure 25.14 on page 649 shows how the GROUP variable works with the SUBGROUP and MIDPOINT variables to label the bars of a vertical bar chart.

Figure 25.14 Using the GROUP Variable to Position a Label in a Bar Chart



The label showing the number of units that were sold in Dallas in the year 1997 is positioned by the values that are assigned to these Annotate variables:

- GROUP=YEAR (where YEAR is a variable in the GCHART data set)
- MIDPOINT=CITY (where CITY is a variable in the GCHART data set)
- SUBGROUP=ITEM (where ITEM is a variable in the GCHART data set).

HSYS Variable

Defines the coordinate system and area of the output used by the SIZE variable to display the Annotate graphics. Additionally, you can use the HSYS variable with client-side annotation with Java or ActiveX to control the markersize and linesize for the BAR, DRAW, DRAW2TXT, POLY, and SYMBOL functions.

Type: character

Length: 1

Default: 4

Syntax

HSYS='coordinate-system';

coordinate-system

specifies a value that represents a coordinate system. Values can be 1 through 9 and A through C as shown in the following table:

Absolute Systems	Relative Systems	Coordinate System Units
1	7	percentage of data area
2	8	data values
3	9	percentage of graphics output area
4	A	cell in graphics output area
5	B	percentage of procedure output area
6	C	cell in procedure output area

These values are also used by the XSYS and YSYS variables. See “Coordinate Systems” on page 596 for a description of the areas and coordinate systems.

Functions

You can use HSYS with these functions, all of which also use the SIZE variable:

DRAW

DRAW2TXT

FRAME

LABEL

PIE

PIECNTR

SYMBOL

Details

The coordinate system that you specify with the HSYS variable affects how the function interprets the value of the SIZE variable. For example, if you use HSYS='3' and SIZE=10 with the DRAW function, the thickness of the line is 10 percent of the graphics output area. If you use HSYS='1' and SIZE=10 with DRAW, the thickness of the line is 10 percent of the data area.

HTML Variable

Defines a link in the HTML file created for a drill-down graph. This link is associated with an area of the graph and contains valid HTML syntax that can point to a report or another graph that you want to display when the user drills down on the area.

Type: character

Length: up to 1024

Default: none

Syntax

HTML=*'link-string'*;

link-string

specifies the text that defines the link for drill-down. For more information about drill-down graphs and how to specify the link string, see “Adding Links with the HTML= and HTML_LEGEND= Options” on page 574.

Functions

You can use the HTML variable with these functions:

BAR
 FRAME
 IMAGE
 LABEL
 PIE
 POLY
 SYMBOL

Details

Use a LENGTH statement to set the length of the HTML variable to the longest string you need for the link string. Be sure to set the HTML value to a null if you continue writing observations to the annotate data set after you are done assigning links. For example, the following code defines link information for two squares, but then sets the HTML variable to null when drawing a frame; otherwise the background area within the frame will use the link information from the last defined HTML value and become a hot zone in the graph.

```
data squares;
  length function style color $ 8
         html text $ 15;
  xsys='3'; ysys='3';

  /* draw a green square */
  color='green';
  function='move'; x=10; y=65; output;
  function='bar'; x=30; y=95; style='solid';
```

```

        html='href=green.gif'; output;

        /* draw a red square */
        color='red';
        function='move'; x=60; y=65; output;
        function='bar'; x=80; y=95;
        html='href=red.gif'; output;

        /* draw a blue frame */
        function='frame'; color='blue'; style='empty';
        /* set null link for background area in frame */
        html=''; output;
run;
```

IMGPATH Variable

Specifies an image to be displayed from the current (X,Y) coordinates to the (X,Y) coordinates that are associated with this variable.

Type: character

Length: 255

Syntax

IMGPATH = *fileref* | '*external-file*';

fileref

specifies an existing fileref that points to an external image file.

external-file

specifies the full path or full file name of an external image file. The format of the external file specification varies between operating environments.

Details

The IMGPATH variable can be used only with the “IMAGE Function” on page 625.

The manner in which the specified image is to be displayed is determined by the “STYLE Variable (Images)” on page 662.

LINE Variable

Controls the drawing of a line by determining either the type of line to draw or the relative position of the line.

Type: numeric

Default for all functions: 1

Syntax

LINE=*line-type*;

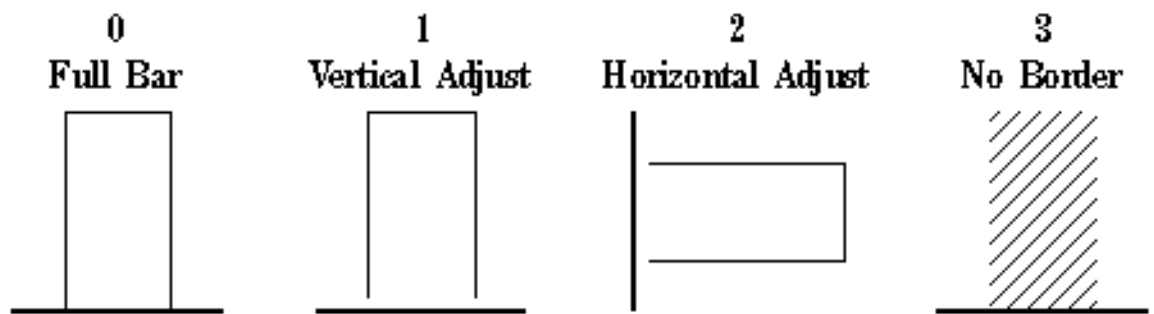
Functions

The behavior and syntax of the LINE variable is function-dependent.

BAR

In the BAR function, valid values for the LINE variable can be 0, 1, 2, or 3. These values determine how the outline of the bar is to be drawn, as shown in the following figure. A value of 0 draws the outline all the way around the bar. A value of 1 draws the outline only on the vertical sides of the bar. A value of 2 draws the outline only on the horizontal sides of the bar. A value of 3 draws no outline.

Figure 25.15 LINE Values for Bars



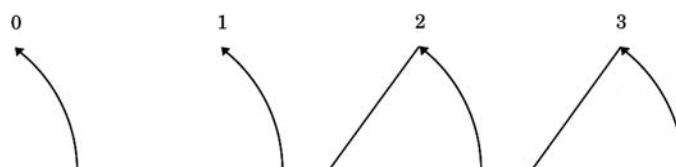
DRAW, DRAW2TXT, FRAME, POLY

Valid values are whole numbers from 0 to 46. A value of 0 specifies that the line not be drawn. A value of 1 specifies a solid line. The remaining values specify different segmented lines, as illustrated in Figure 7.22 on page 208.

PIE

Valid values are 0, 1, 2, or 3. The value specifies which lines of a pie slice are to be drawn for the current arc, as shown in Figure 25.16 on page 653.

Figure 25.16 LINE Values Used with the PIE Function



MIDPOINT Variable

Positions graphics elements on the bars of a vertical or horizontal bar chart drawn by the GCHART procedure.

Type: Numeric or character; must match the type of the MIDPOINT= variable in the GCHART procedure.

Length: Should match the length of the MIDPOINT= variable in the GCHART procedure.

Default: none

Restriction: Used only with vertical or horizontal bar charts produced by the GCHART procedure.

Syntax

MIDPOINT=*midpoint-value*;

midpoint-value

references midpoint data value(s) in the GCHART procedure either as a variable name or as an explicit data value. *Midpoint-value* can have one of the following forms:

midpoint-variable the name of a midpoint variable.

midpoint-data-value a specific numeric data value.

'*midpoint-data-value*' a specific character data value.

Generally, specify a variable name if you want to annotate all of the bars in a horizontal or vertical bar chart. To annotate a bar chart for a specific value of the MIDPOINT variable, specify a specific value.

Functions

You can use the MIDPOINT variable only with the data coordinate systems 1, 2, 7, and 8, and with these functions:

BAR

DRAW

LABEL

MOVE

PIE

PIECNTR

POINT

POLY

POLYCONT

SYMBOL

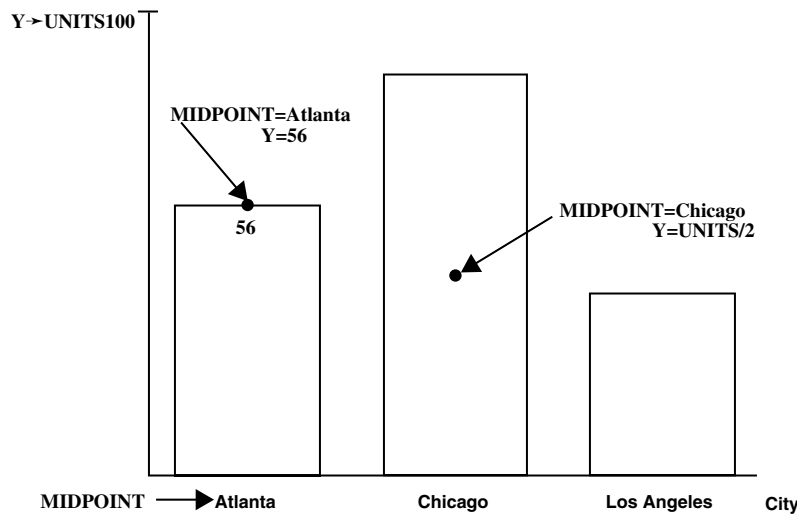
Details

Using the MIDPOINT variable is similar to using the X and Y variables to position graphics elements in a vertical or horizontal bar chart when using data system coordinates. For example, suppose you produce a vertical bar chart in which the chart variable CITY produces a bar for each city in a data set. The height of each bar is determined by the value of the SUMVAR= variable, UNITS.

You can label these bars by assigning the chart variable CITY to the Annotate MIDPOINT variable. The MIDPOINT variable provides the x coordinate for the label. By default, Annotate assigns the statistic variable, in this case the SUMVAR= variable, UNITS, to the Annotate Y variable, which provides the y coordinate for the label.

Figure 25.17 on page 655 shows how the values of the MIDPOINT and Y variables position the label that shows the number of units sold in Atlanta. The value, which is calculated and printed by the LABEL function, is 56.

Figure 25.17 Using the MIDPOINT Variable to Position a Label in a Bar Chart



The labels in this figure are positioned by the values that are assigned to these Annotate variables:

- MIDPOINT=CITY (where CITY is the chart variable); the MIDPOINT variable provides the horizontal coordinate in the vertical bar chart.
- Y=UNITS (where UNITS is the SUMVAR= variable); the Y variable provides the vertical coordinate. By specifying $Y=\text{units}/2$, you can vertically center the label in the bar.

Note: In a horizontal bar chart, the MIDPOINT variable controls the y coordinate and the statistic variable controls the x coordinate. △

CAUTION:

Be careful when using MIDPOINT and X and Y variables in the same data set. Using the MIDPOINT and X variables in an Annotate data set that is used to annotate a VBAR chart or the MIDPOINT and Y variables in the same data set used to annotate an HBAR chart can cause unexpected results. When annotating a VBAR chart, the

Annotate facility uses the MIDPOINT variable as the horizontal coordinate if it exists in the Annotate data set and ignores the X variable. Consequently, you should use the MIDPOINT variable as the horizontal coordinate for all observations in an Annotate data set if you use it for one.

A similar behavior occurs if you use both the MIDPOINT and Y variables in an Annotate data set that is used to annotate HBAR charts. The MIDPOINT variable is always used, regardless of whether it has a missing value, and the Annotate facility ignores the Y variable. In this case, as well, use the MIDPOINT variable for the vertical coordinate for all observations in an Annotate data set if you use it for one. Δ

POSITION Variable

Controls placement and alignment of a text string specified by the LABEL function.

Type: character

Length: 1

Default: 5

Syntax

POSITION=*'text-position'* | '0';

text-position

specifies the placement of the text string in relation to the position that is defined by the X and Y variables. *Text-position* can be one of the characters 1 through 9, A through F, <, +, or >. These characters represent the positions that are described in the following table:

Position	Right Aligned	Centered	Left Aligned
One cell above	1	2	3
Centered	4 <	5 +	6 >
One cell below	7	8	9
Half cell above	A	B	C
Half cell below	D	E	F

These positions are illustrated in Figure 25.19 on page 658.

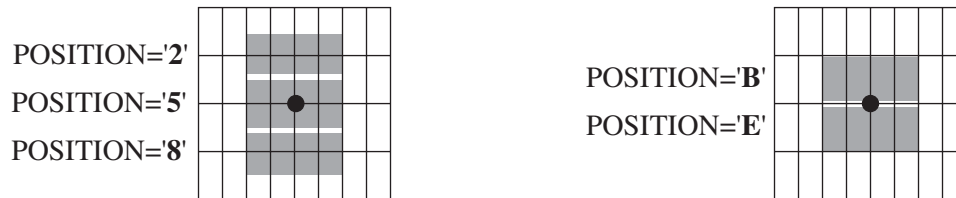
'0'

specifies a pause in the string in order to change an attribute, such as the color of the text.

Details

Stacking text strings To stack text strings, specify a different position value of each string. Figure 25.18 on page 657 shows two ways to stack text.

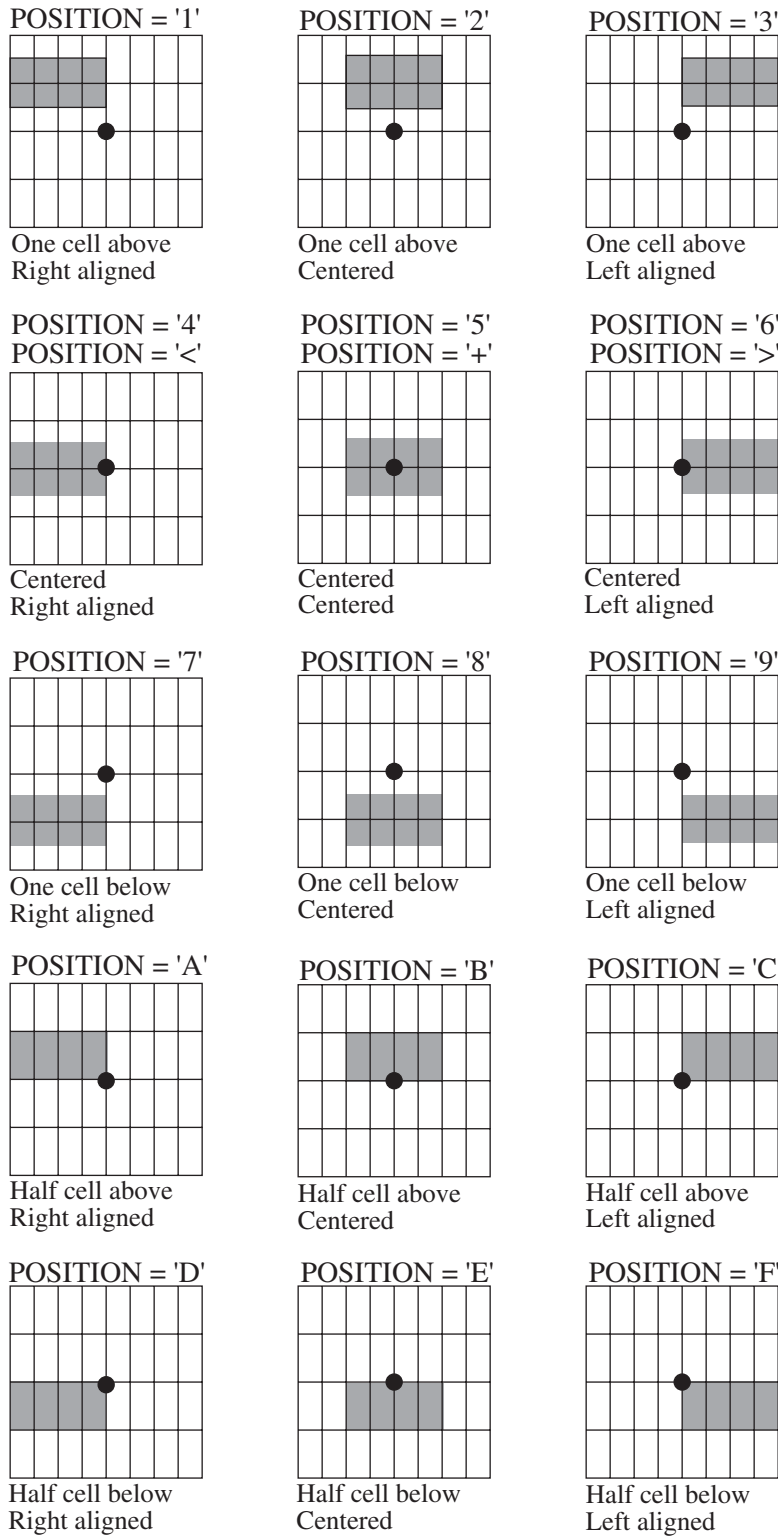
Figure 25.18 Combining POSITION Values to Stack Text



Positioning numeric labels The <, +, and > positions perform the same function as 4, 5, and 6, respectively, but are recommended only for labels that are numbers. The <, +, and > positions are especially useful when you are labeling a horizontal bar chart. You can use <, +, or > if the numbers in your font are significantly smaller than the text and you are having trouble centering labels. If the numbers in your font are the same height or close to the same height as the text, you can use positions 4, 5, and 6 to center the labels.

Note: You cannot stack <, +, and > positions as you can 4, 5, and 6 positions. △

Figure 25.19 Effect of POSITION Values on Text Strings

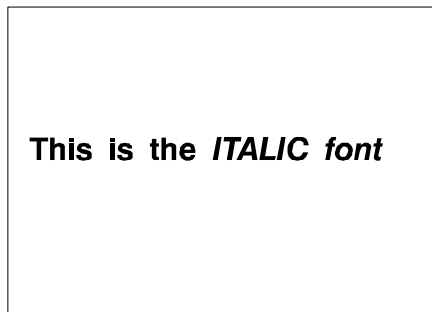


Changing attributes in the middle of a text string 0 is a special value to use when you want to pause and then continue a text string. With this value you can change colors, fonts, and so on in the middle of a line, while retaining the exact position of the text at

the pause. When POSITION='0', the combined text string is left-justified beginning at the point that is defined by the X and Y variables. However, you must define missing values for X for the continuation string. The following Annotate data set changes the font in the middle of the string. The result is shown in Figure 25.20 on page 659.

```
data anno;
  length style $ 8 text $ 12;
  xsys='3'; ysys='3'; hsys='3'; x=5; y=50;
  style='swissb'; size=10; text='This is the';
  position='0'; output;
  x=.; style='swissbi'; text=' ITALIC font';
  output;
run;
```

Figure 25.20 Using POSITION='0' to Change the Attributes of a Text String



ROTATE Variable

Specifies the angle at which to rotate the graphics element.

Type: numeric

Default: 0.00

Syntax

ROTATE=*rotation-angle*;

Functions

The ROTATE variable is function dependent.

If function is...	then the variable...
PIE	specifies the sweep of the generated arc that begins at the angle that is specified by the ANGLE variable that is used with the PIE function.
LABEL	rotates the individual text characters with respect to the baseline.

SIZE Variable

Determines the size of the graphics element with which it is used.

Type: numeric

Length: 8

Default: 1.00 (2 when HSYS=3)

Syntax

SIZE=*size-factor*;

Functions

The SIZE variable is function dependent.

If function is...	then the variable...
DRAW, DRAW2TXT, FRAME, POLY, or POLYCONT	determines the thickness of the line being drawn.
LABEL	specifies the height of the text.
PIE or PIECNTR	determines the radius of the pie.
PIEXY	sets the radius multiplier.
SYMBOL	selects the height of the symbol.

Details

The SIZE variable uses the coordinate system that is specified by the “HSYS Variable” on page 649, which specifies the type of coordinate system used to generate the graph.

As the thickness of the line increases, it may be impossible to center around a given coordinate. For example, if you specify a thickness of value 2 and HSYS='4', the first line is drawn at the (X, Y) coordinates. The second is drawn slightly above the first. The exact amount varies by device, but it is always one pixel in width. A thickness of value 3 produces one line above, one line at, and one line below the (X, Y) coordinate position.

The `SIZE` variable is equivalent to the `HEIGHT=` option in the `SYMBOL` statement. See `HEIGHT=` on page 187 for details.

See Figure 25.6 on page 621 for examples of line thicknesses.

Figure 25.21 Sample Line Thicknesses Used with the `SIZE` Variable



STYLE Variable (Fonts)

Specifies a font for text or symbols produced by the `LABEL` or `SYMBOL` functions.

Type: character

Length: Depends on specification.

Default: default hardware font

Not supported by: ActiveX (Partial), Java

Syntax

`STYLE=font` | `"hardware-font-name"` | `'NONE'`;

font

specifies a font. *Font* can be either the name of a software font that is stored in a catalog or a hardware font specification of the form `HWxxxxnnn`. For example, `STYLE='CENTB'` specifies a software font that is stored in the catalog `SASHELP.FONTS`. The maximum length for *font* is 8 characters.

hardware-font-name

specifies the name of a hardware font as shown in the Chartype window of the device entry. The maximum length for *hardware-font-name* is 256 characters.

Hardware-font-name must be enclosed in both double and single quotation marks, for example, `STYLE="Palatino-Italic"`.

NONE

specifies the default hardware font.

See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for more information about specifying fonts.

If the value of the `STYLE` variable is missing, SAS/GRAPH software searches for a font specification in this order:

- 1 the font specified by the `FTEXT=` graphics option
- 2 the hardware font, if the device supports one
- 3 the `SIMULATE` font.

Details

When the `STYLE` variable is used with the `SYMBOL` function, it behaves the same as the `FONT=` option in the `SYMBOL` statement. By default, no font is specified and the

symbol that is specified by the TEXT variable is taken from the special symbol table. If you use STYLE to specify a symbol font, such as Marker, the string that is assigned by the TEXT variable is the character code for a symbol. If you use STYLE to specify a text font, such as Swiss, the string assigned by the TEXT variable is displayed as text. See the FONT= option of the SYMBOL statement for details.

Note: Java does not support the STYLE variable. However, you can use special symbols from the MARKER font by using the SYMBOL function. △

STYLE Variable (Images)

Determines the appearance of images specified with the IMGPATH variable and the IMAGE function.

Type: character

Default: 'TILE'

Syntax

STYLE='TILE' | 'FIT';

'TILE'

Uses copies of the image to fill the image area.

'FIT'

Stretches one instance of the image to fill the image area.

Details

This version of the STYLE variable can be used only with the “IMAGE Function” on page 625.

STYLE Variable (Patterns)

Specifies a pattern for bars, pies, frames, and rectangles

Type: character

Length: 8

Default: EMPTY | PEMPTY | MEMPTY

Not supported by: Java (partial), ActiveX (partial)

Syntax

STYLE='fill-pattern';

fill-pattern

specifies a pattern to use with the graphics element. The value for *fill-pattern* is function-dependent:

*Function**Valid Fill Pattern Values****BAR,FRAME***

SOLID S	Fill with a solid color.
EMPTY E	No fill.
<i>style</i> < <i>density</i> >	<i>style</i> R for right-slanted fill lines, L for left-slanted fill lines, or X for crossing fill lines
	<i>density</i> Whole numbers 1 through 5 specify increasing thickness for the fill lines.

Note: Client-side rendering using Java or ActiveX supports only SOLID and EMPTY and defaults to EMPTY if any other value is used. △

An illustration of these pattern styles is provided in the definition of the VALUE= option of the PATTERN statement.

PIE

PSOLID PS	Solid fill.
PEMPTY PE	No fill, the default.
<i>Pdensity</i> < <i>style</i> < <i>angle</i> >>	Whole numbers 1 through 5 specify increasing thickness for the fill lines.
	<i>style</i> N, the default, optionally specifies parallel fill lines; X optionally specifies crossed fill lines.
	<i>angle</i> Optionally specifies the angle of the fill lines. Values range from 0 to 360. The angle is measured counterclockwise from the horizontal. The default is 0°, which draws horizontal lines.

Note: Client-side rendering using Java or ActiveX supports only PSOLID and PEMPTY and defaults to PEMPTY if any other value is used. △

An illustration of these pattern styles is provided in the definition of the VALUE= option of the PATTERN statement.

POLY

MSOLID MS	Fill with a solid color.
MEMPTY ME	No fill, the default.
<i>Mdensity</i> < <i>style</i> < <i>angle</i> >>	Whole numbers 1 through 5 specify increasing thickness for the fill lines.
	<i>style</i> N, the default, optionally specifies parallel fill lines; X optionally specifies crossed fill lines.
	<i>angle</i> Optionally specifies the angle of the fill lines. Values range from 0 to 360. The angle is measured counterclockwise from

the vertical. The default is 0° , which draws vertical lines.

Note: Client-side rendering using Java or ActiveX supports only MSOLID and MEMPTY and defaults to MEMPTY if any other value is used. \triangle

An illustration of these pattern styles is provided in the definition of the VALUE= option of the PATTERN statement.

SUBGROUP Variable

Positions graphics elements within subgrouped bars of a vertical or horizontal bar chart produced by the GCHART procedure.

Type: Numeric or character; must match the type of the SUBGROUP variable used in the GCHART procedure.

Length: Should match the length of the SUBGROUP= variable in the GCHART procedure.

Default: none

Restriction: The bar charts must have been produced using the SUBGROUP= option.

Syntax

SUBGROUP=*subgroup-value*;

subgroup-value

references value(s) of the SUBGROUP= variable in the GCHART procedure either as a variable name or as an explicit data value. *Subgroup-value* can have one of the following forms:

subgroup-variable the name of a subgroup variable.

subgroup-data-value a specific numeric data value.

subgroup-data-value a specific character data value.

Generally, specify a variable name if you want to annotate all of the bars in a horizontal or vertical bar chart. To annotate a bar chart for a specific value of the SUBGROUP variable, specify a specific value.

Functions

You can use the SUBGROUP variable only with the data coordinate system 1, 2, 7, or 8, and with these functions:

BAR

DRAW

LABEL

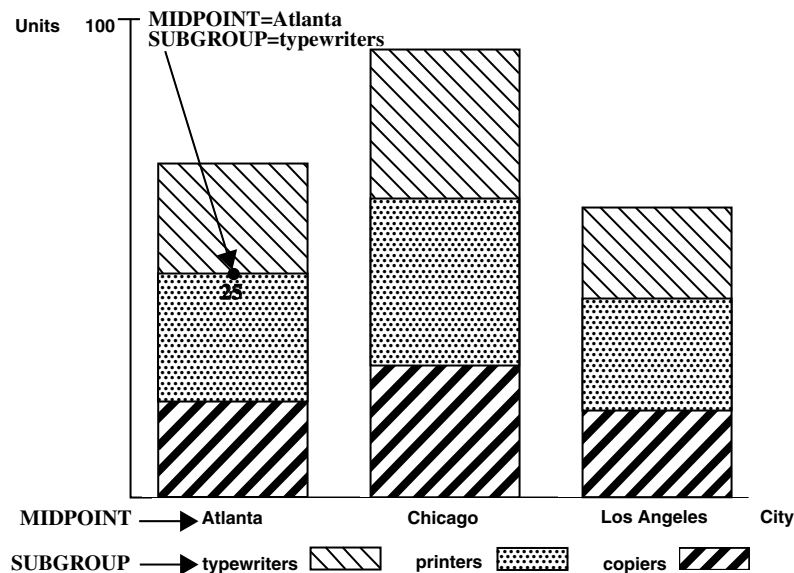
MOVE
 PIE
 PIECNTR
 POINT
 POLY
 POLYCONT
 SYMBOL

Details

Using the SUBGROUP variable is similar to using the X and Y variables with data system coordinates to position the graphics elements in subgroup segments in vertical and horizontal bar charts. For example, in a vertical bar chart that produces a bar for each city in a data set, you can easily label the subgroups in each bar by setting *subgroup-variable* to the GCHART variable by which the bar is being subgrouped. This variable provides the *y* coordinate of the label (so don't specify a competing value for *y*, but instead specify *y=.* or *y=y*).

The MIDPOINT variable works well with the SUBGROUP variable to provide the *x* coordinate. In this example, if you set the MIDPOINT variable to the GCHART variable that contains the names of the cities, the MIDPOINT variable provides your *x* coordinate. Rather than providing the X and Y variables, you would use the SUBGROUP and MIDPOINT variables. Figure 25.22 on page 665 shows how the SUBGROUP variable works with the MIDPOINT variable to label the bars of a vertical bar chart.

Figure 25.22 Using the SUBGROUP Variable to Position a Label in a Bar Chart



The label showing the number of printers sold in Atlanta is positioned by the values that are assigned to these Annotate variables:

- MIDPOINT=CITY (where CITY is a variable in the GCHART data set)

- SUBGROUP=ITEM (where ITEM is a variable in the GCHART data set).

TEXT Variable

Specifies the text or symbol to be placed on the graphics output.

Type: character

Length: up to 200

Default: blank string

Syntax

TEXT='text-string' | 'special-symbol';

text-string

specifies the text that is used as a label (LABEL or COMMENT function) or symbol (SYMBOL function). The maximum length for *text-string* is 200 characters.

special-symbol

specifies the name of a symbol from the special symbol table that is illustrated in Figure 7.21 on page 202. The maximum length for *special-symbol* is eight characters.

Functions

You can use the TEXT variable with these functions:

COMMENT

LABEL

SYMBOL

Details

Define the TEXT variable with sufficient length to contain all of the characters in your text string. If you need longer strings, use separate observations and POSITION='0' to continue the text.

Use a LENGTH statement to set the length of the TEXT variable if the length of a text string is longer than one character.

WHEN Variable

Specifies when the function is performed in relation to generating other graphics output for the procedure or in relation to generating other Annotate graphics.

Type: character

Length: 1

Default: B

Syntax

WHEN='B' | 'A' ;

B | A

specifies whether to draw the annotation before (B) or after (A) the graph. These values are not case sensitive. A missing value is equivalent to specifying B.

Note: Some annotations coded with WHEN='B' that work on the server may not be visible with client-side rendering using Java or ActiveX because the annotations are drawn behind the backplane. The only solution is to code WHEN='A'. △

Functions

You can use the WHEN variable with these functions:

BAR

DRAW

DRAW2TXT

FRAME

LABEL

MOVE

PIE

PIECNTR

PIEXY

POINT

POLY

POLYCONT

SYMBOL

Details

Normally, observations in an Annotate data set are processed sequentially. If you use the WHEN variable, all those observations with a WHEN value of B are processed first, the procedure output is then processed (if one is to be produced), and finally the observations with a WHEN value of A are processed.

X Variable

Identifies the x coordinate of where a graphics element is to be drawn.

Type: numeric

Default: value of XLAST or XLSTT

Syntax

X =horizontal-coordinate;

Functions

You can use the X variable with these functions:

BAR
DRAW
IMAGE
LABEL
MOVE
PIE
PIECNTR
POINT
POLY
POLYCONT
SYMBOL

Note: The X or XC variable is required unless either the MIDPOINT, GROUP, or SUBGROUP variable provides the horizontal coordinate. \triangle

Details

Specify a corresponding vertical coordinate when using the X variable. This vertical coordinate can be specified with the Y , YC , MIDPOINT, or SUBGROUP variables, depending on the type of graph that you are annotating.

The X variable uses the units that are specified in the $XSYS$ variable. If you use $XSYS='2'$ and the data axis is typed as character, use the XC variable instead of the X variable.

If the value of the X variable is missing for a function that requires it, the value of the $XLAST$ variable is used with nontext functions and the value of the $XLSTT$ variable is used with text functions.

XC Variable

Identifies the x coordinate of a graphics element when the coordinate value is character.

Type: character

Length: Should match that of the plot variable in the procedure.

Default: the value of $XLAST$ or $XLSTT$

Restrictions: Used only with output from the GCHART and GPLOT procedures. Ignored if the axes are numeric.

Syntax

$XC='character-type-horizontal-coordinate';$

Functions

You can use the XC variable with these functions:

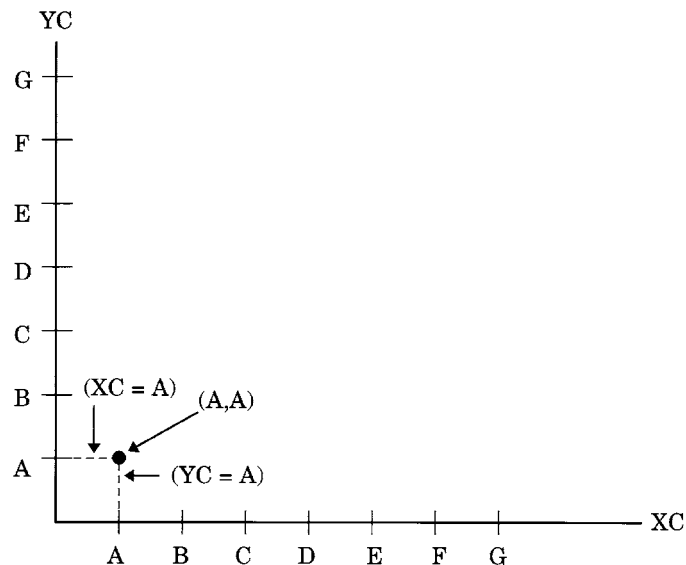
BAR
DRAW
LABEL
MOVE
PIE
PIECNTR
POINT
POLY
POLYCONT
SYMBOL

Details

The XC variable is the character equivalent of the X variable. Use XC when the axis values are character. You must also specify a value of 2 (absolute data values) for the XSYS variable. (See also “XSYS Variable” on page 670.) If you use a value other than 2 for the XSYS variable, the graphics output is not displayed properly.

Figure 25.23 on page 669 illustrates the XC variable.

Figure 25.23 Using the XC and YC Variables with Character Data



Note: The X or XC variable is required unless either the MIDPOINT, GROUP, or SUBGROUP variable provides the horizontal coordinate. Δ

CAUTION:

Do not use the X and XC variables in the same data set. Using both X and XC variables in the same data set can cause unpredictable results. Δ

XSYS Variable

Defines the coordinate system and area of the output used by the X and XC variables to display the Annotate graphics.

Type: character

Length: 1

Default: 4

Syntax

XSYS='coordinate-system';

coordinate-system

specifies a value that represents a coordinate system. Values can be 1 through 9 and A through C as shown in the following table:

Absolute Systems	Relative Systems	Coordinate System Units
1	7	percentage of data area
2	8	data values
3	9	percentage of graphics output area
4	A	cell in graphics output area
5	B	percentage of procedure output area
6	C	cell in procedure output area

These values are also used by the HSYS and YSYS variables. See “Coordinate Systems” on page 596 for a description of the areas and coordinate systems.

Functions

You can use the XSYS variable with these functions:

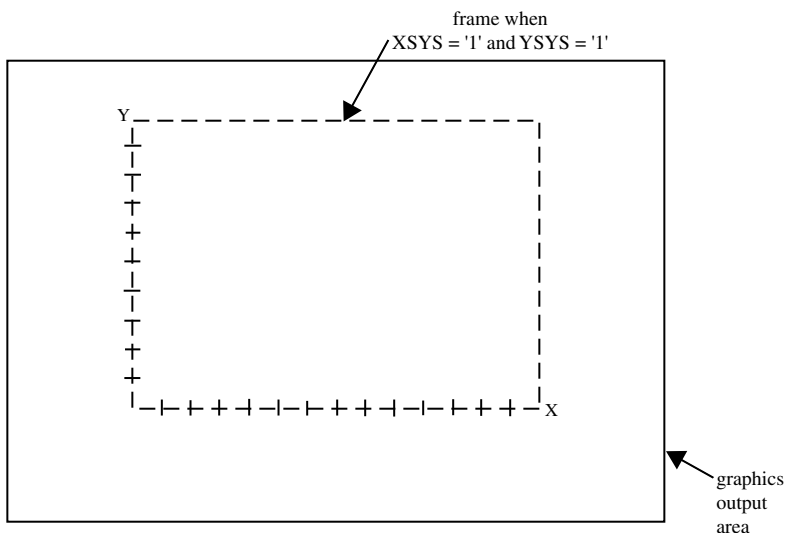
BAR

DRAW
 FRAME
 LABEL
 MOVE
 PIE
 PIECNTR
 POINT
 POLY
 POLYCONT
 SYMBOL

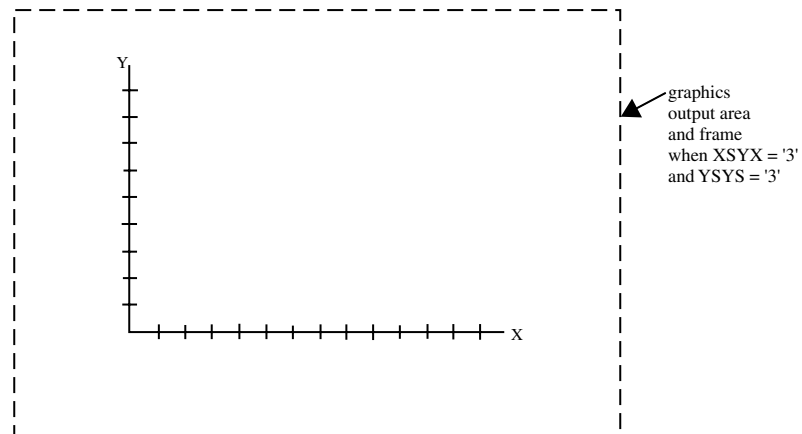
The behavior of the XSYS variable is function-dependent for the following functions:

- BAR, DRAW* The coordinate system that you specify with the XSYS variable affects how the function interprets the value of the X or XC variable. If XC is used, XSYS='2' must also be used.
- FRAME* The XSYS and YSYS variables define the area enclosed by the frame. To draw a frame that encloses the axis area, use XSYS='1' and YSYS='1', as shown in the following figure.

Figure 25.24 Frame Created When XSYS='1' and YSYS='1'

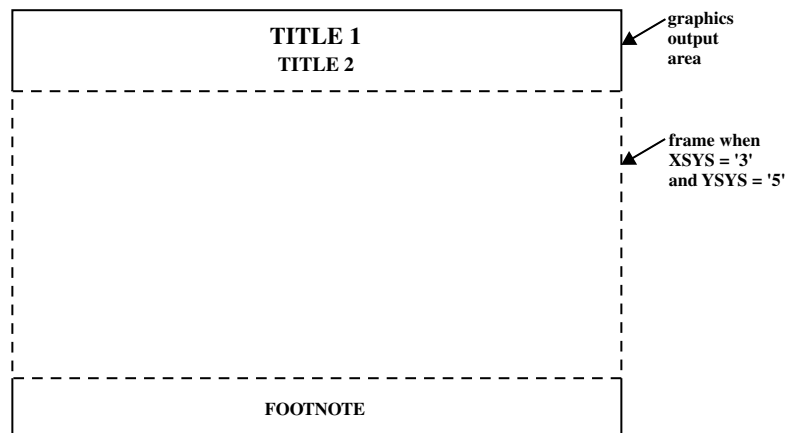


To draw a frame that encloses the entire graphics output area, specify XSYS='3' and YSYS='3', as shown in the following figure.

Figure 25.25 Frame Created When XSYS='3' and YSYS='3'

To limit the size of the frame to the size of the procedure output area, specify a value of 5 for XSYS and YSYS.

Note that the values of XSYS and YSYS can differ. You can specify a frame that occupies the entire width of the graphics output area and only the vertical width of the procedure output area by specifying XSYS='3' and YSYS='5', as shown in the following figure.

Figure 25.26 Frame Created When XSYS='3' and YSYS='5'

Details

The coordinate system that you specify with the XSYS variable affects how the function interprets the value of the X or XC variable.

Note: Not all coordinate systems can be used with all Annotate variables. For any restrictions, see the individual variables in this section. Δ

Y Variable

Identifies the *y* coordinate of where a graphics element is to be drawn.

Type: numeric

Default: value of YLAST or YLSTT

Syntax

Y=vertical-coordinate;

Functions

You can use the Y variable with these functions:

BAR

DRAW

IMAGE

LABEL

MOVE

PIE

PIECNTR

POINT

POLY

POLYCONT

SYMBOL

Note: The Y or YC variable is required unless either the MIDPOINT, GROUP, or SUBGROUP variable provides the vertical coordinate. △

Details

Specify a corresponding horizontal coordinate when using the Y variable. You can specify the horizontal coordinate with the X, XC, MIDPOINT, or SUBGROUP variable, depending on the type of graph you are annotating.

The Y variable uses the units specified in the YSYS variable. If you use YSYS='2' and the axis data is type character, use the YC variable instead of the Y variable.

If the value of the Y variable is missing for a function that requires it, the value YLAST is used for nontext functions and the value of YLSTT is used for text functions.

YC Variable

Identifies the *y* coordinate of a graphics element when the coordinate value is character.

Type: character

Length: Should match that of the plot variable in the procedure.

Default: YLAST | YLSTT

Restrictions: Used only with output from the GCHART and GPLOT procedures. Ignored if the axes are numeric.

Syntax

YC='character-type-vertical-coordinate';

Functions

You can use the YC variable with these functions:

BAR

DRAW

LABEL

MOVE

PIE

PIECNTR

POINT

POLY

POLYCONT

SYMBOL

Details

The YC variable is the character equivalent of the Y variable. Use YC when the axis values are character. You must also specify a value of 2 (absolute data values) for the YSYS variable. (See “YSYS Variable” on page 674.) If you use a value other than 2 for the YSYS variable, the graphics output is not displayed properly.

See Figure 25.23 on page 669 for an illustration of the YC variable.

Note: The X or XC variable is required unless either the MIDPOINT, GROUP, or SUBGROUP variable provides the horizontal coordinate. \triangle

CAUTION:

Do not use Y and YC variables in the same data set. Using both Y and YC variables in the same data set can cause unpredictable results. \triangle

YSYS Variable

Defines the coordinate system and area of the output used by Y and YC to display the Annotate graphics.

Type: character

Length: 1

Default: 4

Syntax

YSYS='coordinate-system';

coordinate-system

specifies a value that represents a coordinate system. Values can be 1 through 9 and A through C, as shown in the following table:

Absolute Systems	Relative Systems	Coordinate System Units
1	7	percentage of data area
2	8	data values
3	9	percentage of graphics output area
4	A	cell in graphics output area
5	B	percentage of procedure output area
6	C	cell in procedure output area

These values are also used by the HSYS and XSYS variables. See “Coordinate Systems” on page 596 for a description of the areas and coordinate systems.

Functions

The YSYS variable is function-dependent, as defined in the “XSYS Variable” on page 670

You can use the YSYS variable with these functions:

BAR

DRAW

FRAME

LABEL

MOVE

PIE

PIECNTR

POINT

POLY

POLYCONT

SYMBOL

Details

The coordinate system that you specify with the YSYS variable affects how the function interprets the value of the Y or YC variable.

Note: Not all coordinate systems can be used with all Annotate variables. For any restrictions, see the individual variables in this section. △

Z Variable

Identifies the z coordinate of where a graphics element is to be drawn.

Type: numeric

Length: 8

Default: none

Restrictions: On the server, is used only with output from the G3D procedure. For client-side annotation with Java or ActiveX, you can use the Z variable with GMAP, GCHART, GCONTOUR, GPLOT, and G3D, for example to add annotations above the plane of the map.

Syntax

Z=depth-coordinate;

Functions

You can use the Z variable with these functions:

BAR

DRAW

IMAGE

LABEL

MOVE

PIE

PIECNTR

POINT

POLY

POLYCONT

SYMBOL

Details

The Z variable uses the units that are specified in the ZSYS variable.

ZSYS Variable

Defines the coordinate system and area of the output used by Z variable to display the Annotate graphics.

Type: character

Length: 1

Default: 2

Syntax

ZSYS='coordinate-system';

coordinate-system

specifies a value that represents a coordinate system. Values can be 1, 2, 7, or 8 as shown in the following table:

Absolute Systems	Relative Systems	Coordinate System Units
1	7	percentage of data area
2	8	data values

See “Coordinate Systems” on page 596 for a description of the areas and coordinate systems.

Functions

You can use the ZSYS variable with these functions:

BAR

DRAW

IMAGE

LABEL

MOVE

PIE

PIECNTR

POINT

POLY

POLYCONT

SYMBOL

Details

The coordinate system that you specify with the ZSYS variable affects how the function interprets the value of the Z variable.

Note: Not all coordinate systems can be used with all Annotate variables. For any restrictions, see the individual variables in this section. △

Annotate Internal Coordinates

The Annotate facility maintains two sets of internal coordinates that are stored in the variable pairs (XLAST, YLAST) and (XLSTT, YLSTT). One set of variables (XLAST, YLAST) stores coordinate values that are generated by nontext functions and the other set (XLSTT, YLSTT) stores coordinates generated by text functions. These two variable pairs supply default values when the X or Y variable contains a missing value.

Both pairs are initially set to 0 and remain 0 until a function updates the values. You cannot assign explicit values to these variables, but you can manipulate their values with some of the Annotate functions.

XLAST, YLAST Variables

Track the last values specified for the X and Y variables when X and Y are used with nontext functions.

Details

The coordinate values that are stored in the (XLAST, YLAST) variables are automatically updated by these nontext functions: BAR, DRAW, MOVE, PIE, and POINT. These values are then available for use by other nontext functions that follow in the DATA step. (The DRAW2TXT graphics function uses XLAST and YLAST but does not update them.)

Because (XLAST, YLAST) are updated internally, you cannot specify values for them. However, their values can be manipulated by these programming functions:

CNTL2TXT
PIECNTR
PIEXY
POP
PUSH
SWAP
TXT2CNTL

XLSTT, YLSTT Variables

Track the last position for the X and Y variables when X and Y are used with text-handling functions.

Details

The coordinate values stored in the (XLSTT, YLSTT) variables are automatically updated by the LABEL and SYMBOL text functions. These values are then available for use by other text functions that follow in the DATA step.

Because (XLSTT, YLSTT) are updated internally, you cannot specify values for them. However, their values can be manipulated by these programming functions:

CNTL2TXT

DRAW2TXT

POP

PUSH

SWAP

TXT2CNTL

Annotate Macros

You can use Annotate macros within a SAS DATA step to simplify the process of creating Annotate observations. With a macro, you specify a function and assign variable values in one step without having to write explicit variable assignment statements. You can mix assignment statements and macro calls in the same DATA step.

This section describes all of the Annotate macros including the complete syntax and a description of the parameters. For more information on accessing and using macros, and for a summary of operations performed by the Annotate macros, see “Using Annotate Macros” on page 697.

%ANNOMAC Macro

Compiles Annotate macros and makes them available for use.

Variables written out: none directly

Syntax

```
%ANNOMAC;
```

Details

In a SAS session, you must submit the ANNOMAC macro before you can use the Annotate macros.

%BAR, %BAR2 Macros

Draws a rectangle using two sets of x/y coordinates, which specify diagonal corners. You can specify the rectangle's line type, line color, fill type, and fill color.

Variables written out: COLOR, FUNCTION, LINE, STYLE, X, Y

Internal variables updated: XLAST, YLAST

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%BAR (*x1, y1, x2, y2, color, line, style*);

%BAR2(*x1, y1, x2, y2, color, line, style, width*);

x1, y1

specify the location of the first corner of the bar. Values can be numeric coordinates, numeric constants, or numeric variables. For details, see the Annotate “X Variable” on page 667.

x2, y2

specify the location of second corner of the bar, which is drawn diagonal to the first corner. Values can be numeric coordinates, numeric constants, or numeric variables.

color

specifies the outline color and optional fill color using a character string without quotation marks. For details, see the Annotate “COLOR Variable” on page 645.

line

specifies which of the outlines of the bar are to be drawn. The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “LINE Variable” on page 652 for the BAR function.

style

specifies the fill pattern for the bar using a character string without quotation marks. For valid values, see the Annotate “STYLE Variable (Patterns)” on page 662 for the BAR function.

width

specifies the width of the outline and optional fill lines. The value can a number, a numeric constant, or a numeric variable. For details and valid values, see the Annotate “SIZE Variable” on page 660 for the DRAW function.

%CENTROID Macro

Retrieves the centroids of polygons

Variables written out: X, Y, id variables

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%CENTROID (*input-data-set, output-data-set, list-of-id-variables*);

input-data-set

specifies a map data set.

output-data-set

contains the id variables and the X and Y variables.

list-of-id-variables

specifies the variables each of which is to be assigned the centroid coordinates of each observation in the input-data-set. There will be one observation for each unique set of ID values

%CIRCLE Macro

Draws an empty circle with the center at (x , y).

Variables written out: ANGLE, FUNCTION, ROTATE, SIZE, STYLE, X, Y

Internal variables updated: XLAST, YLAST

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%CIRCLE (x , y , *size*, *color*);

x*, *y

specify coordinates for the center of the circle. Values can be coordinate numbers, numeric constants, or numeric variables. For details, see the Annotate “X Variable” on page 667.

size

specifies the radius of the circle. The value can be a number, a numeric constant, or a numeric variable. For details and valid values, see the Annotate “SIZE Variable” on page 660.

color

specifies the color of the circle using a character string without quotation marks. For details, see the Annotate “COLOR Variable” on page 645. Use an asterisk (*) to specify the previous value of the *color* parameter.

See Also

“%SLICE Macro” on page 695 to draw a filled circle.

%CNTL2TXT Macro

Copies the values of the internal coordinates (XLAST, YLAST) to the text coordinate (XLSTT, YLSTT).

Variables written out: FUNCTION

Internal variables updated: XLSTT, YLSTT

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

`%CNTL2TXT;`

Details

The %CNTL2TXT macro is useful when you are calculating the position of labels on a graph. For an example, see “CNTL2TXT Function” on page 617.

%COMMENT Macro

Inserts a comment into an Annotate data set.

Variables written out: FUNCTION, TEXT

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

`%COMMENT (text-string);`

text-string

specifies the text to insert in the Annotate data set. The value can be a character string enclosed in quotation marks or the name of a character variable. For details, see the Annotate “TEXT Variable” on page 666.

%DCLANNO Macro

Automatically sets the correct length and data type for all Annotate variables except the TEXT variable.

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

`%DCLANNO;`

%DRAW Macro

Draws a line from (XLAST, YLAST) to the specified coordinate.

Variables written out: COLOR, FUNCTION, LINE, SIZE, X, Y

Internal variables updated: XLAST, YLAST

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%DRAW (*x, y, color, line, size*);

x, y

specify coordinates for the end point of the line. Values can be coordinate numbers, numeric constants, or numeric variables. For details, see the Annotate “X Variable” on page 667.

color

specifies the color of the line using a character string without quotation marks. For details, see the Annotate “COLOR Variable” on page 645. Use an asterisk (*) to specify the previous value of the *color* parameter.

line

specifies the line type (continuous or segmented). The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “LINE Variable” on page 652 for the DRAW function.

size

specifies the width of the line. The value can be a number, a numeric constant, or a numeric variable. For valid numeric values, see the Annotate “SIZE Variable” on page 660 for the DRAW function.

Details

The point from which the line is drawn is usually set with the MOVE macro.

%DRAW2TXT Macro

Draws a line from the coordinate (XLAST, YLAST) to the text coordinate (XLSTT, YLSTT).

Variables written out: COLOR, FUNCTION, LINE, SIZE

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%DRAW2TXT (*color, line, size*);

color

specifies the color of the line using a character string without quotation marks. For details, see the Annotate “COLOR Variable” on page 645. Use an asterisk (*) to specify the previous value of the *color* parameter.

line

specifies the line type (continuous or segmented). The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “LINE Variable” on page 652 for the DRAW function.

size

specifies the width of the line. The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “SIZE Variable” on page 660 for the DRAW function.

%FRAME Macro

Draws a border around the portion of the display area defined by the reference system and optionally fills the area.

Variables written out: COLOR, FUNCTION, LINE, SIZE, STYLE

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%FRAME (*color, line, size, style*);

color

specifies the outline color and the optional fill color using a character string without quotation marks. For details, see the Annotate “COLOR Variable” on page 645. Use an asterisk (*) to specify the previous value of the *color* parameter.

line

specifies a line type (continuous or segmented) for the frame outline and fill lines. The value can be a number, a numeric constant, or a numeric variable. For valid numeric values, see the Annotate “LINE Variable” on page 652 for the DRAW function.

size

specifies the width of the frame outline and fill lines. The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “SIZE Variable” on page 660 for the DRAW function.

style

specifies the fill pattern for the frame using a character string without quotation marks. For valid values, see the Annotate “STYLE Variable (Patterns)” on page 662 for the FRAME function.

Details

See “%SYSTEM Macro” on page 696 for information on setting the reference system.

%LABEL Macro

Places a text label at the specified coordinates.

Variables written out: ANGLE, COLOR, FUNCTION, POSITION, ROTATE, SIZE, STYLE, TEXT, X, Y

Internal variables updated: XLSTT, YLSTT

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%LABEL (*x, y, text-string, color, angle, rotate, size, style, position*);

x, y

specifies the location of the text string. Values can be coordinate numbers, numeric constants, or numeric variables. The position of the text string relative to *x, y* is determined by the *position* parameter. For details, see the Annotate “X Variable” on page 667.

text-string

specifies the text of the label. The value can be a character variable name or a character string enclosed in quotation marks. For details, see the Annotate “TEXT Variable” on page 666.

color

specifies the color of the text string using a character string without quotation marks. For details, see the Annotate “COLOR Variable” on page 645. Use an asterisk (*) to specify the previous value of the *color* parameter.

angle

specifies the angle of the text string with respect to the horizontal. The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “ANGLE Variable” on page 642 for the LABEL function. The *x, y* coordinates specify the pivot point, and the *position* parameter positions the text relative to *x, y*.

rotate

specifies the rotation angle of each character in the text string. The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “ROTATE Variable” on page 659.

size

specifies the size of the text string. The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “SIZE Variable” on page 660 for the LABEL function.

style

specifies the text font, using a character string without quotation marks. For valid values, see the Annotate “STYLE Variable (Fonts)” on page 661.

position

specifies the placement and alignment of the text string relative to the *x, y* coordinates, using a text string without quotation marks. For valid values, see the Annotate “POSITION Variable” on page 656.

%LINE Macro

Draws a line between two sets of coordinates.

Variables written out: COLOR, FUNCTION, LINE, SIZE, X, Y

Internal variables updated: XLAST, YLAST

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%LINE (*x1, y1, x2, y2, color, line, size*);

x1, y1

specify the coordinates of the start of the line. Values can be numbers, numeric constants, or numeric variables. For details, see the Annotate “X Variable” on page 667 variable.

x1, y2

specify the coordinates of the end of the line. Values can be numbers, numeric constants, or numeric variables.

color

specifies the color of the line using a character string without quotation marks. For valid values, see the Annotate “COLOR Variable” on page 645. Use an asterisk (*) to specify the previous value of the *color* parameter.

line

specifies the line type, which can be continuous or segmented. The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “LINE Variable” on page 652 for the DRAW function.

size

specifies the width of the line. The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “SIZE Variable” on page 660 for the DRAW function.

%MAPLABEL Macro

Creates an output data set that can be used with the ANNO= option for PROC GMAP.

Variables written out: FUNCTION, STYLE, COLOR, SIZE, HSYS

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%MAPLABEL (*map-dataset, attr-dataset, output-dataset, label-var, id-list, font=font_name, color=n, size=n, hsys=n*);

map-dataset

the name of the map to be annotated.

attr-dataset

the name of the dataset containing the text to be shown on each ID value.

output-dataset

the name of the annotate data set created by the macro.

label-var

the name of the label variable to place on the map (the text for annotate).

id-list

the list of ID vars that you would issue in PROC GMAP to create the map. These values need to be on both the map and the attribute data sets. If you also supply the SEGMENT variable, then every polygon will get a value. Without the SEGMENT variable, only one label per ID set will be shown over the collection of polygons. For instance, Hawaii with SEGMENT gets a label on each island, whereas without SEGMENT, there is only one label centered on the entire set of islands.

font

specifies a font name for the “STYLE Variable (Fonts)” on page 661 variable.

color

specifies a value for the “COLOR Variable” on page 645 variable.

size

specifies a value for the “SIZE Variable” on page 660 variable. Defaults to 2.

hsys

specifies a value for the “HSYS Variable” on page 649 variable. Defaults to 3.

%MOVE Macro

Moves to the (x, y) coordinate.

Variables written out: FUNCTION, X, Y

Internal variables updated: XLAST, YLAST

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%MOVE (x, y) ;

 x, y

specify new coordinates for the next annotation. Values can be numeric coordinates, numeric constants, or numeric variables. For details, see the Annotate “X Variable” on page 667.

%PIEXY Macro

Calculates a point in relation to the latest pie slice.

Variables written out: ANGLE, FUNCTION, SIZE, X, Y

Internal variables updated: XLAST, YLAST

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%PIEXY (*angle*, *size*);

angle

specifies the angle used to calculate the point, relative to the center of the latest pie slice. The value can be a number, a numeric constant, or a numeric variable. For details, see the Annotate “ANGLE Variable” on page 642 for the PIEXY function.

size

specifies the radius multiplier that works with the *angle* parameter to determine the location of the point. The value can be a number, a numeric constant, or a numeric variable. For details and valid values, see the Annotate “SIZE Variable” on page 660 for the PIEXY function.

Details

This macro is useful when you want to label a pie chart or a circle.

When you use this macro, the Annotate facility expects a slice to have been previously drawn. If a slice has not been drawn or if the “PIECNTR Function” on page 630 has not been processed, you can get erroneous results.

%POLY, %POLY2 Macro

Begins drawing a polygon at the specified coordinates and determines the color, fill pattern, and line type of the polygon.

Variables written out: FUNCTION, COLOR, LINE, STYLE, X, Y,

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%POLY (*x*, *y*, *color*, *style*, *line*);

%POLY2(*x*, *y*, *color*, *style*, *line*, *width*);

x, y

specify the starting point for a new polygon. Values can be numeric coordinates, numeric constants, or numeric variables. For details, see the Annotate or the names of the Annotate variables “X Variable” on page 667.

color

specifies the optional polygon fill color using a character string without quotation marks. For valid values, see the Annotate “COLOR Variable” on page 645. Use an asterisk (*) to specify the previous value of the *color* parameter. To specify the color of the polygon outline, see the “%POLYCONT Macro” on page 689.

style

specifies the fill pattern for the polygon, using a character string without quotation marks. For valid values, see the Annotate “STYLE Variable (Patterns)” on page 662 for the POLY function.

line

specifies the polygon’s line type, which can be continuous or segmented. The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “LINE Variable” on page 652 for the POLY function.

width

specifies the width of the polygon’s outline and optional fill lines. The value can be a number, a numeric constant, or a numeric variable. For details and valid values, see the Annotate “SIZE Variable” on page 660 for the POLY function.

See Also

“POLY Function” on page 634

%POLYCONT Macro

Continues drawing the polygon to the next specified coordinates.

Variables written out: COLOR, FUNCTION, X, Y

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%POLYCONT (*x, y, color*);

x, y

specify the end point of the next line in the polygon. Values can be numeric coordinates, numeric constants, or numeric variables. For details, see the Annotate “X Variable” on page 667.

color

specifies the color of the polygon outline using a character string without quotation marks. For valid values, see the Annotate “COLOR Variable” on page 645. Use an asterisk (*) to specify the previous value of the *color* parameter.

Details

The first invocation of the %POLYCONT macro in the polygon-drawing sequence determines the outline color of that polygon. Subsequent color specifications for that polygon in later invocations of the %POLYCONT macro are ignored.

The polygon fill color and line type are specified in the initial “%POLY, %POLY2 Macro” on page 688 or %POLY2 macro.

%POP Macro

Removes the coordinates (XLAST, YLAST) and (XLSTT, YLSTT) from the LIFO system stack and updates the internal coordinate pairs with these retrieved values.

Variables written out: FUNCTION

Internal variables updated: XLAST, YLAST, XLSTT, YLSTT

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%POP;

Details

Use the %POP macro when you want to access the values of the XLAST, YLAST, XLSTT, and YLSTT variables that you previously stored with the %PUSH macro. For more information, see “XLAST, YLAST Variables” on page 678, “XLSTT, YLSTT Variables” on page 678, and “%PUSH Macro” on page 690.

%PUSH Macro

Enters the coordinates (XLAST, YLAST) and (XLSTT, YLSTT) in a LIFO system stack.

Variables written out: FUNCTION, internal coordinates

Internal variables updated: XLAST, YLAST, XLSTT, YLSTT

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%PUSH;

Details

The last-in, first-out (LIFO) stack provides a way to save previously calculated coordinates. It enables you to retain coordinate values for later use by utility functions without recalculating those values. In order to save coordinate values in the stack, you

must explicitly push them onto the stack. See “Using the LIFO Stack” on page 603 for a description of the LIFO stack.

%RECT Macro

Draws a rectangle with diagonal corners at two specified points.

Variables written out: COLOR, FUNCTION, LINE, SIZE, X, Y

Internal variables updated: XLAST, YLAST

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%RECT (*x1, y1, x2, y2, color, line, size*) ;

x1, y1

specify the coordinates of the first corner of the rectangle. Values can be numeric coordinates, numeric constants, or numeric variables. For details, see the Annotate “X Variable” on page 667.

x2, y2

specify the coordinates of the second corner of the rectangle, which is drawn diagonal to the first corner. Values can be numeric coordinates, numeric constants, or numeric variables.

color

specifies the color of the rectangular line using a character string without quotation marks. For valid values, see the Annotate “COLOR Variable” on page 645. Use an asterisk (*) to specify the previous value of the *color* parameter.

line

specifies the rectangle’s line type, which can be continuous or segmented. The value can be a number, a numeric constant, or a numeric variable. For details, see the Annotate “LINE Variable” on page 652 for the DRAW function.

size

specifies the width of the line. The value can be a number, a numeric constant, or a numeric variable. For valid values, see the “SIZE Variable” on page 660 for the DRAW function.

Details

The rectangle is drawn such that the first corner is diagonal to the second corner.

The %RECT macro produces rectangles that do not have fill patterns. Use the %BAR macro to generate filled rectangles. For more information, see “%BAR, %BAR2 Macros” on page 679.

%SCALE Macro

Scales input coordinates relative to the origin (0, 0) based on the relationship between two ranges of minima and maxima.

Variables written out: X, Y

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%SCALE (*ptx, pty, x1, y1, x2, y2, vx1, vy1, vx2, vy2*);

ptx, pty

specifies the coordinates to scale. Values can be numbers, numeric constants, or numeric variables. For details, see the Annotate “X Variable” on page 667.

x1, y1

specifies the minima of the first range. Values can be numbers, numeric constants, or numeric variables.

x2, y2

specifies the maxima of the first range. Values can be numbers, numeric constants, or numeric variables.

vx1, vy1

specifies the minima of the second range. Values can be numbers, numeric constants, or numeric variables.

vx2, vy2

specifies the maxima of the second range. Values can be numbers, numeric constants, or numeric variables.

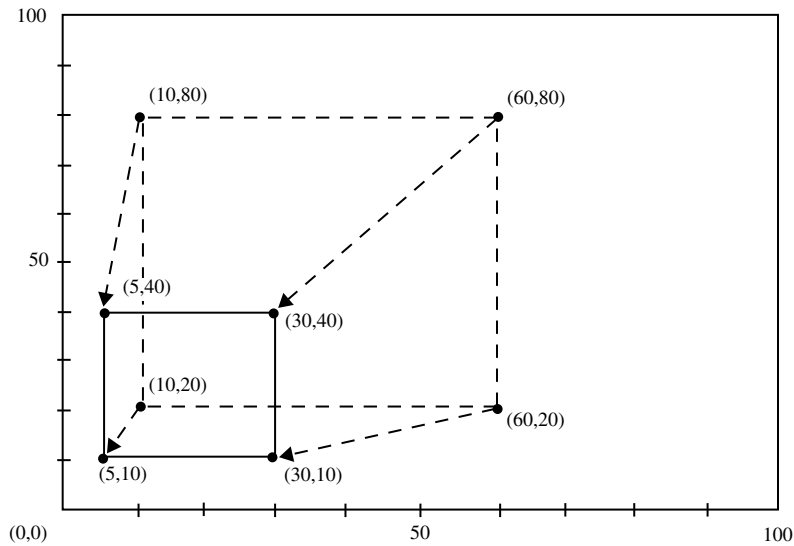
Details

The %SCALE macro reduces or enlarges Annotate graphics elements that use two-dimensional, numeric coordinates. The %SCALE macro does not affect graphics elements that are drawn with text functions.

The difference between the %SCALE and %SCALET macros is that the %SCALE macro always places the origin at (0, 0) and plots the new coordinates with respect to that origin. The %SCALET macro plots the new coordinates with respect to the minima of the second range. For details, see “%SCALET Macro” on page 693.

The following example uses the %SCALE macro to reduce *x* and *y* coordinates by 50 percent, as shown in Figure 25.27 on page 693:

```
%SCALE(x, y, 0, 0, 100, 100, 0, 0, 50, 50);
```

Figure 25.27 Using the %SCALE Macro to Reduce the Size of a Box

%SCALET Macro

Scales input coordinates based on the relationship between two ranges of minima and maxima. The scaled coordinates are plotted relative to the minima of the second range.

Variables written out: X, Y

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%SCALET (*ptx, pty, x1, y1, x2, y2, vx1, vy1, vx2, vy2*);

ptx, pty

specifies the coordinates to scale. Values can be numbers, numeric constants, or numeric variables. For details, see the Annotate “X Variable” on page 667.

x1, y1

specifies the minima of the original range. Values can be numbers, numeric constants, or numeric variables.

x1, y2

specifies the maxima of the original range. Values can be numbers, numeric constants, or numeric variables.

vx1, vy1

specifies the minima of the second range using numeric values. Values can be numbers, numeric constants, or numeric variables. These coordinates are also used as the origin against which the scaled point is plotted.

vx2, vy2

specifies the maxima of the second range. Values can be numbers, numeric constants, or numeric variables.

Details

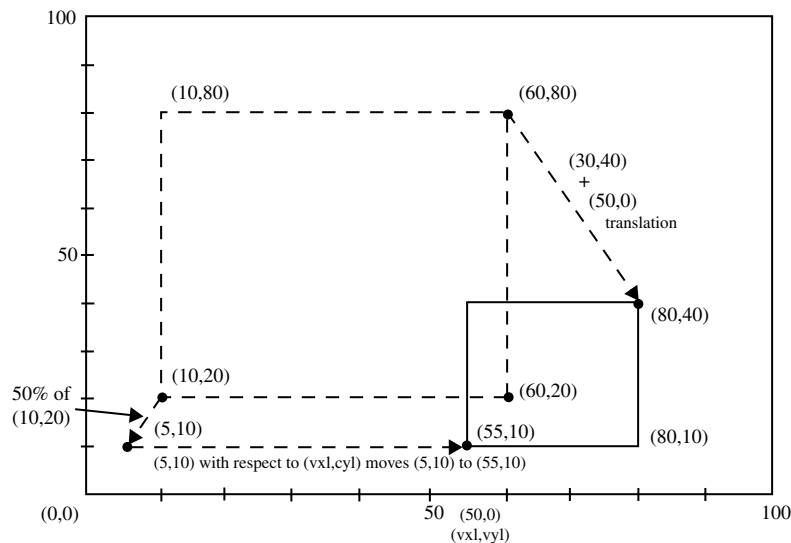
The %SCALET macro reduces or enlarges Annotate graphics elements that use two-dimensional numeric coordinates. The %SCALET macro does not affect graphics elements that are drawn with text functions.

The difference between the %SCALET and %SCALE macros is that the SCALET macro plots the new coordinates with respect to minima of the second range (*vx1, vy1*). The %SCALE macro plots the new coordinates with respect to the origin (0, 0).

The following example uses the %SCALET macro reduces *x* and *y* coordinates by 50 percent and plots the new coordinates with respect to (50, 0), as shown in Figure 25.28 on page 694:

```
%SCALET(x, y, 0, 0, 100, 100, 50, 0, 50, 100);
```

Figure 25.28 Using the %SCALET Macro to Reduce the Size of a Box

**%SEQUENCE Macro**

Specifies when to draw Annotate graphics elements, relative to the procedure's graphics output or relative to the other Annotate graphics drawn.

Variables written out: WHEN

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see "Making the Macros Available" on page 697.

Syntax

%SEQUENCE (*when*);

when

Values can be BEFORE or AFTER, as defined for the Annotate “WHEN Variable” on page 666.

%SLICE Macro

Draws a arc, pie slice, or circle, with available line types, colors, and fill types.

Variables written out: ANGLE, COLOR, FUNCTION, LINE, ROTATE, SIZE, STYLE, X, Y

Internal variables updated: XLAST, YLAST

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%SLICE (*x, y, angle, rotate, size, color, style, line*);

x, y

specify the center point of the arc. Values can be numbers, numeric constants, or numeric variables. For details, see the Annotate “X Variable” on page 667.

angle

specifies the starting point of the arc. The value can be a number, a numeric constant, or a numeric variable. For details and valid values, see the Annotate “ANGLE Variable” on page 642 for the PIE function.

rotate

specifies the sweep of the arc. The value can be a number, a numeric constant, or a numeric variable. For valid values, see the Annotate “ROTATE Variable” on page 659 for the PIE function.

size

specifies the radius of the arc. The value can be a number, a numeric constant, or a numeric variable. For details, see the Annotate “SIZE Variable” on page 660.

color

specifies the color of the arc outline and optional fill using a character string without quotation marks. For valid values, see the Annotate “COLOR Variable” on page 645. Use an asterisk (*) to specify the previous value of the *color* parameter.

style

specifies the fill pattern for the slice or circle, using a character string without quotation marks. For details and valid values, see the Annotate “STYLE Variable (Patterns)” on page 662 for the PIE function.

line

specifies which lines of a pie slice are to be drawn. The value can be a number, a numeric constant, or a numeric variable. For valid values and details, see the “LINE Variable” on page 652 for the PIE function.

%SWAP Macro

Exchanges control between (XLAST, YLAST) and text (XLSTT, YLSTT) coordinates.

Variables written out: FUNCTION

Internal variables updated: XLAST, YLAST, XLSTT, YLSTT

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%SWAP;

%SYSTEM Macro

Defines the Annotate reference systems and the XSYS, YSYS, and HSYS variables.

Variables written out: HSYS, XSYS, YSYS

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%SYSTEM (*xsys*, *ysys*, *hsys*);

xsys*, *ysys*, *hsys

specify values that represent a coordinate system and an area of the output, as defined for the Annotate “XSYS Variable” on page 670. The default is %SYSTEM (4, 4, 4).

Details

Note: Not all coordinate systems are valid with all Annotate variables or all SAS/GRAPH procedures. See “Annotate Functions” on page 615 for any restrictions that apply to the variable that you want to use. △

The ZSYS variable cannot be set through this macro. Use an explicit variable assignment instead:

```
zsys='value'; output;
```

See Coordinate Systems“Coordinate Systems” on page 596 for a description of the areas and coordinate systems.

%TXT2CNTL Macro

Assigns the values of the text (XLSTT, YLSTT) coordinates to the control (XLAST, YLAST) coordinates.

Variables written out: FUNCTION

Internal variables updated: XLAST, YLAST

Prerequisite: You must run the %ANNOMAC macro before using any other annotate macros. For more information, see “Making the Macros Available” on page 697.

Syntax

%TXT2CNTL;

Details

Use the %TXT2CNTL macro when you want nontext functions to use the ending position of a text string as a starting or ending point.

Using Annotate Macros

Macro Structure

The general form of an Annotate macro is

```
%MACRO (parameters);
```

In general, the macro name represents a function and the parameters contain the values for the variables that can be used with the function. All macros except DCLANNO, SYSTEM, and SEQUENCE output an observation.

The parameters are either numeric or character. Numeric parameters can be numeric constants or numeric variable names that have been initialized to the appropriate value. Most character parameters must be expressed as literals, that is character strings without quotation marks. Exceptions are the text values that are used with the COMMENT and LABEL macros, which can be expressed as character strings enclosed in quotation marks or as character variable names.

The Annotate facility assigns the parameter values to the corresponding Annotate variables. Therefore, the observations in an Annotate data set that is created with macros that look the same as the ones that you created with assignment statements.

Making the Macros Available

To use Annotate macros, you must provide your program with access to the data set that contains the macros, and you must compile the macros before you use them. Check

with your SAS Software Consultant to find out if the fileref for the data set that contains the Annotate macros that are supplied with SAS/GRAPH software is allocated automatically at your site. Then access the Annotate macros in one of these ways:

- If the fileref is not set automatically, find out where the Annotate macros are stored and allocate a fileref that points to the data set:

```
filename fileref 'external-file';
```

Then include the Annotate macros in your session:

```
%include fileref (annomac);
```

- If the fileref is set automatically, compile the Annotate macros and make them available by simply submitting the ANNOMAC macro:

```
%annomac;
```

Note: The ANNOMAC macro must be run before any other Annotate macros are used in a SAS session. You will see a message in the SAS log that indicates that the Annotate macros are now available. The message also shows you how to get help for using the macros. \triangle

Annotate Macro Task Summary

The following table summarizes the tasks performed by the Annotate macros.

Table 25.1 Tasks with Annotate Macros

If you want to...	Use this macro...
assign values of (XLSTT,YLSTT) to (XLAST,YLAST)	%TXT2CNTL;
begin drawing a polygon	%POLY(x, y, color, style, line);
continue drawing a polygon	%POLYCONT(x, y, color);
copy (XLAST,YLAST) to (XLSTT,YLSTT)	%CNTL2TXT;
declare all variables	%DCLANNO;
draw a bar	%BAR(x1, y1, x2, y2, color, line, style);
draw a circle	%CIRCLE(x, y, size, color);
draw a frame	%FRAME(color, line, size, style);
draw a line from (XLAST,YLAST) to (XLSTT,YLSTT)	%DRAW2TXT(color, line, size);
draw a line from previous point	%DRAW(x, y, color, line, size);
draw a line	%LINE(x1, y1, x2, y2, color, line, size);
draw a pie slice or arc	%SLICE(x, y, angle, rotate, size, color, style, line);
draw a rectangle	%RECT(x 1,y 1,x 2,y 2, color, line, size);
draw text	%LABEL(x, y, text, color, angle, rotate, size, style, position);
exchange the values of (XLAST,YLAST) and (XLSTT,YLSTT)	%SWAP;
move to a point near a pie slice	%PIEXY(angle, size);

If you want to...	Use this macro...
move to a point without drawing	<code>%MOVE(x, y);</code>
put values into a stack	<code>%PUSH;</code>
retrieve values from a stack	<code>%POP;</code>
scale and move input	<code>%SCALET(<i>ptx, pty, x0, y0, x1, y1, x0, vy0, vx1, vy1</i>);</code>
scale input	<code>%SCALE(<i>ptx, pty, x0, y0, x1, y1, x0, vy0, vx1, vy1</i>);</code>
set the coordinate system for the observation	<code>%SYSTEM(<i>xsys, ysys, hsys</i>);</code>
set when to draw an observation	<code>%SEQUENCE(<i>when</i>);</code>
write a comment to the data set	<code>%COMMENT(<i>text</i>);</code>

Annotate Error Messages

If there is an error in your Annotate data set, one or more diagnostic messages are printed in the SAS log. A partial list of these messages is supplied here. Annotate data sets are checked for errors this way:

- If an error is found in preprocessing, this message appears:

```
NOTE: ERROR DETECTED IN ANNOTATE= libref.dataset
```

- If an error is found as an observation is being read, this message appears:

```
PROBLEM IN OBSERVATION number -- message
```

where *message* is the text of the error message.

- If the error limit of 20 errors is reached at any point during processing of the data set, a termination message similar to this one appears:

```
ERROR LIMIT REACHED IN ANNOTATE PROCESS
```

```
20 TOTAL ERRORS
```

Some common diagnostic messages are explained here.

A CALCULATED COORDINATE LIES OUTSIDE THE VISIBLE AREA

Explanation: The *x* or *y* coordinate is outside the display area (defined by HPOS= and VPOS= values).

User Action: Check for an invalid or misspecified coordinate system value, or *x* or *y* values outside displayed range.

A CALCULATED WINDOW COORDINATE LIES OUTSIDE THE WINDOW AREA

Explanation: the *x* or *y* coordinate is outside of the window area. This message may accompany the message for invalid coordinate system specification.

User Action: Check for an invalid or misspecified coordinate system value, or *x* or *y* values outside displayed range.

A PERCENTAGE VALUE LIES OUTSIDE 0 TO 100 BOUNDARIES

Explanation: The *x* or *y* value requested is negative or greater than 100 percent. This message is informational.

User Action: Check requested value for accuracy.

ANNOTATE MIDPOINT DATATYPE DOES NOT MATCH GCHART- INPUT WAS

Explanation: The MIDPOINT variable in the Annotate data set is character, and the GCHART midpoint is numeric or vice versa.

User Action: Check for misspelling or wrong variable assignment, or check for quotes in the assignment statement.

ANNOTATE GROUP DATATYPE DOES NOT MATCH GCHART- INPUT WAS

Explanation: The GROUP variable in the Annotate data set is character, and the GCHART group is numeric or vice versa.

User Action: Check for misspelling or wrong variable assignment, or check for quotes in the assignment statement.

ANNOTATE SUBGROUP DATATYPE DOES NOT MATCH GCHART- INPUT WAS

Explanation: The SUBGROUP variable in the Annotate data set is character, and the GCHART subgroup is numeric or vice versa.

User Action: Check for misspelling or wrong variable assignment, or check for quotes in the assignment statement.

BOTH OLD AND NEW VARIABLE NAMES ENCOUNTERED IN ANNOTATE= DATA SET

Explanation: Variables named both MIDPOINT and MIDPNT or SUBGROUP and SUBGRP occur in the Annotate data set.

User Action: Determine which variable has the proper values for the Annotate data set and either delete the other variable or rename MIDPNT to MIDPOINT and SUBGRP to SUBGROUP.

CALCULATED COORDINATES LIE COMPLETELY OFF THE VISIBLE AREA

Explanation: Both the x and y coordinates supplied are outside the visible display area.

User Action: Check for improper or inappropriate coordinate system specification or coordinates out of range.

CANNOT HAVE MISSING GROUP VALUE IF GROUPS ARE PRESENT

Explanation: The GROUP variable in the Annotate data set contains a missing value.

User Action: If the GROUP= option is specified in the GCHART procedure, the Annotate GROUP variable cannot contain missing values. Remove the missing value from the request. Check reference system for data-dependent request.

CANNOT HAVE SUBGROUP AND X/Y MISSING IN GCHART STREAM

Explanation: Data coordinate system was requested and the X, Y and SUBGROUP variables contain missing values.

User Action: The X, Y or SUBGROUP variable must have a value if a data coordinate system is requested. Check stream for improper request.

CANNOT OMIT GROUP VARIABLE IF GCHART GROUPS ARE PRESENT

Explanation: You used a data coordinate system and specified GROUP= in the GCHART procedure, but the Annotate data set does not contain the GROUP variable.

User Action: Supply the GROUP variable in the Annotate data set.

CHARACTER VALUE SHOWN IS NOT ON THE HORIZONTAL AXIS

Explanation: The specified value of the XC variable is not on the x axis of the graph or chart. The observation is ignored.

User Action: Check for misspelling, for uppercase or lowercase conflict, or for exclusion in an axis specification.

CHARACTER VALUE SHOWN IS NOT ON THE VERTICAL AXIS

Explanation: The specified value of the YC variable does not occur on the y axis of the graph or chart. The observation is ignored.

User Action: Check for misspelling, for uppercase or lowercase conflict, or for exclusion in an axis specification.

- CONFLICT BETWEEN PROCEDURE AXIS TYPE AND ANNOTATE DATA TYPE**
 Explanation: The axis type is character and the x and y coordinates are numeric or vice versa.
 User Action: Check values for proper type matching.
- DATA SYSTEM NOT SUPPORTED FOR THIS STATEMENT**
 Explanation: The data coordinate systems 1, 2, 7, 8 are not permitted for this statement.
 User Action: Choose a different reference system for this observation.
- DATA SYSTEM REQUESTED, BUT POINT IS NOT ON GRAPH**
 Explanation: The coordinate specified is not on displayed graph, and data coordinate system placement has been requested.
 User Action: Check for improper specification of data value or graph axis parameters, or incorrect system specification. If this occurs, you may be able to use percent of the data area to position Annotate graphics.
- G3D DATA SYSTEM REQUESTED, ALL SYSTEMS NOT DATA DEPENDENT**
 Explanation: Not all requested XSYS, YSYS, and ZSYS variable values are data values.
 User Action: If one variable in G3D annotation is data-dependent, all variables must be data-dependent. Either specify all points in the data coordinate system or use another reference system value.
- G3D DATA SYSTEM REQUESTED, VARIABLE CONTAINED MISSING VALUE**
 Explanation: The X, Y, or Z variable contained a missing value.
 User Action: All values in G3D data placement requests must be specified. Remove the missing value from the request.
- INTERNAL SYSTEM STACK OVERFLOW- TOO MANY PUSH FUNCTIONS**
 Explanation: The limit of stack positions has been exhausted. The maximum number of stack positions is system-dependent. Each PUSH operation uses one position; each POP frees one position for re-use.
 User Action: Rewrite the program section to decrease the number of values stored in the stack.
- INTERNAL SYSTEM STACK UNDERFLOW- TOO MANY POP FUNCTIONS**
 Explanation: The POP function has been issued with no values in the LIFO stack.
 User Action: Check for unequal numbers of PUSH versus POP functions. They can be unequal, but you cannot have more values moved with the POP function than are stored with the PUSH function. At least one PUSH must occur (it before) a POP can be issued.
- LABEL FUNCTION REQUESTED, BUT TEXT VARIABLE NOT ON DATA SET**
 Explanation: A TEXT variable has not been found for the LABEL function.
 User Action: If FUNCTION='LABEL', the TEXT variable must contain the string to be placed in the display area. Check for misspelling of variable name or specification of the wrong Annotate data set.
- LINE VALUE SPECIFIED IS NOT WITHIN LIMITS- $0 \leq L \leq 3$**
 Explanation: An invalid special line value has been specified.
 User Action: The LINE value specified was not acceptable for FUNCTION='BAR' or the RECT macro. Check function for definition of line values or previous value used in DATA step prior to this observation.
- LINE VALUE SPECIFIED IS NOT WITHIN LIMITS- $1 \leq L \leq 46$**
 Explanation: The LINE value specified is not in the range 1 through 46.
 User Action: Check for improper specification of data value. Line styles represented by the LINE values can be found in the line-type table "Specifying Line Types" on page 207.

MINIMUM VARIABLES NOT MET—AMBIGUITY PREVENTS SELECTION.

Explanation: The combinations of available X, Y, XC, YC, GROUP, MIDPOINT, and SUBGROUP variables do not identify the data-dependent values uniquely.

User Action: Check variable requirements and respecify.

MINIMUM VARIABLES NOT MET- MUST HAVE X/XC,Y/YC IN DATA SET

Explanation: The X, XC, Y, or YC variables have not been found in the Annotate data set.

User Action: The X or XC and Y or YC variables must be in the data set. This message represents a minimum validity check of the supplied Annotate data set.

POLYCONT ENCOUNTERED BEFORE POLY

Explanation: The POLYCONT function was encountered with no POLY function specification.

User Action: Probable sequencing error. Check for missing POLY command, improper ordering of polygon points, or interruption of POLY type commands by other valued functions. Also, check the value of WHEN for a mismatch.

"POLYCONT" INTERRUPTED

Explanation: A POLYCONT definition has been interrupted and resumed in the Annotate data set. This usually accompanies the error message

```
POLYCONT ENCOUNTERED BEFORE POLY
```

User Action: Check data stream for proper order.

POSITION VALUE INVALID- MUST BE ONE OF "0123...9ABCDEF"

Explanation: The value of the POSITION variable is not in range '0' through '9' or 'A' through 'F' or '<', '+', or '>' in a LABEL command.

User Action: Check desired value in POSITION description and correct.

REQUESTED POLYGON CONTAINS TOO MANY VERTICES (OBSERVATIONS)

Explanation: The maximum allocation for polygon points is exhausted. The maximum number of vertices is limited by a device's memory.

User Action: Define polygon with fewer points or break polygon into sections.

SYSTEM VALUE INVALID- MUST BE ONE OF "0123...9ABC"

Explanation: The value supplied for the XSYS, YSYS, or HSYS variable is not valid.

User Action: Check the desired value and correct the data set.

TEXT STRING EXTENDS BEYOND BOUNDARY OF SYSTEM DEFINED

Explanation: The text string is too long.

User Action: Check for excessive SIZE value or shorten the string. This error could be caused by HSYS='4' and a small value of the VPOS graphics option.

USE THE XC VARIABLE FOR DATA VALUES WHEN TYPE IS CHARACTER

Explanation: The X variable is character type in the Annotate data set when it should be numeric.

User Action: If character data are being plotted, use the XC variable to specify any data-related points pertaining to character values. If data are not character, omit quotes in X data value assignment.

USE THE YC VARIABLE FOR DATA VALUES WHEN TYPE IS CHARACTER

Explanation: The Y variable is character type in the Annotate data set when it should be numeric.

User Action: If character data are being plotted, use the YC variable to specify any data-related points pertaining to character values. If data are not character, omit quotes in Y data value assignment.

VALUE SHOWN IS NOT A VALID FONT OR PATTERN TYPE

Explanation: The value of the STYLE variable is not a valid font or pattern.

User Action: Check the value supplied for misspelling, truncation, and support in the FUNCTION description.

VALUE SHOWN IS NOT A VALID FUNCTION

Explanation: The value in the FUNCTION variable is not recognized as an available function.

User Action: Check for misspellings or truncation of value. Truncation can be corrected by specifying a length of 8 bytes in the LENGTH statement in the DATA step that generates the data set.

VALUE SHOWN IS NOT A VALID SIZE FACTOR

Explanation: The SIZE value of the variable is negative or excessive.

User Action: Check request or calculation for positive value result.

VARIABLE SHOWN HAS IMPROPER LENGTH IN ANNOTATE= DATA SET

Explanation: The length is incorrect for variable indicated. Either the length of the character string exceeds the length for the variable specified in a LENGTH statement, or the variable was not specified in a LENGTH statement.

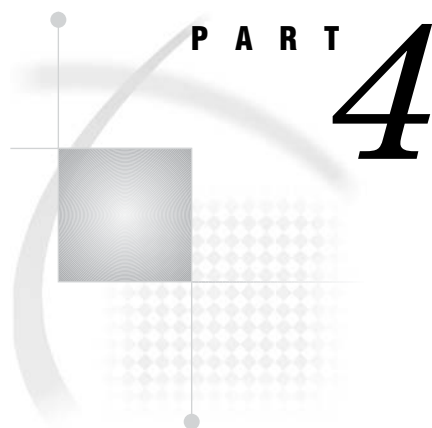
User Action: Make sure the variable length is defined in a length statement and that the length specified adequately covers the length of the character strings that are used.

VARIABLE SHOWN IS NOT OF THE PROPER DATA TYPE

Explanation: The data type does not match required type for variable listed.

Either variable type is character where a numeric is required, or numeric where a character is required.

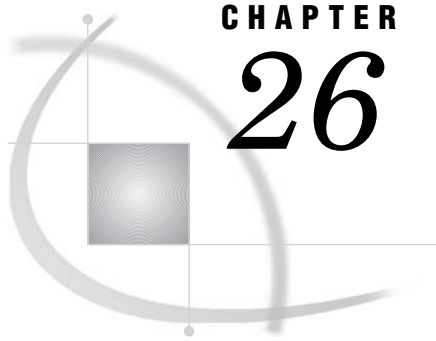
User Action: Specify proper type for variable as described in "Annotate Variables" on page 642.



SAS/GRAPH Procedures

- Chapter 26*. **The GANNO Procedure** 707
- Chapter 27*. **The GAREABAR Procedure** 725
- Chapter 28*. **The GBARLINE Procedure** 739
- Chapter 29*. **The GCHART Procedure** 773
- Chapter 30*. **The GCONTOUR Procedure** 885
- Chapter 31*. **The GDEVICE Procedure** 915
- Chapter 32*. **The GFONT Procedure** 939
- Chapter 33*. **The GIMPORT Procedure** 969
- Chapter 34*. **The GKEYMAP Procedure** 983
- Chapter 35*. **The GMAP Procedure** 995
- Chapter 36*. **The GOPTIONS Procedure** 1075
- Chapter 37*. **The GPLOT Procedure** 1081
- Chapter 38*. **The GPRINT Procedure** 1147
- Chapter 39*. **The GPROJECT Procedure** 1161
- Chapter 40*. **The GRADAR Procedure** 1183
- Chapter 41*. **The GREDUCE Procedure** 1213

<i>Chapter 42</i>	The GREMOVE Procedure	1223
<i>Chapter 43</i>	The GREPLAY Procedure	1237
<i>Chapter 44</i>	The GSLIDE Procedure	1277
<i>Chapter 45</i>	The GTESTIT Procedure	1285
<i>Chapter 46</i>	The G3D Procedure	1295
<i>Chapter 47</i>	The G3GRID Procedure	1327
<i>Chapter 48</i>	The MAPIMPORT Procedure	1347



CHAPTER 26

The GANNO Procedure

Overview 707

Procedure Syntax 708

PROC GANNO Statement 708

Examples 710

Example 1: Scaling Data-Dependent Output 710

Example 2: Storing Annotate Graphics 713

Example 3: Using the NAME= Option to Produce Multiple Graphs 715

Example 4: Using Annotate Graphics in a Drill-Down Graph 719

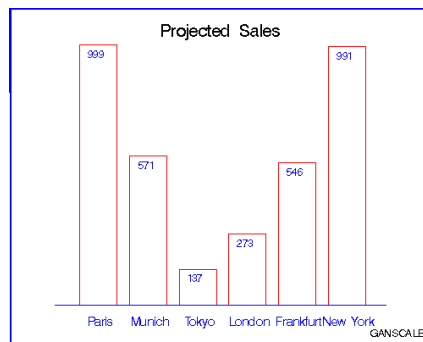
Overview

The GANNO procedure displays graphs created by Annotate data sets. The procedure can also be used to scale data-dependent graphics to fit the graphics output area. Note that the GANNO procedure ignores all currently defined title and footnote statements and some graphics option specifications, including `BORDER=`. To include titles, footnotes, and graphics options along with your Annotate data set, use the `GSLIDE` procedure instead of the `GANNO` procedure.

By default, both the `GANNO` and `GSLIDE` procedures scale graphics output from the data set to fill the entire graphics area. However, if you are using a data coordinate system and the data values are so large that some of the graphics elements do not fit in the graphics output area and are not displayed, you can use the `GANNO` procedure with the `DATASYS` option. This will cause the procedure to scale the output to fit the available space. The `GSLIDE` procedure does not have this capability.

Figure 26.1 on page 707 displays output from an Annotate data set.

Figure 26.1 Displaying Annotate Graphics with the GANNO Procedure



The program for this graph is in Example 1 on page 710.

Procedure Syntax

Requirements: An input Annotate data set is required.

Supports: Output Delivery System (ODS)

```
PROC GANNO ANNOTATE=Annotate-data-set
  <DATASYS>
  <DESCRIPTION='entry-description'>
  <GOUT=<libref.>output-catalog>
  <IMAGEMAP=output-data-set>
  <NAME='entry-name' | variable-name>;
```

PROC GANNO Statement

Identifies the Annotate data set and draws the graphics output defined by that data set. Optionally, it scales the output to accommodate data-dependent coordinate values and specifies an output catalog.

Syntax

```
PROC GANNO ANNOTATE=Annotate-data-set
  <DATASYS>
  <DESCRIPTION='entry-description'>
  <GOUT=<libref.>output-catalog>
  <IMAGEMAP=output-data-set>
  <NAME='entry-name' | variable-name>;
```

Required Arguments

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set that includes Annotate variables that identify graphics commands and parameters.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

Options

Options in the GANNO statement affect all graphs produced by that statement. You can specify as many options as you want and list them in any order.

DATASYS

indicates that absolute or relative data-dependent coordinates occur in the Annotate data set and scales the coordinates to fit the graphics output area. Use the DATASYS option only with Annotate data sets in which the coordinate system variables XSYS, YSYS, and HSYS specify the values 1, 2, 7, or 8.

Use the DATASYS option when graphics elements that were created with data-dependent variables do not fit in the graphics output area. This happens when the coordinate values generated by the data exceed a range of 0 to 100.

If you omit the DATASYS option, the GANNO procedure attempts to draw each graphics element according to the data values assigned to it, without scaling the values. If the range of data values is too large, some graphics elements will not display.

See also: “Using the DATASYS Option to Scale Graphs” on page 710

Featured in: Example 1 on page 710

DESCRIPTION=’entry-description’

DES=’entry-description’

specifies the description of the catalog entry for the chart. The maximum length is 256 characters. The description does not appear on the chart. By default, the GANNO procedure assigns the description OUTPUT FROM PROC GANNO.

Featured in: Example 2 on page 713

GOUT=<libref.>output-catalog

specifies the SAS catalog in which to save the graphics output produced by the GANNO procedure. If you omit the libref, the SAS/GRAPH software looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist.

See also: “Storing Graphics Output in SAS Catalogs” on page 53

Featured in: Example 2 on page 713

IMAGEMAP=output-data-set

creates a temporary SAS data set that is used to generate an image map in an HTML output file. The information in the image map data set includes the shape and coordinates of the elements in the graph and drill-down URLs that have been associated with those elements. The drill-down URLs are provided by one or two variables in the input data set. These variables are identified to the GANNO procedure with the HTML= and/or HTML_LEGEND= options.

The %IMAGEMAP macro generates the image map in the HTML output file. The macro takes two arguments, the name of the image map data set and the name or fileref of the HTML output file, as shown in the following example:

```
%imagemap(imgmapds, myimgmap.html);
```

See also: Chapter 25, “Annotate Dictionary,” on page 613 and “Adding Links with the HTML= and HTML_LEGEND= Options” on page 574.

Featured in: Example 4 on page 719

NAME=’entry-name’ | variable-name

specifies one of the following:

- the name of the catalog entry for the graph
- a variable name for each value for which a separate graph is produced.

If the value you assign to the NAME= option is enclosed in quotation marks, the procedure interprets it as a catalog entry name; if the value is not enclosed in quotes, the procedure interprets it as a variable name.

The value *entry-name* specifies the name of the catalog entry for the graph. The maximum length is 8 characters. The default name is GANNO. If the specified name duplicates the name of an existing entry, SAS/GRAPH software adds a number to the duplicate name to create a unique entry, for example, GANNO1.

If you specify *variable-name*, the GANNO procedure produces a separate graph for each different value of that variable. In addition, when you specify

NAME=*variable-name*, each value of the variable is used as the name of the catalog entry for that graph. A value that is longer than 8 characters is truncated. For example, if the value is **Frankfurt**, it is truncated to **Frankfur**. A second catalog entry would be **Frankfu1**. Consequently, you cannot use NAME='entry-name' at the same time.

Note: Specifying NAME=*variable-name* in the PROC GANNO statement produces results similar to those produced by the BY statement in a procedure that supports BY-group processing. See “BY Statement” on page 141 for details. \triangle

Featured in: Example 2 on page 713 Example 3 on page 715

Using the DATASYS Option to Scale Graphs

If your Annotate data set specifies a coordinate system that is based on data values (that is, XSYS, YSYS, and HSYS are assigned the values 1, 2, 7, or 8), the data values determine the size and location of the graphics elements on the output.

If the procedure that specifies the annotation generates axes (such as GPLOT or GCHART), by default the axes are scaled to accommodate the full range of data values and to fit in the procedure output area. Because all values are included in the axes, the graph displays all the Annotate output that is dependent on data values.

However, if the annotation displays with the GSLIDE or GANNO procedure, which do not generate axes, the data values may generate coordinate values that exceed the limits of the graphics output area.

In this case, you can use the DATASYS option to tell the procedure that the Annotate data set contains data-dependent coordinates and to scale the output accordingly. For an illustration of this process, see Example 1 on page 710.

When you use the DATASYS option, the GANNO procedure reads the entire input data set before drawing the graph and creates an output environment that is data dependent; that is, the environment is based on the minimum and maximum values that are contained in the data set. It then scales the data to fit this environment so that all graphics elements can be drawn.

Although the DATASYS option enables you to generate graphs using one of the data-dependent coordinate systems, it requires that the procedure scan the entire data set to determine the minimum and maximum data values. You can save this extra pass of the data set by using data-dependent values only in procedures that generate axes. Annotate coordinate system 5 (percent of the procedure output area) is recommended for use with the GANNO procedure. This coordinate system works equally well with the GSLIDE procedure if you decide to display the annotation with titles and footnotes.

Examples

Example 1: Scaling Data-Dependent Output

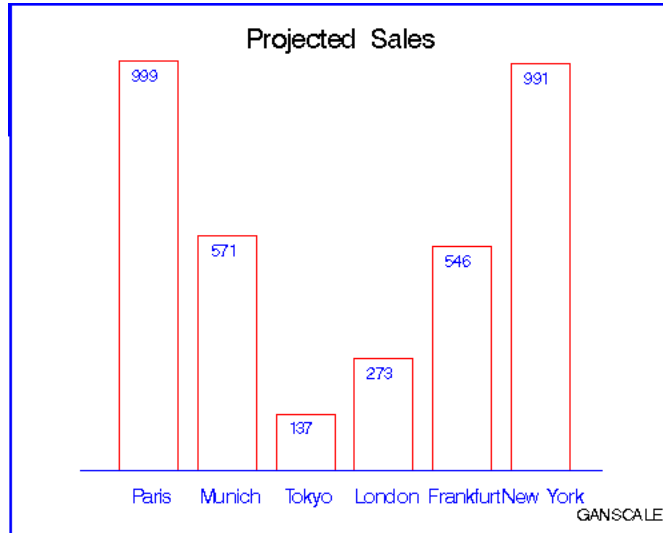
Procedure features:

PROC GANNO statement options:

```
ANNOTATE=
DATASYS
```

Sample library member: GANSCALE

Figure 26.2 Scaled GANNO Output



This example uses an Annotate data set to scale data-dependent output with the DATASYS option and create a vertical bar chart of sales for each of six sites. The values that determine the height of each bar range from 137 to 999. The range of values is so large that the GANNO procedure cannot fit all of the bars in the output area without scaling the output. This program uses the DATASYS option to scale the data values so that the bars fit in the graphics output area.

Set the graphics environment.

```
goptions reset=global gunit=pct cback=white
        colors=(black blue green red);
```

Create the data set WRLDTOTL. WRLDTOTL contains sales data for six sites. SITENAME contains the names of the sites. MEAN contains the average sales for each site.

```
data wrldtotl;
  length sitename $ 10;
  input sitename $ 1-10 mean 12-15;
  datalines;
Paris      999
Munich     571
Tokyo      137
London     273
Frankfurt  546
New York   991
;
run;
```

Create the Annotate data set, WRLDANNO. XSYS and YSYS specify coordinate system 2 (absolute data values) for X and Y. HSYS specifies coordinate system 3 (percent of the graphics output area) for SIZE. The SET statement processes every observation in WRLDTOTL.

```
data wrldanno;
  length function color $ 8 text $ 20;
  retain line 0 xsys ysys '2' hsys '3' x 8;
  set wrldtotl end=end;
```

Draw the bars. The MOVE function defines the lower left corner of the bar. The BAR function draws the bar. Bar height (Y) is controlled by MEAN.

```
function='move'; x=x+8; y=20; output;
function='bar'; y=y+(mean); x=x+9;
style='empty'; color='red'; output;
```

Label the bar with the name of site.

```
function='label'; y=0; x=x-4; size=3.5;
position='E'; style='swiss';
color='blue'; text=sitename; output;
```

Move to the top of the bar and write the value of MEAN.

```
function='move'; y=y+(mean)-3; output;
function='label'; x=x-1; text=left(put(mean,3.));
position='5'; style='swiss'; size=3; output;
```

After all the observations are processed, add an axis line, title, footnote, and frame. The MOVE and DRAW functions draw the axis line. The LABEL function writes the title and the footnote. The FRAME function draws a border around the output.

```
if end then do;
  function='move'; x=10; y=20; output;
  function='draw'; x=90; y=20; line=1;
  size=.5; color='blue'; output;
  function='label'; x=50; y=95; text='Projected Sales';
  xsys='3'; ysys='3'; position='5'; style='swissb';
  size=5; color=' '; output;
  x=92; y=5; size=3; style='swiss'; text='GANSCALE'; output;
  function='frame'; color='blue'; when='b';
  style='empty'; output;
end;
run;
```

Display the annotate graphics. The ANNOTATE= identifies the data set that contains the graphics commands. DATASYS tells the procedure to use the maximum and minimum data values to construct the output environment. In addition, the values of X and Y are scaled to fit the environment and all of the bars display on the graph.

```
proc ganno annotate=wrlldanno
  datasys;
run;
quit;
```

Example 2: Storing Annotate Graphics

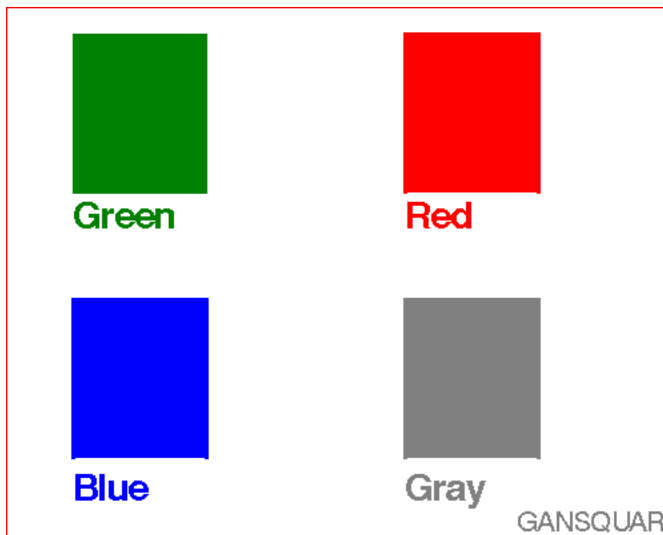
Procedure features:

PROC GANNO statement options:

DESCRIPTION=
GOUT=
NAME=

Sample library member: GANSQUAR

Figure 26.3 Four Squares



This example creates an Annotate data set that draws four colored squares, displays the data set as a single graphics output, and stores the output as a catalog entry in a permanent catalog. In this example, the NAME= option specifies a text string that identifies the name that is stored with the graphics output in the catalog.

Set the graphics environment.

```
goptions reset=global gunit=pct cback=white
  colors=(black blue green red);
```

Create the Annotate data set, SQUARES. XSYS and YSYS specify coordinate system 3 (percent of the graphics output area) for X and Y.

```
data squares;
  length function style color $ 8 text $ 15;
  xsys='3'; ysys='3';
```

Draw the first square. The COLOR variable assigns the color for the square. The FUNCTION variable selects the operation to be performed by the Annotate facility. The X and Y variables contain coordinate values. The BAR function draws the square. When the STYLE variable is used with the BAR function, it selects the fill pattern for the bar.

```
color='green';
function='move'; x=10; y=65; output;
function='bar'; x=30; y=95; style='solid'; output;
```

Label the first square. The LABEL function creates the label. The POSITION value of 6 left-justifies the text with respect to X and Y. The TEXT variable specifies the text string to be written.

```
function='label'; x=10; y=63; position='6';
style='swissb'; size=2; text='Green'; output;
```

Draw and label the second square.

```
color='red';
function='move'; x=60; y=65; output;
function='bar'; x=80; y=95; output;
function='label'; x=60; y=63; position='6';
style='swissb'; size=2; text='Red'; output;
```

Draw and label the third square.

```
color='blue';
function='move'; x=10; y=15; output;
function='bar'; x=30; y=45; output;
function='label'; x=10; y=12; position='6';
style='swissb'; size=2; text='Blue'; output;
```

Draw and label the fourth square.

```
color='gray';
function='move'; x=60; y=15; output;
function='bar'; x=80; y=45; output;
function='label'; x=60; y=12; position='6';
style='swissb'; size=2; text='Gray'; output;
```

Add a footnote.


```
x=88; y=5; position='5'; size=1.5; style='swiss';
text='GANSQUAR'; output;
```

Draw a red frame.

```
function='frame'; color='red'; when='b';
style='empty'; output;
run;
```

Display the annotate graphics. GOUT= assigns the catalog in which the graphics output is stored. NAME= assigns a name to the entry stored in the WORK.EXCAT catalog. DESCRIPTION= assigns a description to the catalog entry.

```
proc ganno annotate=squares
  gout=excat
  name='GANSQUAR'
  description='Four squares';
run;
quit;
```

Example 3: Using the NAME= Option to Produce Multiple Graphs

Procedure features:

PROC GANNO statement option:
NAME=

Sample library member: GANMULTI

In this example, the GANNO procedure uses the NAME= option to generate multiple graphs from one Annotate data set. Since NAME= is assigned the variable COLOR, the GANNO procedure generates separate graphics output for each value of the COLOR, as shown in Figure 26.4 on page 717, Figure 26.5 on page 718, Figure 26.4 on page 717 and Figure 26.6 on page 718.

Each output is stored as a separate entry in the temporary output catalog WORK.EXCAT. The entries are named according to the values of COLOR: **BLUE**, **GRAY**, **GREEN**, and **RED**. Note that the output for **GRAY** includes the footnote shown in Example 2 on page 713. The output for **RED** shows the frame that is generated by the Annotate data set. The black borders in the other outputs are not generated by the code.

Set the graphics environment.

```
goptions reset=global gunit=pct cback=white
colors=(black blue green red);
```

Create the Annotate data set, SQUARES. XSYS and YSYS specify coordinate system 3 (percent of the graphics output area) for X and Y.

```
data squares;
  length function style color $ 8 text $ 15;
  xsys='3'; ysys='3';
```

Draw the first square. The COLOR variable assigns the color for the square. The FUNCTION variable selects the operation to be performed by the Annotate facility. The X and Y variables contain coordinate values. The BAR function draws the square. When the STYLE variable is used with the BAR function, it selects the fill pattern for the bar.

```
color='green';
function='move'; x=10; y=65; output;
function='bar'; x=30; y=95; style='solid'; output;
```

Label the first square. The LABEL function creates the label. The POSITION value of 6 left-justifies the text with respect to X and Y. The TEXT variable specifies the text string to be written.

```
function='label'; x=10; y=63; position='6';
style='swissb'; size=2; text='Green'; output;
```

Draw and label the second square.

```
color='red';
function='move'; x=60; y=65; output;
function='bar'; x=80; y=95; output;
function='label'; x=60; y=63; position='6';
style='swissb'; size=2; text='Red'; output;
```

Draw and label the third square.

```
color='blue';
function='move'; x=10; y=15; output;
function='bar'; x=30; y=45; output;
function='label'; x=10; y=12; position='6';
style='swissb'; size=2; text='Blue'; output;
```

Draw and label the fourth square.

```
color='gray';
function='move'; x=60; y=15; output;
function='bar'; x=80; y=45; output;
function='label'; x=60; y=12; position='6';
style='swissb'; size=2; text='Gray'; output;
```

Add a footnote.

```
x=88; y=5; position='5'; size=1.5; style='swiss';
text='GANSQUAR'; output;
```

Draw a red frame.

```
function='frame'; color='red'; when='b';  
style='empty'; output;  
run;
```

Generate the annotate graphics, separating graphs by color. NAME= identifies the variable whose values PROC GANNO uses to generate the output. GANNO produces separate output for each value of COLOR. The COLOR value is the name of the catalog entry.

```
proc ganno annotate=squares  
name=color  
gout=excat  
description='Individual squares';  
run;
```

Figure 26.4 Output for COLOR Value BLUE (WORK.EXCAT.BLUE.GRSEG)

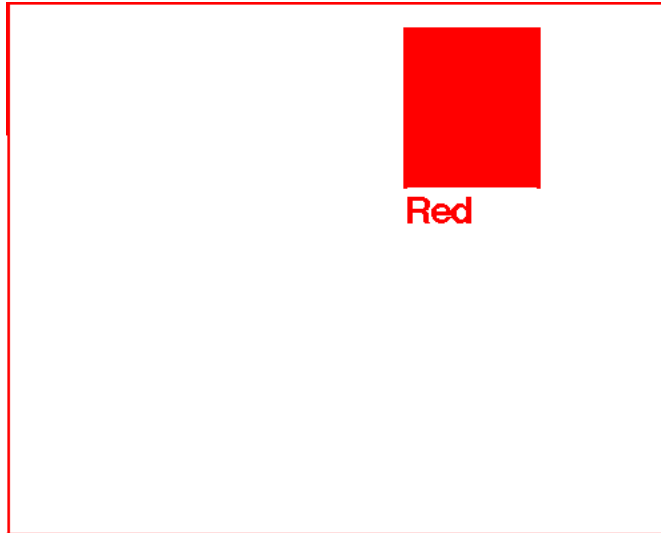


Figure 26.5 Output for COLOR Value GRAY (WORK.EXCAT.GRAY.GRSEG)



Figure 26.6 Output for COLOR Value GREEN (WORK.EXCAT.GREEN.GRSEG)



Figure 26.7 Output for COLOR Value RED (WORK.EXCAT.RED.GRSEG)

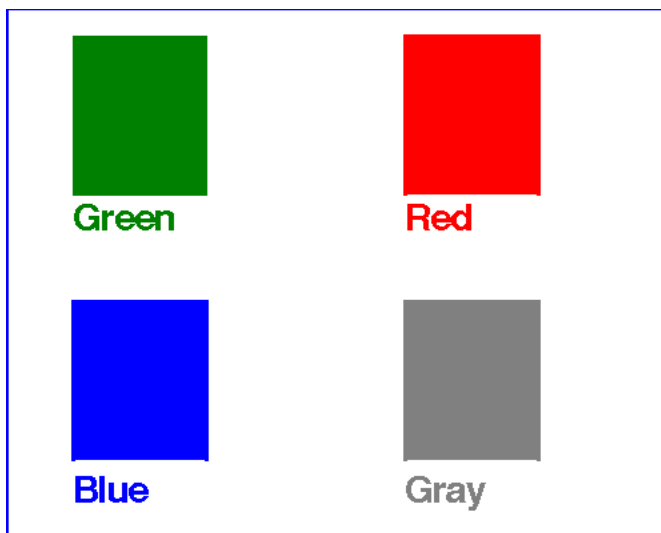
Example 4: Using Annotate Graphics in a Drill-Down Graph

Procedure features:

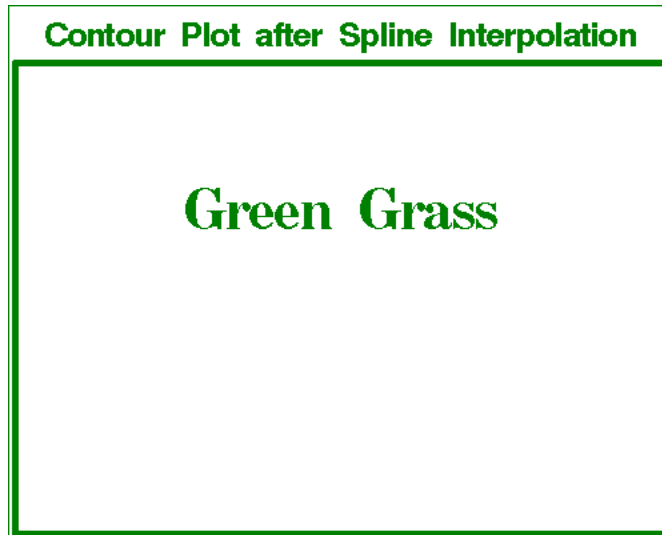
PROC GANNO statement option:
IMAGEMAP=

Sample library member: GANDRILL

This example creates essentially the same Annotate data set used in Example 2 on page 713. It draws four colored squares and displays the data set as a single graphics output.



However, this time the example shows you how to use Annotate graphics to generate a drill-down graph. The example uses the HTML variable in the Annotate data set to specify linking information that defines each of the four squares as a hot zone. When the graph is viewed in a browser, you can click on a square to drill down to a related graph. For example, if you click on the green square, it drills down to a graph that confirms that you selected the green square.



The example uses the HTML device driver to generate the drill-down graph. To implement the drill-down capability, the Annotate data set uses the HTML variable to provide the linking information (see “HTML Variable” on page 651), and the GANNO procedure uses the IMAGEMAP= option to create an Imagemap data set. The presence of the HTML variable in the Annotate data set and the IMAGEMAP= option on the GANNO procedure causes the HTML device driver to generate an image map for the graph. It writes the image map to the file index.html, which the HTML device driver creates for displaying Web output (see Chapter 17, “Generating Web Output with the Annotate Facility,” on page 499).

To prevent the HTML device driver from writing over the contents of index.html after the drill-down graph has been generated, the example switches to the GIF device driver. It then runs four GSLIDE procedures to generate the target output. Each GSLIDE procedure uses the NAME= option to name the graph it produces, ensuring that the GIF driver creates files named green.gif, blue.gif, red.gif, and gray.gif. These are the files that are referenced as targets by the strings that are specified for the Annotate data set’s HTML variable.

Allocate a storage location for all the output files, and set the graphics environment.

The HTML device driver generates output that includes both HTML and GIF files, so the libref must point to an aggregate storage location. It cannot point to a file.

```
/* define the output location */
filename webout 'path-to-Web-server';
/* set the graphics environment */
goptions reset=global gunit=pct
        colors=(black blue green red);
```

Create the Annotate data set. The HTML variable is used to define the linking information for each square. Because the GSLIDE procedures that generate the target output use NAME= to ensure the output files are named green.gif, red.gif, blue.gif, and gray.gif, strings that reference those names are assigned to the HTML variable for the appropriate observation in the data. For the final observation, the HTML variable's value is set to a null string; otherwise it would retain the last assigned value, which is **href=gray.gif**. In that case, the graph's background area would be defined as a hot zone that links to file gray.gif. For a description of the other functions and variables used in the Annotate data set, see Example 2 on page 713.

```

/* create Annotate data set */
data squares;
  length function style color $ 8
         html text $ 15;
  xsys='3'; ysys='3';

  /* draw the green square */
  color='green';
  function='move'; x=10; y=65; output;
  function='bar'; x=30; y=95; style='solid';
  html='href=green.gif'; output;

  /* label green square */
  function='label'; x=10; y=63; position='6';
  style='swissb'; size=2; text='Green'; output;

  /* draw the red square */
  color='red';
  function='move'; x=60; y=65; output;
  function='bar'; x=80; y=95;
  html='href=red.gif'; output;

  /* label red square */
  function='label'; x=60; y=63; position='6';
  style='swissb'; size=2; text='Red'; output;

  /* draw the blue square */
  color='blue';
  function='move'; x=10; y=15; output;
  function='bar'; x=30; y=45;
  html='href=blue.gif'; output;

  /* label blue square */
  function='label'; x=10; y=12; position='6';
  style='swissb'; size=2; text='Blue'; output;

  /* draw the gray square */
  color='gray';
  function='move'; x=60; y=15; output;
  function='bar'; x=80; y=45;
  html='href=gray.gif'; output;

  /* label gray square and add a footnote */
  function='label'; x=60; y=12; position='6';
  style='swissb'; size=2; text='Gray'; output;

```

```

        /* draw a blue frame */
        function='frame'; color='blue'; style='empty';
        /* set null link for background area in frame */
        html=''; output;
run;

```

Set the graphics environment for the Web page. DEV= specifies the HTML device driver, which will create the HTML and GIF files needed for the Web page. GSFNAME= specifies the libref that points to the storage location that was allocated for the Web output. XPIXELS= and YPIXELS= define a size in pixels for the graphics area. TRANSPARENCY specifies that the background areas in all generated graphs should appear to be transparent when the images are displayed in a browser.

```

/* set the graphics options for the web page */
goptions dev=html gsfname=webout
        xpixels=450 ypixels=400
        transparency;

```

Generate the drill-down graph. IMAGEMAP= specifies ANNOMAP as the name for the Imagemap data set.

```

/* generate annotate graphics */
proc ganno annotate=squares
        imagemap=annomap
        description='Four squares';
run;

```

Change to the GIF driver and generate the target output. DEV= changes the device driver to GIF so that the target output files will be generated as GIF files. FTEXT= and CTEXT= specify a font and color for the text in graphics output. PROC GSLIDE is then run four times to generate the four graphs that will serve as target output for the links that are defined in the drill-down graph.

```

/* change to gif driver for target output */
goptions dev=gif ftext=centb ctext=green;

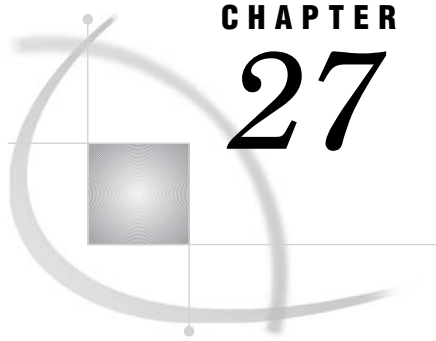
/* generate the target output */
proc gslide wframe=4
        cframe=green name='green';
        note height=20;
        note height=10
                justify=center
                'Green Grass';
run;

goptions ctext=blue;
proc gslide wframe=4
        cframe=blue name='blue';
        note height=20;
        note height=10
                justify=center

```



```
        'Blue Sky';  
run;  
  
goptions ctext=red;  
proc gslide wframe=4  
    cframe=red name='red';  
    note height=20;  
    note height=10  
        justify=center  
    'Red Wine';  
run;  
  
goptions ctext=gray;  
proc gslide wframe=4  
    cframe=gray name='gray';  
    note height=20;  
    note height=10  
        justify=center  
    'Gray Mare';  
run;  
quit;
```

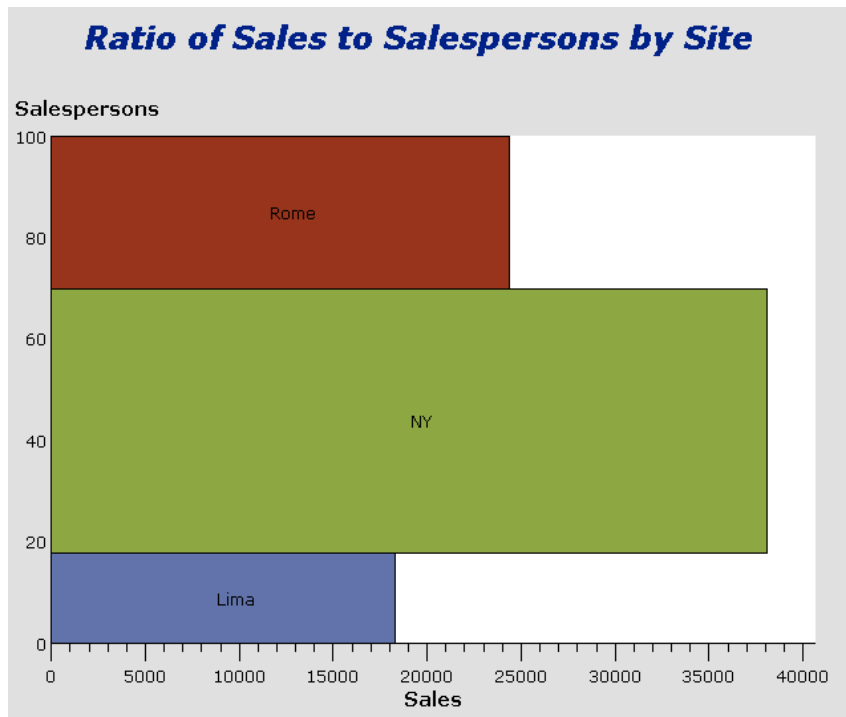
CHAPTER 27

The GAREABAR Procedure

<i>Overview</i>	725
<i>Concepts</i>	726
<i>Procedure Syntax</i>	727
<i>PROC GAREABAR Statement</i>	727
<i>HBAR, HBAR3D, VBAR, VBAR3D Statements</i>	728
<i>Examples</i>	729
<i>Example 1: A Simple Area Bar Chart</i>	729
<i>Example 2: Area Bar Chart with a Numeric Category Variable</i>	731
<i>Example 3: Area Bar Chart with a Subgrouping</i>	733
<i>Example 4: Area Bar Chart with Subgrouping and RSTAT and WSTAT as Percentages</i>	735

Overview

The GAREABAR procedure enables you to produce an area bar chart showing the magnitudes of *two* variables for each category of data. For example, the following area bar chart shows the sales total for each of three geographical sites. An additional dimension is graphed for the width variable, namely, the number of sales persons at each site. One can see from this chart that while the sales total of NY is the greatest, its number of sales people is also the greatest. In a plain bar chart, the width is the same for each bar. In an area bar chart, the width and height of each bar is determined by the value of variables.



Note: The GAREABAR procedure requires the following:

- a prior GOPTIONS statement where DEV=ACTIVEX or DEV=ACTXIMG, because PROC GAREABAR only works with the ActiveX control on a Windows system
- a prior ODS HTML statement
- a closing ODS HTML CLOSE statement.

Δ

Concepts

The GAREABAR procedure produces a chart based on the values of a *category variable*. A category variable can be either character or numeric. The GAREABAR procedure treats all values of a numeric category variable as DISCRETE (such as quarters 1,2,3,4) even if the values are apparently continuous (such as 1.234 and 4.002). PROC GAREABAR does not calculate a midpoint.

Also, PROC GCHART, by default, sorts the values of the category variable either alphabetically or numerically. PROC GAREABAR displays the category variable in data order (the order in which its values occur in the data set).

For each category variable, PROC GAREABAR graphs the dimensions of two numeric variables. For the VBAR statement, the SUMVAR variable (response variable) is graphed along the vertical axis, and the width variable is graphed along the horizontal axis. Conversely, for the HBAR statement, the SUMVAR variable (response variable) determines the length of the bar on the horizontal axis, and the width variable determines the thickness of the bar on the vertical axis.

Both the width variable and the response variable can be displayed as either percentage or sum. Specify WSTAT=PERCENT or WSTAT=SUM for the width variable, and specify RSTAT=PERCENT or RSTAT=SUM for the response variable. The default for both is SUM.

In addition, you can use the SUBGROUP option to subgroup the response variable either by percentage or by sum. Examples of subgrouping are shown in Example 3 on page 733 and Example 4 on page 735.

Procedure Syntax

Requirements:

GOPTIONS statement with DEV=ACTIVEX | ACTXIMG
 ODS HTML statement (before and after)
 HBAR, HBAR3D, VBAR, or VBAR3D statement

Global statements: FOOTNOTE, GOPTIONS, TITLE

Supports: RUN-group processing

PROC GAREABAR<DATA=*input-data-set*>

VBAR *category-variable*width-variable* </ SUMVAR=*response-variable option(s)*>;

VBAR3D *category-variable*width-variable* </ SUMVAR=*response-variable option(s)*>;

HBAR *category-variable*width-variable* < / SUMVAR=*response-variable option(s)*>;

HBAR3D *category-variable*width-variable* < / SUMVAR=*response-variable option(s)*>;

PROC GAREABAR Statement

Identifies the data set containing the category variable, the response variable (SUMVAR), and the width variable.

Requirements: An input data set is required. If none is specified, the procedure uses the most recently created data set.

Syntax

PROC GAREABAR<DATA=*input-data-set*>;>

Requirements

input-data-set

contains the data to be graphed.

GOPTIONS DEV=ACTIVEX|ACTXIMG

PROC GAREABAR requires a GOPTIONS statement where the value of DEV= is ACTIVEX or ACTXIMG (before the PROC).

ODS HTML

PROC GAREABAR requires an ODS HTML statement (both before and after the PROC).

HBAR, HBAR3D, VBAR, VBAR3D Statements

These statements create area bar charts where each bar shows two dimensions (a width variable and response variable) for each category variable.

Syntax

```
HBAR | HBAR3D | VBAR | VBAR3D category-variable*width-variable
  </ SUMVAR=response-variable option(s)>;
```

Required Arguments

All arguments must be in the input data set.

category-variable

is either character or numeric. Defines the category of data to chart. Each category variable results in a separate bar. If the category-variable is numeric, all its values are treated as DISCRETE (such as the years 2000, 2001, 2002). No midpoint is calculated for a numeric category variable.

response-variable

is always numeric. For HBAR and HBAR3D, the length of the each bar along the horizontal axis represents the response variable. For VBAR and VBAR3D, the height of the each bar along the vertical axis measures the response variable.

Use the RSTAT option to specify whether the response variable is measured as a percentage or as a sum. The default is SUM.

width-variable

is always numeric. For HBAR and HBAR3D, the width variable is measured by the magnitude of each bar along the vertical axis. For VBAR and VBAR3D, the width variable is measured by the magnitude of each bar along the horizontal axis.

Use the WSTAT option to specify whether the width variable is measured as a percentage or as a sum. The default is SUM.

Options

CFR= | CFRAME=*background-color*

specifies a background color for the chart. The specified color must be a valid SAS/GRAPH color name, or a character variable of length eight whose value is the color. The default color is white.

CTEXT=*text-color*

specifies a color for all text on the chart. The specified color must be a valid SAS/GRAPH color name, or a character variable of length eight whose value is the color. The default color is black.

DISCRETE

causes the chart to show discrete width role values on the width axis rather than a continuous axis. If you do not specify DISCRETE, the continuous axis result is assumed. .

FRAME | NOFRAME

FRAME (the default) draws a frame around the procedure output area. The frame color is the first color in the colors list.

NOFRAME suppresses the frame that is drawn around the chart by default.

NAME=

when DEV=ACTXIMG, specifies the name of the graph (PNG file) produced by GAREABAR. The maximum length for entry-name is eight characters. If the specified name duplicates the name of an existing entry, then SAS/GRAPH software overwrites the existing entry.

Use the PATH or GPATH option of the ODS HTML statement or ODS MARKUP statement to specify the location for storing the .png file.

SUBGROUP=

used to subdivide the response-variable dimension (SUMVAR). A SUBGROUP variable can be either character or numeric. For example, if the category variable is company, and the response variable is revenue, then specifying a SUBGROUP of country will subdivide the revenue for each company according to country. A numeric example is subdividing revenue by quarters: 1, 2, 3, 4.

WSTAT= | WIDTHSTAT=SUM or PCT | PERCENT

specifies whether the width variable is represented as a percentage or as a sum. The default is SUM.

RSTAT= | RESPSTAT=SUM or PCT | PERCENT

If SUBGROUP is specified, then RSTAT specifies whether the subgrouping is represented as a percentage or as a sum. The default is SUM.

If SUBGROUP is *not* specified, then the response variable can only be SUM.

Examples

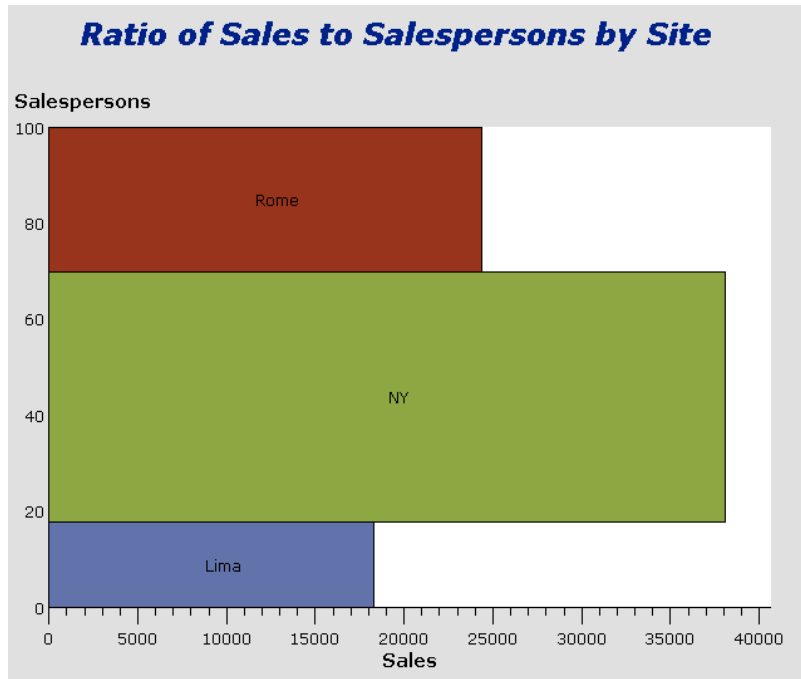
Example 1: A Simple Area Bar Chart

Procedure features:

SUMVAR=

Sample library member: GABSUMVR

This example graphs the total sales for each of three geographic sites (Rome, NY, Lima) along the X axis. Along the Y axis, the relative thickness of each bar shows the number of salespersons at each site. The chart shows that although NY had the highest sales (the longest bar), it also had the greatest number of salespersons (as shown by the thickness of the bar).



The procedure for this chart is:

Uncomment this line, and change the output destination to a directory and file name that makes sense for you.

```
*filename odsout 'c:\test\filename.htm';
```

PROC GAREABAR is only supported with device= **ACTIVEVEX** or **ACTXIMG**.

```
goptions reset=all dev=activex;
ods html file=odsout;

data totals;
  input Site $ Quarter Sales Salespersons;
  cards;
Lima    1  4043.97    4
NY      1  8225.26   12
Rome    1  3543.97    6
Lima    2  3723.44    5
NY      2  8595.07   18
Rome    2  5558.29   10
Lima    3  4437.96    8
NY      3  9847.91   24
Rome    3  6789.85   14
Lima    4  6065.57   10
NY      4 11388.51   26
Rome    4  8509.08   16
;
;
```


Because SUMVAR=SALES, the total sales are plotted along the horizontal axis (HBAR). Because SITE*SALESPERSONS and WSTAT=PERCENT, the percentage of salespersons at each site is shown by the relative thickness of each bar along the vertical axis.

```
proc gareabar data=totals;
  hbar site*salespersons /sumvar=sales wstat=PERCENT;
run;
```

ODS HTML CLOSE causes the HTML file to be written to disk.

```
ods html close;
```

The variables in this procedure are as follows:

site	the category variable: Lima, NY, Paris
salespersons	the width variable, in this case displayed as a percentage along the vertical axis
sales	the response variable (SUMVAR), displayed along the horizontal axis because, in this case, the statement is HBAR.

Example 2: Area Bar Chart with a Numeric Category Variable

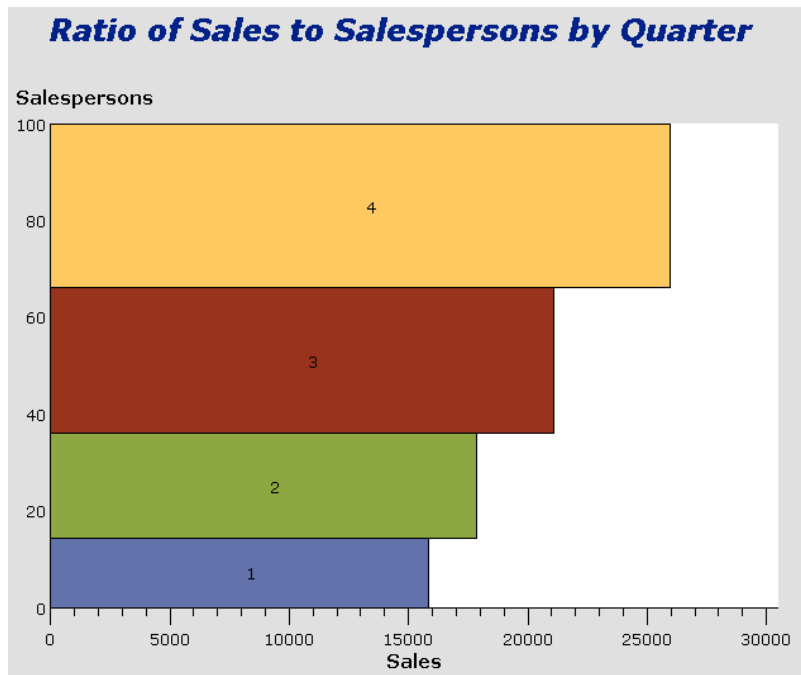
Procedure features:

SUMVAR=

Sample library member: GABSUMVR

This example is similar to Example 1 and shows that the category variable can be numeric—in this case 1, 2, 3, 4 for the four quarters of a year. The GAREABAR procedure treats all values of a numeric category variable as DISCRETE and does not calculate a midpoint even if the values of the category variable are continuous.

This example graphs the total sales for each quarter of the year along the horizontal axis. The relative thickness of each bar along the vertical axis shows the total number of salespersons during that quarter. The chart shows that as the number of salespersons increased from quarter to quarter, the total sales also increased.



The procedure is as follows:

Uncomment this line, and change the output destination to a directory and file name that makes sense for you.

```
*filename odsout 'c:\test\filename.htm';
```

PROC GAREABAR is only supported with device= ACTIVEX or ACTXIMG.

```
goptions reset=all dev=activex;
ods html file=odsout;

data totals;
input Site $ Quarter Sales Salespersons;
cards;
Lima 1 4043.97 4
NY 1 8225.26 12
Rome 1 3543.97 6
Lima 2 3723.44 5
NY 2 8595.07 18
Rome 2 5558.29 10
Lima 3 4437.96 8
NY 3 9847.91 24
Rome 3 6789.85 14
Lima 4 6065.57 10
NY 4 11388.51 26
Rome 4 8509.08 16
;
```

Because SUMVAR=SALES, the total sales are plotted along the horizontal axis (HBAR).

Because QUARTER*SALESPERSONS and WSTAT=PERCENT, the percentage of salespersons for each quarter is shown by the relative thickness of each bar along the vertical axis.

```
proc gareabar data=totals;
  hbar quarter*salespersons / sumvar=sales wstat=PCT;
run;
```

ODS HTML CLOSE causes the HTML file to be written to disk.

```
ods html close;
```

The variables in the example are as follows:

quarter	the category variable: quarters 1, 2, 3, and 4
salespersons	the width variable, in this case displayed as a percentage along the vertical axis.
sales	the response variable (SUMVAR), displayed along the horizontal axis.

Example 3: Area Bar Chart with a Subgrouping

Procedure features:

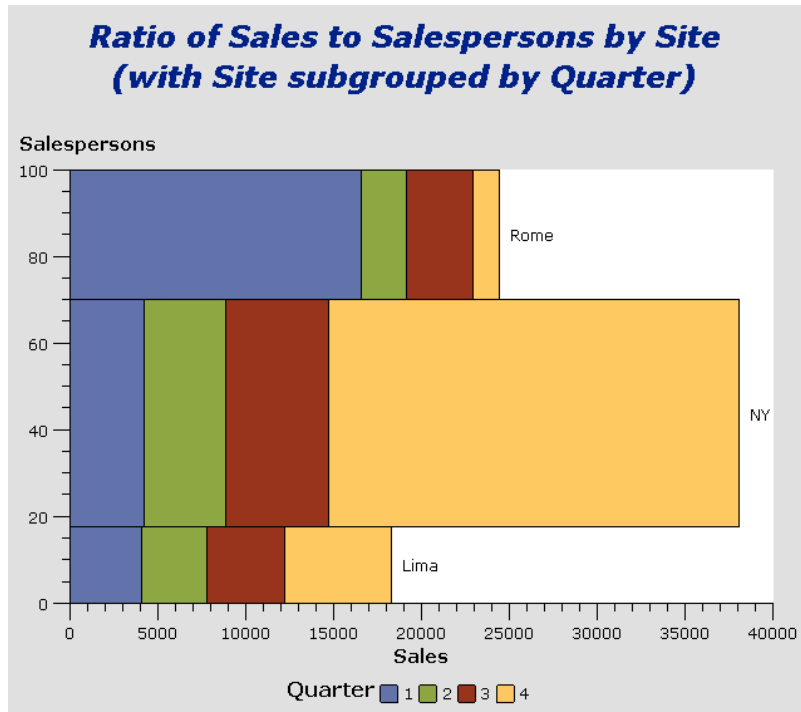
SUBGROUP=

Sample library member: GABSUBGR

This example uses the SUBGROUP= option to display the same magnitudes as displayed by Examples 1 and 2. Like Example 1, this example shows the total sales for each of three geographic sites along the horizontal axis. The relative thickness of each bar along the vertical axis shows the number of salespersons at each site.

In addition, by subgrouping the response variable by quarter, this example shows the relative percentage of sales for each quarter. Thus, one can see from this chart that NY (the middle bar) had most of its sales in the fourth quarter, whereas Rome (the topmost bar) had most of its sales in the first quarter.

The value of SUBGROUP= can be character or numeric.



The procedure is as follows:

Uncomment this line, and change the output destination to a directory and file name that makes sense for you.

```
*filename odsout 'c:\test\filename.htm';
```

PROC GAREABAR is only supported with device= ACTIVEX or ACTXIMG.

```
goptions reset=all dev=activex;
ods html file=odsout;

data totals;
  input Site $ Quarter $ Sales Salespersons;
cards;
Lima    1  4043.97    4
NY      1  4225.26   12
Rome    1 16543.97    6
Lima    2  3723.44    5
NY      2  4595.07   18
Rome    2  2558.29   10
Lima    3  4437.96    8
NY      3  5847.91   24
Rome    3  3789.85   14
Lima    4  6065.57   10
NY      4 23388.51   26
Rome    4  1509.08   16
;
```

```

/* define title */
title1 'Ratio of Sales to Salespersons by Site';
title2 '(with Site subgrouped by Quarter)';

```

Because SUMVAR=SALES, the total sales are plotted along the horizontal axis (HBAR).

Because SITE*SALESPERSONS and WSTAT=PERCENT, the percentage of salespersons for each quarter is shown by the relative thickness of each bar along the vertical axis.

Because SUBGROUP=QUARTER and RSTAT=SUM, the quarters are displayed as absolute numbers along the horizontal bar.

```

proc gareabar data=totals;
  hbar site*salespersons /sumvar=sales
                                subgroup=quarter
                                rstat=SUM
                                wstat=PCT;
run;
ods html close;

```

The variables in the example are as follows:

site	the category variable: Lima, NY, Rome.
salespersons	the width variable, in this case displayed as a percentage (wstat=PCT) along the vertical axis.
sales	the response variable (SUMVAR), displayed as a sum (rstat=SUM) along the horizontal axis because, in this case, the statement is HBAR.
quarter	quarters 1, 2, 3, and 4, displayed as absolute numbers (rstat=SUM) along the horizontal bar.

Example 4: Area Bar Chart with Subgrouping and RSTAT and WSTAT as Percentages

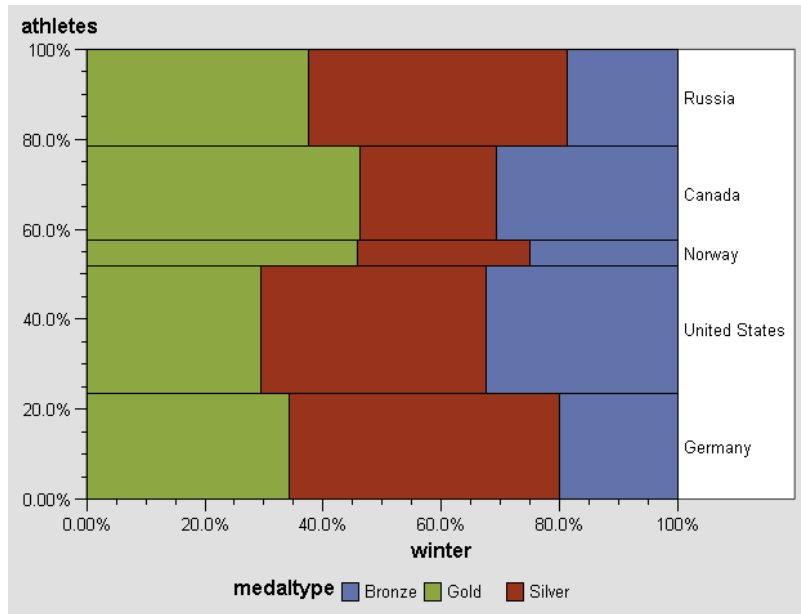
Procedure features:

SUBGROUP=, RSTAT=, WSTAT=

Sample library member: GABWSTAT

This example uses the RSTAT= option, in conjunction with the SUBGROUP= option, to display the response variable (medals won in the winter Olympics), subgrouped by the percentage (RSTAT=PCT) of each medal type (gold, silver, bronze). The width variable is the number of athletes, displayed (in this case along the vertical axis) as the percentage of athletes (WSTAT=PCT) of each of five different nationalities.

When the SUBGROUP= option is specified, you can use the RSTAT= option to specify whether the subgrouping is to be displayed as a percentage or as a sum.



The procedure is as follows:

Uncomment this line, and change the output destination to a directory and file name that makes sense for you.

```
*filename odsout 'c:\test\filename.htm';
```

PROC GAREABAR is only supported with device= ACTIVEX or ACTXIMG.

ODS LISTING CLOSE prevents the output from going to the OUTPUT window in addition to disk.

```
ods listing close;
ods html file=odsout;
goptions dev=activex;

data medals;
input country $15. medaltyp $ winter summer athletes;
datalines;
Germany      Gold   12 14 176
Germany      Silver 16 17   0
Germany      Bronze  7 26   0
United States Gold   10 39 210
United States Silver 13 25   0
United States Bronze 11 33   0
Norway       Gold   11  4  42
Norway       Silver  7  3   0
Norway       Bronze  6  3   0
Canada       Gold    6  3 157
Canada       Silver  3  3   0
Canada       Bronze  4  8   0
Russia       Gold    6 32 160
Russia       Silver  7 28   0
```

```
Russia          Bronze  3 28  0
;
```

Because SUMVAR=WINTER, this proc displays the number medals won in the winter Olympics, subgrouped by the percentage (RSTAT=PCT) of each medal type (gold, silver, bronze). The width variable is the number of athletes, displayed (in this case along the vertical axis) as the percentage of athletes (WSTAT=PCT) of each of five different nationalities.

When the SUBGROUP= option is specified, you can use the RSTAT= option to specify whether the subgrouping is to be displayed as a percentage or as a sum.

```
PROC GAREABAR data=medals;
  hbar country*athletes /sumvar=winter
                        subgroup=medaltype
                        wstat=percent
                        rstat=percent;
run;
quit;
```

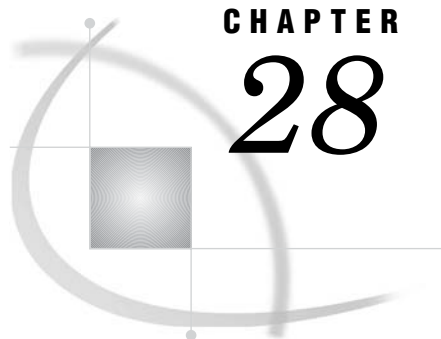
ODS HTML CLOSE causes the HTML file to be written to disk.

ODS LISTING restores subsequent output to the OUTPUT window.

```
ods html close;
ods listing;
```

The variables in the example are as follow:

country	the category variable: Russia, Canada, Norway, United States, Germany.
athletes	displayed as a percentage (WSTAT=PERCENT) along the vertical axis.
winter	the value of SUMVAR, the number of medals won in the winter Olympics, displayed along the horizontal axis because, in this case, the statement is HBAR.
medaltype	the value of SUBGROUP, displayed as a percentage (RSTAT=PERCENT) of each medal type (gold, silver, bronze).



CHAPTER 28

The GBARLINE Procedure

<i>Overview</i>	739
<i>About Bar Line Charts</i>	740
<i>About Interpolation Methods</i>	740
<i>Concepts</i>	741
<i>About the Bar Variable</i>	742
<i>About Midpoints</i>	742
<i>Character Values</i>	742
<i>Discrete Numeric Values</i>	743
<i>Continuous Numeric Values</i>	744
<i>Selecting and Ordering Midpoints</i>	744
<i>About the Plot Variable</i>	745
<i>About Chart Statistics</i>	745
<i>Frequency</i>	745
<i>Cumulative Frequency</i>	745
<i>Percentage</i>	746
<i>Cumulative Percentage</i>	746
<i>Sum</i>	746
<i>Mean</i>	746
<i>Calculating Weighted Statistics</i>	746
<i>Missing Values</i>	747
<i>Plot Variable Values Out of Range</i>	747
<i>About Patterns</i>	748
<i>Default Patterns and Outlines</i>	748
<i>User-Defined Patterns, Outlines, and Images</i>	748
<i>Version 6 Patterns</i>	749
<i>Procedure Syntax</i>	749
<i>PROC GBARLINE Statement</i>	750
<i>BAR Statement</i>	751
<i>PLOT Statement</i>	765
<i>Examples</i>	768
<i>Example 1: Producing a Basic Bar Line Graph with Styles</i>	768
<i>Example 2: Calculating Weighted Statistics</i>	770

Overview

The GBARLINE procedure produces bar line charts. Bar line charts are vertical bar charts with a plot overlay. These charts graphically represent the value of a statistic calculated for one or more variables in an input SAS data set. The charted variables can be either numeric or character.

The procedure calculates these statistics:

- frequency or cumulative frequency counts
- percentages or cumulative percentages
- sums
- means.

Use the GBARLINE procedure to

- display and compare exact and relative magnitudes
- examine the contribution of parts to the whole
- analyze where data are out of balance
- display long series of data, showing trends and patterns.

In conjunction with the SYMBOL statement, the GBARLINE procedure can produce needle plot overlays, and overlay plots with stepped interpolation.

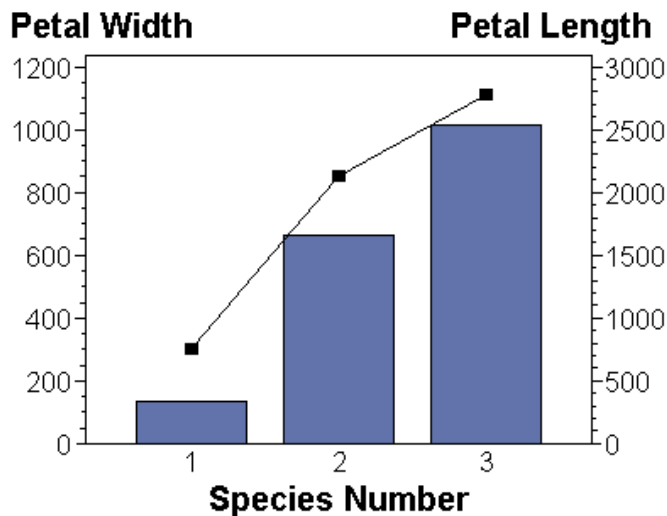
Note: PROC GBARLINE is not supported by Java. △

About Bar Line Charts

Bar line charts display the magnitude of data with bars, each of which represents a category of data (midpoint). The height of the bars represents the value of the bar statistic for the corresponding midpoint.

Figure 28.1 on page 740 shows the relationship between petal width and petal length for three species of flowers. The horizontal axis is the midpoint axis and the vertical axes are response axes. Each axis is labeled with the variable name. Each species is a midpoint, so each bar is labeled with the species identifier.

Figure 28.1 Bar Line Graph



About Interpolation Methods

You can produce plot overlays such as needle plot overlays by specifying interpolation methods with the SYMBOL statement. For PROC GBARLINE, you can use the SYMBOL statement to

- connect the data points to the zero line on the vertical axis (NEEDLE)
- use a step function to connect the data points (STEP)
- produce overlay plots with unconnected data points (NONE)
- connect data points with straight lines (JOIN).

For bar line graphs, points on the plot overlay are automatically joined, which is equivalent to specifying the JOIN interpolation method.

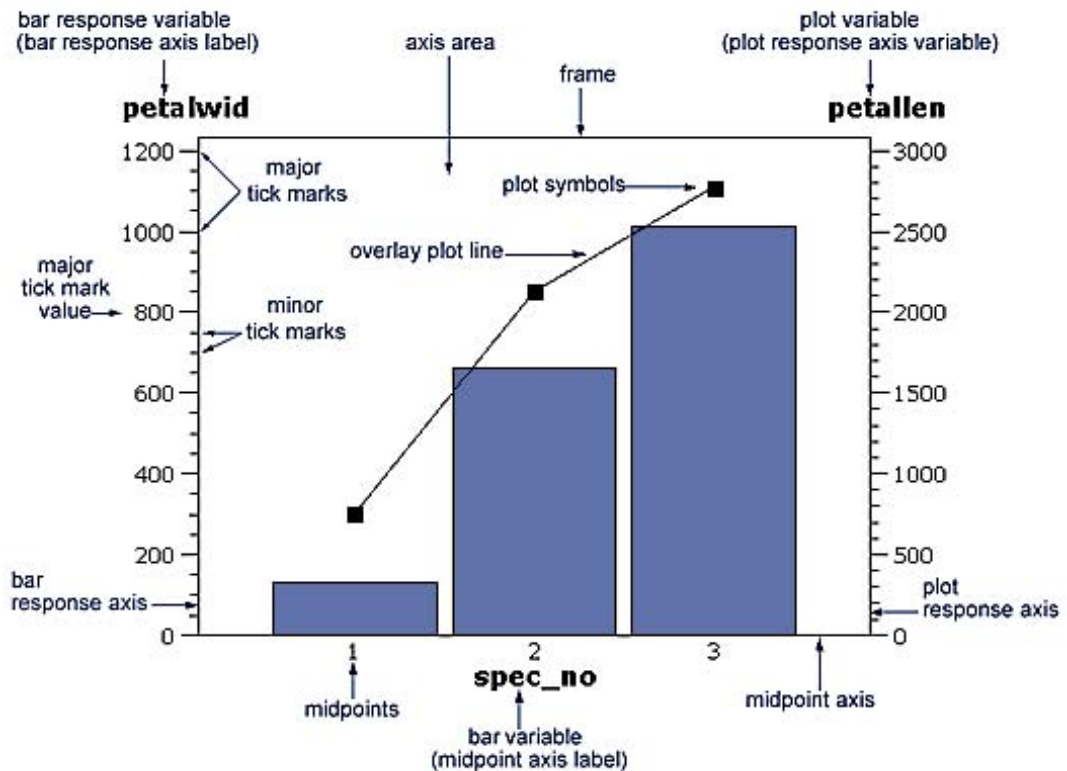
“SYMBOL Statement” on page 183 describes the JOIN, NEEDLE, STEP, and NONE interpolation methods.

Concepts

The GBARLINE procedure produces bar charts based on the values of a *bar variable* with plot overlays based on the values of a *plot variable*. The values of the bar variable are represented by a set of *midpoints*. The graph itself displays information about the bar variable in the form of *bar statistics*.

Figure 28.2 on page 741 illustrates the parts of a bar line graph.

Figure 28.2 Parts of a Bar Line Graph



Bar line graphs have three axes:

- a midpoint axis that shows the categories of data, based on the bar variable
- a left response axis that displays the scale of values for the bar statistic (based on the summary variable, if specified)

- a right response axis that displays the scale of values for the plot statistic.

The response axes are divided into evenly spaced intervals identified with major tick marks that are labeled with the corresponding statistic value. Minor tick marks are evenly distributed between the major tick marks. Each axis is labeled with the variable name or label.

About the Bar Variable

The *bar variable* is the variable in the input data set whose value determines the categories of data represented by the bar. The bar variable generates the midpoints to which each observation in the data set contribute.

The bar variable can be either character or numeric. Character bar variables contain character values, which are always discrete. Numeric bar variables fall into two categories: discrete and continuous.

- *Discrete variables* contain a finite number of specific numeric values that are to be represented on the chart. For example, a variable that contains years, such as 1984 or 2002, is a discrete variable.
- *Continuous variables* contain a range of numeric values that are to be represented on the chart. For example, a variable of temperature data that contains real values between 0 and 212 is a continuous variable.

Numeric bar variables are always treated as continuous variables unless the DISCRETE option is used in the BAR statement.

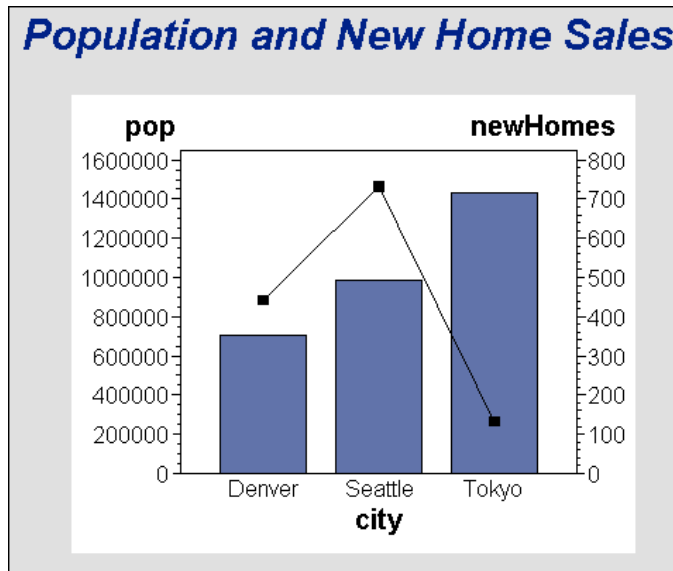
About Midpoints

Midpoints are the values of the bar variable that identify categories of data. By default, midpoints are selected or calculated by the procedure. The way the procedure handles the midpoints depends on whether the values of the bar variable are character, discrete numeric, or continuous numeric.

Character Values

A character bar variable generates a midpoint for each unique value of the variable. In the following example, the bar variable CITY contains the names of three different cities, and each city is a midpoint, resulting in three midpoints for the chart:

Figure 28.3 Character Midpoints

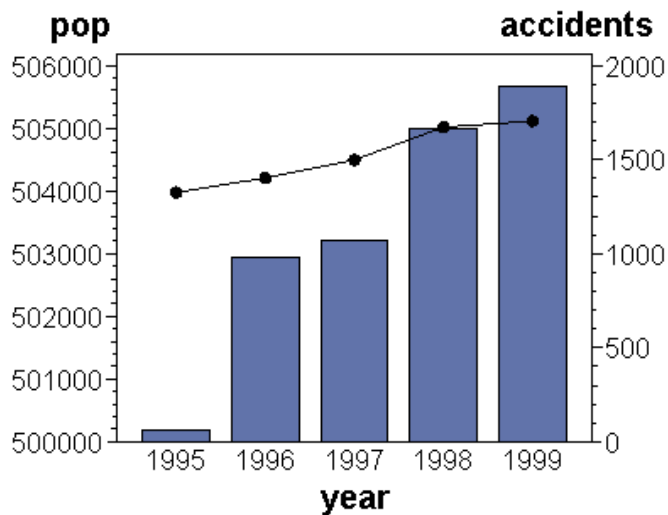


By default, character midpoints are arranged in alphabetic order. If a character variable has an associated format, then the values are arranged in order of the formatted values.

Discrete Numeric Values

A numeric bar variable used with the DISCRETE option generates a midpoint for each unique value of the bar variable. In the following example, the numeric variable YEAR used with the DISCRETE option produces one midpoint for each year:

Figure 28.4 Discrete Numeric Midpoints



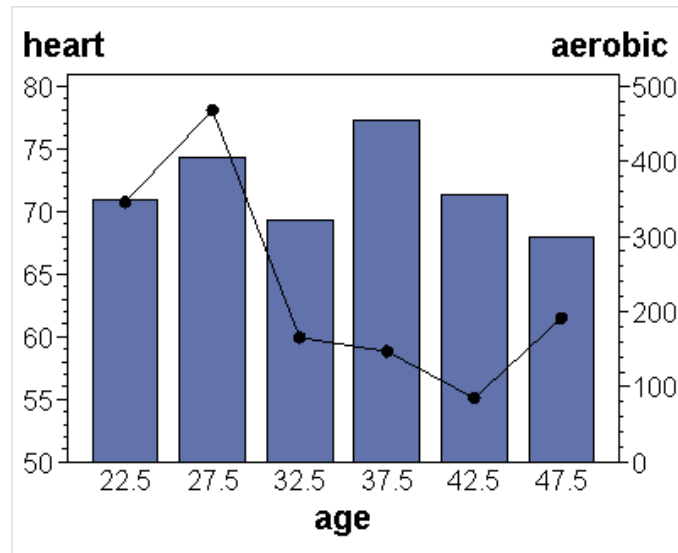
By default, numeric midpoints are arranged in ascending order. If the numeric variable has an associated format, then each formatted value generates a separate midpoint. Formatted numeric variables are arranged in ascending order according to their unformatted numeric values.

Continuous Numeric Values

A continuous numeric variable generates midpoints that represent ranges of values. By default, the GBARLINE procedure determines the ranges, calculates the median value of each range, and displays the appropriate median value at each midpoint on the chart. A value that falls exactly halfway between two midpoints is placed in the higher range.

In the following example, the numeric variable AGE produces five midpoints, each of which represents a six-year age range; the median value of the range is displayed at each midpoint:

Figure 28.5 Continuous Numeric Midpoints



By default, midpoints of ranges are arranged in ascending order.

Selecting and Ordering Midpoints

For character or discrete numeric values, you can use the MIDPOINTS= option to rearrange the midpoints or to exclude midpoints from the chart. For example, to change the default alphabetic order of the midpoints in Figure 28.3 on page 743, specify

```
midpoints='Tokyo' 'Denver' 'Seattle'
```

To exclude the midpoint for Denver, specify

```
midpoints='Tokyo' 'Seattle'
```

In this case, values excluded by the option are not included in the calculation of the bar statistic.

You can order or select discrete numeric midpoint values just as you do character values, but you omit the quotation marks when specifying numeric values.

For continuous numeric variables, use the LEVELS= or MIDPOINTS= option to change the number of midpoints, to control the range of values each midpoint

represents, or to change the order of the midpoints. To control the range of values each midpoint represents, use the MIDPOINTS= option to specify the median value of each range. For example, to select the ranges 20–29, 30–39, and 40–49, specify

```
midpoints=25 35 45
```

Alternatively, to select the number of midpoints that you want and let the procedure calculate the ranges and medians, use the LEVELS= option.

You can also use formats to control the ranges of continuous numeric variables, but in that case the values are no longer continuous but become discrete.

Note: You cannot use the MIDPOINTS= option to exclude continuous numeric values from the chart because values below or above the ranges specified by the option are automatically included in the first and last midpoints, respectively. To exclude continuous numeric values from a chart, use a WHERE statement in a DATA step or the WHERE= DATA set option. △

See also the description of the LEVELS= and MIDPOINTS= options.

About the Plot Variable

The *plot variable* is the variable in the input data set whose values are used to generate the overlay plot line. The plot variable is optional, but if specified, it must be a numeric variable.

To specify a plot variable, use the SUMVAR= option on the PLOT statement:

```
PLOT / SUMVAR=height;
```

When you specify a plot variable with the SUMVAR= option, the only statistics available for the plot are the sum or the mean. You can specify the statistic with the TYPE= option. SUM (TYPE=SUM) is the default.

If you do not specify a plot variable, then the bar variable is used as the plot variable. The only statistics available for the plot are percentage, cumulative percentage, frequency, or cumulative frequency. The default statistic is frequency (TYPE=FREQ).

For more information about these statistics, see “About Chart Statistics” on page 745. See also the descriptions of the SUMVAR= and TYPE= options for the PLOT statement.

About Chart Statistics

The *chart statistics* are the statistical values calculated for the bar variables and the plot variables. The GBARLINE procedure calculates six chart statistics. You can specify the chart statistics with the TYPE= option. For the bar, the default statistic is frequency. For the plot, the default statistic is sum.

The examples given in the descriptions of these statistics assume a data set with two variables, CITY and SALES. The values of CITY are **Denver**, **Seattle**, and **Tokyo**. There are 21 observations: seven for Denver, nine for Seattle, and five for Tokyo.

Frequency

The frequency statistic is the total number of observations in the data set for each midpoint. For example, seven observations of the bar variable, CITY, contain the value **Denver**, so the frequency for the **Denver** midpoint is 7.

Cumulative Frequency

The cumulative frequency statistic adds the frequency for the current midpoint to the frequency of all of the preceding midpoints. For example, the frequency for the **Denver**

midpoint is 7, and the frequency for the next midpoint, **Seattle**, is 9, so the cumulative frequency for **Seattle** is 16.

Percentage

The percentage statistic is calculated by dividing the frequency for each midpoint by the total frequency count for all midpoints in the chart or group and multiplying it by 100. For example, the frequency count for the **Denver** midpoint is 7 and the total frequency count for the chart is 21, so the percentage statistic for **Denver** is 33.3%.

Cumulative Percentage

The cumulative percentage statistic adds the percentage for the current midpoint to the percentage for all of the preceding midpoints in the chart or group. For example, the percentage for the **Denver** midpoint is 33.3, and the percentage for the next midpoint, **Seattle**, is 42.9, so the cumulative percentage for **Seattle** is 76.2.

Sum

The sum statistic is the total of the values, for each midpoint, for the variable specified by the SUMVAR= option. For example, if you specify SUMVAR=SALES and the values of the SALES variable for the seven **Denver** observations are **8734, 982, 1504, 3207, 4502, 624, and 918**, the sum statistic for the **Denver** midpoint is 20,471.

You must use the SUMVAR= option to specify the variable for which you want the sum statistic.

Mean

The mean statistic is the average of the values, for each midpoint, for the variable specified by the SUMVAR= option. For example, if TYPE=MEAN and SUMVAR=SALES, the mean statistic for the **Denver** midpoint is 2924.42.

You must use the SUMVAR= option to specify the variable for which you want the mean statistic.

Calculating Weighted Statistics

By default, each observation is counted only once in the calculation of a chart statistic. To calculate weighted statistics in which an observation can be counted more than once, use the FREQ= option. This option identifies a variable whose values are used as a multiplier for the observation in the calculation of the statistic. If the value of the FREQ= variable is missing, 0, or negative, then the observation is excluded from the calculation.

If you use the SUMVAR= option, then the SUMVAR= variable value for an observation is multiplied by the FREQ= variable value for the observation for use in calculating the chart statistic.

For example, to use a variable called COUNT to produce weighted statistics, assign FREQ=COUNT. If you also assign the variable HEIGHT to the SUMVAR= option, then the following table shows how the values of COUNT and HEIGHT would affect the statistic calculation:

Value of COUNT	Value of HEIGHT	Number of times the observation is used	Value used for HEIGHT
1	55	1	55
5	65	5	325
.	63	0	-
-3	60	0	-

By default, the percentage and cumulative percentage statistics are calculated based on the frequency. If you want to graph a percentage or cumulative percentage based on a sum, then you can use the FREQ= option to specify a variable to use for the "sum" calculation and specify the PCT statistic, as shown in this example:

```
freq=count type=pct
```

Because the variable that is specified by the FREQ= option determines the number of times an observation is counted, the value of COUNT is the equivalent of the sum statistic.

See also the descriptions of the TYPE=, SUMVAR=, and FREQ= options.

Note: The FREQ= option is not supported by ActiveX or Java. Δ

Missing Values

By default, the GBARLINE procedure ignores missing midpoint values for the bar variable. If you specify the MISSING option, then missing values are treated as a valid midpoint and are included on the chart.

When the value of the variable that is specified in the FREQ= option is missing, 0, or negative, the observation is excluded from the calculation of the chart statistic.

When the value of the variable specified in the SUMVAR= option is missing, the observation is excluded from the calculation of the chart statistic.

If the value of the plot variable is missing, then the GBARLINE procedure does not include the observation in the plot overlay. If you specify interpolation with a SYMBOL definition, then the plot is not broken at the missing value.

Plot Variable Values Out of Range

Exclude data values from a plot overlay by restricting the range of axis values with the RAXIS= options or with the ORDER= option in an AXIS statement. When an observation contains a value outside of the specified axis range, the GBARLINE procedure excludes the observation from the plot and issues a message to the log.

If you specify interpolation with a SYMBOL definition, then by default values outside of the axis range are excluded from interpolation calculations and, as a result, can change interpolated values for the plot overlay.

To specify that values out of range are included in the interpolation calculations, use the MODE= option in a SYMBOL statement. When MODE=INCLUDE, values that fall outside of the axis range are included in interpolation calculations but excluded from the plot. The default (MODE=EXCLUDE) omits observations that are outside of the axis range from interpolation calculations. See the MODE= option of in "SYMBOL Statement" on page 183 for details.

About Patterns

When a chart needs one or more patterns, the procedure uses either default patterns and outlines that are automatically generated by SAS/GRAPH, or patterns, colors, outlines, and images that are defined by PATTERN statements, graphics options, and procedure options.

The following sections summarize pattern behavior for the GBARLINE procedure. For more information, see “PATTERN Statement” on page 169.

Default Patterns and Outlines

In general, the default pattern that the GBARLINE procedure uses is a solid fill that it rotates once through the colors list, skipping the color that is being used as the foreground color. The procedure also outlines all areas in the foreground color. (Typically, the foreground color is the first color in the device’s colors list.)

Specifically, the GBARLINE procedure uses default patterns and outlines when you do not specify any of the following:

- *any* PATTERN statements
- the COLORS= graphics options (that is, you use the device’s default colors list and it has more than one color)
- the COUTLINE= option in the BAR statement.

If you do not specify any of these statements or options, then the GBARLINE procedure

- selects the first default fill pattern, which is always solid, and rotates it through the colors list, generating one solid pattern for each color. If the first color in the device’s colors list is black (or white), then the procedure skips that color and begins generating patterns with the next color.
- uses the foreground color to outline every patterned area.

If the procedure needs additional patterns, PROC GBARLINE selects the next default pattern fill (empty) and rotates it through the colors list, skipping the foreground color as before. The procedure continues in this fashion until it has generated enough patterns for the chart.

Changing any of these conditions may change or override the default behavior:

- If you specify a colors list with the COLORS= option in a GOPTIONS statement and the list contains more than one color, then the procedure rotates the default solid pattern through that list, using every color, even if the foreground color is black (or white). The default outline color remains the foreground color.
- Whenever there are PATTERN definitions in effect, whether or not the GBARLINE procedure can use them, the default outline color for all patterns changes from foreground to SAME, as described in “User-Defined Patterns, Outlines, and Images” on page 748.

For a description of these graphics options, see Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261.

User-Defined Patterns, Outlines, and Images

You can use PATTERN statements to explicitly specify patterns, including color or fill type or both. You can also specify images to fill the bars. For complete information on all patterns, see “PATTERN Statement” on page 169. See also the section on controlling patterns and colors for each chart type.

When you use PATTERN statements, the procedure uses the specified patterns until all of the PATTERN definitions they generate have been used. Then, if more patterns are required, the procedure returns to the default pattern rotation.

Whenever you specify *any* PATTERN statement, the default pattern outline changes. Instead of the foreground color, the outline color is the same as the fill color; for example, a blue bar has a blue outline. The effect is the same as specifying COUTLINE=SAME. Even when the procedure runs out of user-defined patterns and generates default patterns, the outlines continue to match the interior pattern color.

To change the outline color of any pattern, whether it's a default or user-defined pattern, use the COUTLINE= option in the BAR statement that generates the chart.

You can use the PATTERN statement to fill specified bars with specified images. For details, see “Placing Images on the Bars of Two-Dimensional Bar Charts” on page 116.

You can also add background images. The IBACK= option (see “IBACK” on page 317) specifies image files that fill the background area. For further information, including a listing of recognized image file types, see “Image File Types Supported by SAS/GRAPH” on page 106 and “Placing a Background Image” on page 113.

Version 6 Patterns

If you specify the V6COMP graphics option, then the procedure generates patterns by rotating the appropriate Version 6 default patterns through all of the colors in the colors list. With V6COMP, all patterns are outlined in the same color as the fill.

Note: The V6COMP graphics option is not supported by ActiveX for graphs generated by the GBARLINE procedure. Δ

Procedure Syntax

Requirements:

One BAR statement

Global statements: AXIS, FOOTNOTE, GOPTIONS, PATTERN, TITLE

Reminder: The procedure can include the BY, FORMAT, LABEL, and WHERE statements also.

Supports:

RUN-group processing

Output Delivery System (ODS)

Not supported by: Java

```
PROC GBARLINE <DATA=input-data-set>
  <ANNOTATE=Annotate-data-set>
  <IMAGEMAP=output-data-set>;
```

```
BAR bar-variable </option(s)>;
```

```
<PLOT </option(s)>;>
```

PROC GBARLINE Statement

Identifies the data set containing the chart variables. Optionally specifies annotation.

Requirements: An input data set is required.

Not supported by: Java

Syntax

```
PROC GBARLINE <DATA=input-data-set>
  <ANNOTATE=Annotate-data-set>
  <IMAGEMAP=output-data-set>;
```

Options

PROC GBARLINE statement options affect all graphs produced by the procedure.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate all graphs that are produced by the GBARLINE procedure. To annotate individual graphs, use the ANNOTATE= option in the BAR statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

DATA=*input-data-set*

specifies the SAS data set that contains the variable(s) to chart. By default, the procedure uses the most recently created SAS data set.

See also: “SAS Data Sets” on page 29 and “About the Bar Variable” on page 742

IMAGEMAP=*output-data-set*

creates a temporary SAS data set that is used to generate an image map in an HTML output file. The information in the image map data set includes the shape and coordinates of the elements in the graph and drill-down URLs that have been associated with those elements. The drill-down URLs are provided by one or two variables in the input data set. These variables are identified to the GBARLINE procedure with the HTML= option.

The %IMAGEMAP macro generates the image map in the HTML output file. The macro takes two arguments, the name of the image map data set and the name or fileref of the HTML output file, as shown in the following example:

```
%imagemap(imgmapds, myimgmap.html);
```

BAR Statement

Creates vertical bar charts in which the height of the bars represents the value of the bar statistic for each category of data.

Requirements: One bar variable is required.

Global statements: AXIS, FOOTNOTE, PATTERN, TITLE

Supports: Drill-down functionality

Not supported by: Java

Description

The BAR statement specifies the variable that defines the categories of data to chart. These statements automatically

- determine the midpoints
- calculate the bar statistic for each midpoint (the default is FREQ)
- scale the response axis and the bars according to the statistic value
- determine bar width and spacing
- assign patterns to the bars; the default bar pattern is SOLID
- draw a frame around the axis area using the first color in the colors list.

You can use statement options to select or order the midpoints (bars), to control the tick marks on the response axis, to change the type of bar statistic, to display specific statistics, and to modify the appearance of the chart. You can also specify additional variables by which to sum the data.

In addition, you can

- use global statements to modify the axes and the bar patterns. See Chapter 7, “SAS/GRAPH Statements,” on page 121 for more information.
- add titles and footnotes to the chart. See “TITLE, FOOTNOTE, and NOTE Statements” on page 210 for more information.
- use an Annotate data set to enhance the chart. See Chapter 24, “Using Annotate Data Sets,” on page 587 for more information.
- display an image in the background of the chart. See “IBACK” on page 317 for more information.
- display images in the bars of the chart. See the IMAGE= option on page 171 for the PATTERN statement.

Syntax

BAR *bar-variable* <*option(s)*>;

option(s) can be one or more options from any or all of the following categories:

- appearance options
 - ANNOTATE=*Annotate-data-set*
 - CAUTOREF=*reference-line-color*
 - CAXIS=*axis-color*
 - CERROR=*error-bar-color*
 - CFRAME=*background-color*
 - COUTLINE=*bar-outline-color* | SAME

CREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*

CTEXT=*text-color*

FRAME | NOFRAME

FRONTREF

LAUTOREF=*reference-line-type*

LREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*

PATTERNID=BY | MIDPOINT

SPACE=*bar-spacing*

WIDTH=*bar-width*

WOUTLINE=*bar-outline-width*

□ **statistic options**

CFREQ

CLM=*confidence-level*

CPERCENT

ERRORBARS=BARS | BOTH | TOP

FREQ

FREQ=*numeric-variable*

INSIDE=*statistic*

MEAN

OUTSIDE=*statistic*

PERCENT

SUM

SUMVAR=*summary-variable*

TYPE=*statistic*

□ **midpoint options**

DISCRETE

LEVELS=*number-of-midpoints*

MIDPOINTS=*value-list*

MIDPOINTS=OLD

MISSING

□ **axes options**

ASCENDING

AUTOREF

AXIS=AXIS<1...99>

CLIPREF

DESCENDING

MAXIS=AXIS<1...99>

MINOR=*number-of-minor-ticks*

NOAXIS

NOBASEREF

NOZERO

RANGE

RAXIS=*value-list* | AXIS<1...99>

REF=*value-list*

□ **catalog entry description options**

DESCRIPTION=*'entry-description'*

NAME=*'string'*

□ ODS options

HTML=*variable*

Required Arguments

bar-variable

specifies the variable that defines the categories of data to chart. The variable must be in the input data set.

See also: “About the Bar Variable” on page 742

Options

Options in the BAR statement affect all graphs that are produced by that statement. You can specify as many options as you want and list them in any order. For details on specifying colors, see Chapter 6, “SAS/GRAPH Colors and Images,” on page 91. For details on specifying images, see “Specifying Images in SAS/GRAPH Programs” on page 106. For a complete description of the graphics options, see Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate charts produced by the BAR statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

ASCENDING

arranges the bars in ascending order of the value of the bar statistic. By default, bars are arranged in ascending order of midpoint value, without regard to the lengths of the bars. ASCENDING reorders the bars from shortest to longest. The ordering is left to right.

ASCENDING overrides any midpoint order specified in the MIDPOINTS= option or specified in the ORDER= option in an AXIS statement assigned to the midpoint axis.

AUTOREF

draws a reference line at each major tick mark on the chart (left) response axis. To draw reference lines at specific points on the response axis, use the REF= option.

By default, reference lines are drawn in front of the bars. To draw reference lines behind the bars, use the CLIPREF option.

AXIS=AXIS<1...99>

See RAXIS= on page 761.

CAUTOREF=*reference-line-color*

specifies the color of reference lines drawn at major tick marks, as determined by the AUTOREF option. The default color is either the value of the CAXIS= option or the first color in the color list. To specify a line type for these reference lines, use the LAUTOREF= option.

CAXIS=*axis-color*

specifies a color for the response and midpoint axis lines and for the default axis area frame. If you omit the CAXIS= option, PROC GBARLINE searches for a color specification in this order:

- 1 the COLOR= option in AXIS definitions

2 the first color in the colors list (the default).

CERROR=error-bar-color

specifies the color of error bars. The default is the color of the response axis, which is controlled by the CAXIS= option.

CFRAME=background-color

CFR=background-color

specifies the color with which to fill the axis area.

The axis area color does not affect the frame color, which is always the same as the midpoint axis line color and controlled by the CAXIS= option. By default, the axis area is not filled.

The CFRAME= option is overridden by the NOFRAME option.

Note: If the background color, the bar color, and the outline color are the same, then you will not be able to distinguish the bars. △

CFREQ

displays the cumulative frequency statistic above the bars. A maximum of two statistics can be printed. This option is ignored if the bars are too narrow to avoid overlapping values or if the FREQ option is specified.

See also: “About Chart Statistics” on page 745 and “Displaying Statistics In Bar Line Charts” on page 763

CLIPREF

clips the reference lines at the bars. This makes the reference lines appear to be behind the bars.

CLM=confidence-level

specifies the confidence intervals to use when drawing error bars. Values for *confidence-level* must be greater than or equal to 50 and strictly less than 100. The default is 95. See ERRORBAR= for details on how error bars are computed and drawn.

COUTLINE=bar-outline-color | SAME

outlines all bars or bar segments using the specified color. SAME specifies that the outline color of a bar is the same as the interior pattern color.

The default outline color depends on the PATTERN statement:

- If you do not specify a PATTERN statement, the default outline color is black for the ActiveX device. Otherwise, the default outline color is the foreground color (the first color in the colors list).
- If you specify the PATTERN statement or the V6COMP graphics option, the default is COUTLINE=SAME.

The COUTLINE= option is not valid when SHAPE=CYLINDER.

See also: “Controlling Bar Line Chart Patterns, Colors, and Images” on page 764 and “About Patterns” on page 748

CPERCENT

CPCT

displays the cumulative percentage statistic above the bars. A maximum of two statistics can be printed. This option is ignored if the bars are too narrow to avoid overlapping values or if the FREQ, CFREQ, or PERCENT option is specified.

See also: “About Chart Statistics” on page 745 and “Displaying Statistics In Bar Line Charts” on page 763

CREF=reference-line-color | (reference-line-color) | reference-line-color-list

CR=reference-line-color | (reference-line-color) | reference-line-color-list

specifies colors for reference lines. Specifying a single color without parentheses applies that color to all reference lines, including lines drawn with the AUTOREF and REF= options. The CAUTOREF= option overrides the CREF= reference line color for reference lines drawn with the AUTOREF option. Specifying a single color in parentheses applies that color only to the first reference line drawn with the REF= option. Specifying a reference color list applies colors in sequence to successive lines drawn with the REF= option. The syntax of the color list is of the form (*color1 color2 ...colorN*) or (*color1, color2 ..., colorN*). The default color for reference lines is either the value of the CAXIS= option or the first color in the color list. To specify line types for these reference lines, use the LREF= option.

CTEXT=*text-color*

specifies the color of all text on the chart that is not otherwise assigned a color. Text includes axis values and axis labels in the response and midpoint axes; and the displayed statistics. For the ActiveX device, the default color is black. For other devices, if you omit the CTEXT= option, PROC GBARLINE searches for a color specification in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the first color in the colors list (the default).

The CTEXT= option overrides the color specification for the axis label and the tick mark values in the COLOR= option in an AXIS definition assigned to an axis.

The CTEXT= option is overridden by the COLOR= suboption of a LABEL= or VALUE= option in an AXIS definition assigned to an axis. In this case the suboption determines the color of the axis label or the color of the tick mark values, respectively.

DESCENDING

arranges the bars in descending order of the value of the bar statistic. By default, bars are arranged in ascending order of midpoint value, without regard to the lengths of the bars. DESCENDING reorders the bars from longest to shortest. The ordering is left to right.

DESCENDING overrides any midpoint order that is specified with the MIDPOINTS= option or that is specified in the ORDER= option in an AXIS statement assigned to the midpoint axis.

DESCRIPTION=*'entry-description'*

DES=*'entry-description'*

specifies the description of the catalog entry for the chart. The maximum length for the *entry-description* is 256 characters. The description does not appear on the chart. By default, the GBARLINE procedure assigns a description of the form GBARLINE CHART OF *variable*, where *variable* is the name of the bar variable.

The *entry-description* can include the #BYLINE, #BYVAL, and #BYVAR substitution options, which work as they do when used on TITLE, FOOTNOTE, and NOTE statements. Refer to the description of the options on page 222 and to “Substituting BY Line Values in a Text String” on page 226. The 256-character limit applies before the substitution takes place for these options; thus, if in the SAS program the entry-description text exceeds 256 characters, it is truncated to 256 characters, and then the substitution is performed.

The descriptive text is shown in each of the following:

- the "description" portion of the Results window
- the catalog-entry properties that you can view from the Explorer window
- the Table of Contents that is generated when you use CONTENTS= on an ODS statement (see “Linking to Output through a Table of Contents” on page 495), assuming the GBARLINE output is generated while the contents page is open
- the Description field of the PROC GREPLAY window.

DISCRETE

treats a numeric bar variable as a discrete variable rather than as a continuous variable. The GBARLINE procedure creates a separate midpoint and, hence, a separate bar for each unique value of the bar variable. If the bar variable has a format associated with it, then each formatted value is treated as a midpoint.

The LEVELS= option is ignored when you use DISCRETE. The MIDPOINTS= option overrides DISCRETE. The ORDER= option in an AXIS statement that is assigned to the midpoint axis can rearrange or exclude discrete midpoint values.

ERRORBAR=BARS | BOTH | TOP

draws confidence intervals for either of the following:

- the mean of the SUMVAR= variable for each midpoint if you specify TYPE=MEAN
- the percentage of observations assigned to each midpoint if you specify TYPE=PCT with no SUMVAR= option.

The ERRORBAR= option cannot be used with values of the TYPE= option other than MEAN or PCT. Valid values for ERRORBAR= are:

BARS

draws error bars as bars half the width of the main bars.

BOTH

draws error bars as two ticks joined by a line (default).

TOP

draws the error bar as a tick for the upper confidence limit that is joined to the top of the bar by a line.

By default, ERRORBAR= uses a confidence level of 95 percent. You can specify different confidence levels with the CLM= option.

When you use ERRORBAR= with TYPE=PCT, the confidence interval is based on a normal approximation. Let TOTAL be the total number of observations, and PCT be the percentage assigned to a given midpoint. The standard error of the percentage is approximated as

$$APSTDERR = 100 * \sqrt{(PCT/100) * (1 - (PCT/100)) / TOTAL};$$

Let LEVEL be the confidence level specified using the CLM= option, with a default value of 95. The upper confidence limit for the percentage is computed as

$$UCLP = PCT + APSTDERR * \text{PROBIT}(1 - (1 - LEVEL/100)/2);$$

The lower confidence limit for the percentage is computed as

$$LCLP = PCT - APSTDERR * \text{PROBIT}(1 - (1 - LEVEL/100)/2);$$

When you use ERRORBAR= with TYPE=MEAN, the sum variable must have at least two non-missing values for each midpoint. Let N be the number of observations assigned to a midpoint, MEAN be the mean of those observations, and STD be the standard deviation of the observations. The standard error of the mean is computed as

$$STDERR = STD / \sqrt{N};$$

Let LEVEL be the confidence level specified using the CLM= option, with a default value of 95. The upper confidence limit for the mean is computed as

$$UCLM = MEAN + STDERR * \text{TINV}(1 - (1 - LEVEL/100)/2, N-1);$$

The lower confidence limit for the mean is computed as

$$LCLM = MEAN - STDERR * \text{TINV}(1 - (1 - LEVEL/100)/2, N-1);$$

If you want the error bars to represent a given number *C* of standard errors instead of a confidence interval, and if the number of observations assigned to each midpoint is the same, then you can find the appropriate value for the CLM= option by running a DATA step. For example, if you want error bars that represent one standard error (*C*=1) with a sample size of *N*, you can run the following DATA step to compute the appropriate value for the CLM= option and assign that value to a macro variable &LEVEL:

```
data null;
  c = 1;
  n = 10;
  level = 100 * (1 - 2 * (1 - probt( c, n-1)));
  put all;
  call symput('level',put(level,best12.));
run;
```

Then when you run the GBARLINE procedure, you can specify CLM=&LEVEL.

Note that this trick does not work precisely if different midpoints have different numbers of observations. However, choosing an average value for *N* may yield sufficiently accurate results for graphical purposes if the sample sizes are large or do not vary much.

FRAME | NOFRAME

FR | NOFR

specifies whether the axis area frame is drawn. The default is FRAME, which draws a frame around the axis area. Specifying NOFRAME removes the axis area frame, including any background color or image. To remove one or more axis elements, use either the AXIS statement or the NOAXIS option.

The NOFRAME option overrides the CFRAME= option and “IBACK” on page 317.

The color of the frame or backplane outline is the color of the midpoint axis, which is determined by the CAXIS= option.

If the V6COMP graphics option is in effect, the default value for GRSEGs is NOFRAME. See “Version 6 Patterns” on page 749 for more information.

FREQ

displays the frequency statistic above the bars. Non-integer values are rounded down to the nearest integer. A maximum of two statistics can be printed. This option is ignored if the bars are too narrow to avoid overlapping values. This option overrides the CFREQ, PERCENT, CPERCENT, SUM, and MEAN options.

See also: “About Chart Statistics” on page 745 and “Displaying Statistics In Bar Line Charts” on page 763

FREQ=numeric-variable

specifies a variable whose values weight the contribution of each observation in the computation of the bar statistic. Each observation is counted the number of times that is specified by the value of *numeric-variable* for that observation. If the value of *numeric-variable* is missing, 0, or negative, then the observation is not used in the statistic calculation. Non-integer values of *numeric-variable* are truncated to integers. The FREQ= option is valid with all bar statistics.

Because you cannot use TYPE=PERCENT, TYPE=CPERCENT, TYPE=FREQ, or TYPE=CFREQ with the SUMVAR= option, you must use the FREQ= option to calculate percentages, cumulative percentages, frequencies, or cumulative frequencies based on a sum.

The statistics will not be affected by applying a format to *numeric-variable*.

Not supported by: ActiveX, Java

See also: “Calculating Weighted Statistics” on page 746

FRONTREF

specifies that reference lines drawn by the AUTOREF or REF= options should be drawn in front of the bars.

HTML=*variable*

identifies the variable in the input data set whose values create links in the HTML file created by the ODS statement. These links are associated with the bars and point to the data or graph you wish to display when the user drills down on the area. The values of *variable* can be up to 1024 characters long. Characters after the 1024-character limit (including any closing quotes) are truncated.

INSIDE=*statistic*

displays the values of the specified statistic inside the bars. *Statistic* can be one of the following:

- FREQ
- CFREQ
- PERCENT | PCT
- CPERCENT | CPCT
- SUM
- MEAN

To display statistics with INSIDE=SUM or INSIDE=MEAN, you must also specify the SUMVAR= option.

See also: “About Chart Statistics” on page 745 and “Displaying Statistics In Bar Line Charts” on page 763

LAUTOREF=*reference-line-type*

specifies a line type for reference lines drawn at major tick marks, as specified by the AUTOREF option. The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. The default value 1 draws a solid line. To specify colors for these reference lines, use the CAUTOREF= option.

LEVELS=*number-of-midpoints*

specifies the number of midpoints for a numeric bar variable. The range for each midpoint is calculated automatically, using the algorithm in Terrell and Scott (1985). The LEVELS= option is ignored if

- the bar variable is character type
- the DISCRETE option is used
- the MIDPOINTS= option is used.

Featured in: Example 1 on page 768

LREF=*reference-line-type* | (*reference-line-type* | *reference-line-type-list*)**LR=*reference-line-type* | (*reference-line-type* | *reference-line-type-list*)**

specifies line types for reference lines. Line types are specified as whole numbers from 1 to 46, with 1 representing a solid line and the other values representing dashed lines. Specifying a line type without parentheses applies that type to all reference lines drawn with the AUTOREF and REF= options. Note that the LAUTOREF= option overrides LREF=*reference-line-type* for reference lines drawn with the AUTOREF option. Specifying a single line type in parentheses applies that line type to the first reference line drawn with the REF= option. Specifying a line type list applies line types in sequence to successive reference lines drawn with the REF= option. The syntax of the line-type list is of the form (*type1 type2 ...typeN*). The default line type is specified by the AXIS statement's STYLE= option. By default, STYLE=1, a solid line. To specify colors for these reference lines, use the CREF= option.

MAXIS=AXIS<1...99>

assigns the specified AXIS definition to the midpoint axis. The MAXIS= option is ignored if the specified AXIS definition does not exist.

See also: “AXIS Statement” on page 124 and “About Midpoints” on page 742

MEAN

displays the mean statistic above the bars. A maximum of two statistics can be printed. This option is ignored if the bars are too narrow to avoid overlapping values or if the FREQ, CFREQ, PERCENT, CPERCENT, or SUM option is specified. MEAN is ignored unless you also use the SUMVAR= option.

See also: “About Chart Statistics” on page 745 and “Displaying Statistics In Bar Line Charts” on page 763

MIDPOINTS=*value-list*

specifies the midpoint values for the bars. The way you specify *value-list* depends on the type of the bar variable.

- For numeric bar variables, *value-list* is either an explicit list of values, or a starting and an ending value with an interval increment, or a combination of both forms:

n <...*n*>

n TO *n* <BY *increment*>

n<...*n*> TO *n* <BY *increment*> <*n* <...*n*>>

If a numeric variable has an associated format, the specified values must be the *unformatted* values.

By default, numeric variable values are treated as continuous (if you omit the DISCRETE option), and

- the lowest midpoint consolidates all data points from negative infinity to the median of the first two midpoints
- the highest midpoint consolidates all data points from the median of the last two midpoints up to infinity
- all other values in *value-list* specify the median of a range of values, and the GBARLINE procedure calculates the midpoint values.

If you include the DISCRETE option, then each value in *value-list* specifies a unique numeric value.

- For character bar variables, *value-list* is a list of unique character values enclosed in quotation marks and separated by blanks:

'*value-1*' <...'value-*n*'>

If a character variable has an associated format, the specified values must be the *formatted* values.

For a complete description of *value-list*, see ORDER= on page 130.

If the *value-list* for either type of variable specifies so many midpoints that the axis values overwrite each other, then the values may be unreadable. In this case the procedure writes a warning to the SAS log. On many devices, this problem can be corrected by either adjusting the size of the text with the HTEXT= graphics option or by increasing the number of cells in your graphics display using the HPOS= and VPOS= graphics options.

The ORDER= option in the AXIS statement overrides the order specified in the MIDPOINTS= option. The BAR statement options ASCENDING and DESCENDING also override both the MIDPOINTS= and ORDER= options in the AXIS statement.

See also: “About Midpoints” on page 742

MIDPOINTS=OLD

generates default midpoints using the Nelder algorithm (*Applied Statistics* 25:94–7, 1976). The MIDPOINTS=OLD option is ignored unless the bar variable is numeric.

MINOR=*number-of-minor-ticks*

specifies the number of minor tick marks between each major tick mark on the bar response axis.

The MINOR= option in a bar chart statement overrides the number of minor tick marks specified in the MINOR= option in an AXIS definition assigned to the response axis with the RAXIS= option.

MISSING

accepts a missing value as a valid midpoint for the bar variable. By default, observations with missing values are ignored.

NAME=*'string'*

specifies the name of the catalog entry for the graph. The maximum length for entry-name is eight characters. The default name is GBARLIN. If the name duplicates an existing entry name, then SAS/GRAPH software uses a number to generate a unique name—for example, GBARLIN1.

NOAXIS

suppresses all axes, including axis lines, axis labels, axis values, and all major and minor tick marks. If you specify an axis definition with the MAXIS= or RAXIS= options, then the axes are generated as defined in the AXIS statement, but then all lines, labels, values, and tick marks are suppressed. Therefore, axis statement options such as ORDER=, LENGTH=, and OFFSET= will still be used.

To remove only selected axis elements such as lines, values, or labels, use specific AXIS statement options.

NOAXIS does not suppress either the default frame or an axis area fill requested by the CFRAME= option. To remove the axis frame, use the NOFRAME option in the procedure.

NOBASEREF

suppresses the zero reference line when the SUM or MEAN bar statistic has negative values.

NOZERO

suppresses any midpoints for which there are no corresponding values of the bar variable and, hence, no bar.

Note: If a bar is omitted and if you have also specified bar labels with the VALUE= option in an AXIS statement, then the labels can be shifted and not displayed with the correct bar. \triangle

OUTSIDE=*statistic*

displays the values of the specified statistic above the bars. *Statistic* can be one of the following:

- FREQ
- CFREQ
- PERCENT | PCT
- CPERCENT | CPCT
- SUM
- MEAN

To display statistics with OUTSIDE=SUM or OUTSIDE=MEAN, you must also specify the SUMVAR= option.

See also: “About Chart Statistics” on page 745 and “Displaying Statistics In Bar Line Charts” on page 763

PATTERNID=BY | MIDPOINT

specifies the way fill patterns are assigned. By default, PATTERNID=MIDPOINT. Values for PATTERNID= are as follows:

BY

changes patterns each time the value of the BY variable changes. All bars use the same pattern if the GBARLINE procedure does not include a BY statement.

MIDPOINT

changes patterns every time the midpoint value changes.

See also: “Controlling Bar Line Chart Patterns, Colors, and Images” on page 764

PERCENT

PCT

displays the percentages of observations having a given value for the bar variable above the bars. A maximum of two statistics can be printed. This option is ignored if the bars are too narrow to avoid overlapping values or if the FREQ or CFREQ option is specified.

See also: “About Chart Statistics” on page 745 and “Displaying Statistics In Bar Line Charts” on page 763

RANGE

displays on the axis of the chart the range of numeric values represented by each bar. In the graphics output, the less-than symbol (<) and the less-than-or-equal-to symbol (<=) are used to accurately specify the starting and ending values of each range. The RANGE option has no affect on axes that represent character data. By default, the values shown on the axis are determined by the value of the MIDPOINTS= option on page 759. If specified, the DISCRETE option on page 756 overrides the RANGE option.

RAXIS=value-list | AXIS<1...99>

AXIS=value-list | AXIS<1...99>

specifies values for the major tick mark divisions on the response axis or assigns the specified AXIS definition to the axis. See the MIDPOINTS= option on page 759 for a description of *value-list*. By default, the GBARLINE procedure scales the response axis automatically and provides an appropriate number of tick marks.

You can specify negative values, but negative values are reasonable only when TYPE=SUM or TYPE=MEAN and one or more of the sums or means are less than 0. Frequency and percentage values are never less than 0.

For lists of values, a separate major tick mark is created for each individual value. A warning message is written to the SAS log if the values are not evenly spaced.

If the values represented by the bars are larger than the highest tick mark value, then the bars are truncated at the highest tick mark.

See also: “AXIS Statement” on page 124

REF=value-list

draws reference lines at the specified points on the bar response axis. See the MIDPOINTS= option on page 759 for a description of *value-list*.

Values can be listed in any order, but should be within the range of values represented by the response axis. A warning is written to the SAS log if any of the points are off of the axis, and no reference line is drawn for such points. You can use the AUTOREF option to draw reference lines automatically at all of the major tick marks.

SPACE=bar-spacing

specifies the amount of space between individual bars. *Bar-spacing* can be any non-negative number, including decimal values. Units are character cells. By default, the GBARLINE procedure calculates spacing based on the size of the axis area and the number of bars on the chart. Use SPACE=0 to leave no space between adjacent bars.

The SPACE= option is ignored if the specified spacing requests a chart that is too large to fit in the space available for the midpoint axis, and a warning message is issued.

SUM

displays the sum statistic above the bars. A maximum of two statistics can be printed. This option is ignored if the bars are too narrow to avoid overlapping values or if the FREQ, CFREQ, PERCENT, or CPERCENT option is specified. SUM is ignored unless you also use the SUMVAR= option.

See also: “About Chart Statistics” on page 745 and “Displaying Statistics In Bar Line Charts” on page 763

SUMVAR=summary-variable

specifies a numeric variable for sum or mean calculations. The GBARLINE procedure calculates the sum or, if requested, the mean of *summary-variable* for each midpoint. The resulting statistics are represented by the length of the bars along the response axis, and they are displayed at major tick marks.

When you use the SUMVAR= option, the TYPE= option must be either SUM or MEAN. With the SUMVAR= option, the default is TYPE=SUM.

Featured in: Example 1 on page 768

TYPE=statistic

specifies the bar statistic.

- If the SUMVAR= option is not used, *statistic* can be one of the following:

FREQ
frequency (the default)

CFREQ
cumulative frequency

PERCENT PCT
percentage

CPERCENT CPCT
cumulative percentage

- If the SUMVAR= option is used, *statistic* can be:

SUM
sum (the default)

MEAN
mean

Because you cannot use TYPE=FREQ, TYPE=CFREQ, TYPE=PERCENT, or TYPE=CPERCENT with the SUMVAR= option, you must use the FREQ= option to calculate percentages, cumulative percentages, frequencies, or cumulative frequencies based on a sum. See also “Calculating Weighted Statistics” on page 746.

See also: “About Chart Statistics” on page 745 for a complete description of statistic types

WIDTH=bar-width

specifies the width of the bars. By default, the GBARLINE procedure selects a bar width that accommodates the midpoint values displayed on the midpoint axis using a

hardware font and a height of one cell. Units for *bar-width* are character cells. The value for *bar-width* must be greater than 0, but it does not have to be an integer, for example,

```
bar site / width=1.5;
```

If the requested bar width results in a chart that is too large to fit in the space available for the midpoint axis, then the procedure issues a warning in the log and ignores the WIDTH= specification. If the specified width is too narrow, the procedure may display the midpoint values vertically.

WOUTLINE=*bar-outline-width*

specifies the width of the bar outline in pixels.

The Bar Statistic and the Response Axis

In bar line charts, the scale of values of the bar statistic is displayed on the left response axis. By default, the response axis is divided into evenly spaced intervals identified with major tick marks that are labeled with the corresponding statistic value. Minor tick marks are evenly distributed between the major tick marks unless a log axis has been requested. For sum and mean statistics, the major tick marks are labeled with values of the SUMVAR= variable (formatted if the variable has an associated format). The response axis is also labeled with the statistic type.

Displaying Statistics In Bar Line Charts

Statistic values on bar line charts are not printed by default, so you must explicitly request a statistic with the **FREQ**, **CFREQ**, **PERCENT**, **CPERCENT**, **SUM**, **MEAN**, **INSIDE=**, or **OUTSIDE=** option.

For graphs generated with the ActiveX device, you can display one statistic for each bar. For graphs generated with other devices, you can display up to two statistics for each bar. Statistics can be displayed either above the bars or inside the bars.

To specify a statistic that you want to display above the bars, specify the statistic option (**FREQ**, **CFREQ**, **PERCENT**, **CPERCENT**, **SUM**, or **MEAN**) or specify **OUTSIDE=statistic**. To specify a statistic that you want to display inside the bars, specify **INSIDE=statistic**.

For graphs generated with the ActiveX device, the **OUTSIDE=** option overrides **INSIDE=**, and **INSIDE=** overrides the **FREQ**, **CFREQ**, **PERCENT**, **CPERCENT**, **SUM**, and **MEAN** options. For graphs generated with other devices, the individual statistic options override the **OUTSIDE=** option.

If more than one statistic option is specified, only the highest priority statistic is displayed. The priority order, from highest to lowest, is as follows:

- 1 FREQ
- 2 CFREQ
- 3 PERCENT
- 4 CPERCENT
- 5 SUM
- 6 MEAN

The bars must be wide enough to accommodate the text. You can adjust the width of the bars with the **WIDTH=** option. To control the font and size of the text, use the **HTEXT=** and **FTEXT=** graphics options.

Ordering and Selecting Midpoints

To rearrange character or discrete numeric midpoint values or to select ranges for numeric values, use the **MIDPOINTS=** option. Remember that although changing the

number of midpoints for numeric variables may change the range of values for individual midpoints, it does not change the range of values for the chart as a whole. For details, see “About Midpoints” on page 742.

Like the MIDPOINTS= option, the ORDER= option in the AXIS statement can rearrange the order of the midpoints or suppress the display of discrete numeric or character values. However, the ORDER= option cannot calculate the midpoints for a continuous numeric variable, or exclude values from the calculations. For details, see the description of the ORDER= option on page 130.

Controlling Bar Line Chart Patterns, Colors, and Images

Default Patterns and Outlines

Each bar in a bar line chart is filled with a pattern. By default, the procedure

- fills the bars with bar patterns, beginning with the default fill, SOLID, and rotating it through the colors list. When the solid patterns are exhausted, the procedure selects the next default bar pattern (empty) and rotates it through the colors list. It continues in this fashion until all of the required patterns have been assigned.

If you use the device’s default colors and the first color in the list is either black or white, then the procedure does not create a pattern in that color. If you specify a colors list with the COLORS= graphics option, then the procedure uses all the colors in the list to generate the patterns.

- outlines bars using the first color in the colors list.

See “About Patterns” on page 748 for more information on how the GBARLINE procedure assigns default patterns and outlines.

User-Defined Patterns

To override the default patterns and select fills and colors for the bars, use the PATTERN statement. Only solid and empty bar patterns are valid; all other pattern fills are ignored. For a complete description of all bar patterns, see the VALUE= option on page 171.

Whenever you use PATTERN statements, the default pattern outline color changes to SAME. That is, the outline color is the same as the fill color. To specify the outline color, use the COUTLINE= option (see COUTLINE= on page 754).

When Patterns Change

The PATTERNID= option controls when the pattern changes. By default, PATTERNID=MIDPOINT, which specifies that the pattern changes every time the midpoint value changes.

Instead of changing the pattern for each midpoint, you can change the pattern for each BY group by changing the value of the PATTERNID= option. See the PATTERNID= option on page 761 for details.

Axis Color

By default, axis elements use the first color in the colors list or the colors that are specified by AXIS statement color options. However, BAR statement options can also control the color of the axis lines, text, and frame.

To change the color of...	Use this option...
the axis text	CTEXT=
the axis lines	CAXIS=
the area within the frame	CFRAME=

Adding Images to Bar Line Charts

You can apply images to the bars and to the backplane frame of bar line charts developed with the BAR statement. For details, see “Specifying Images in SAS/GRAPH Programs” on page 106.

PLOT Statement

Creates a plot overlay on top of the bar line chart.

Requirements: If specified, the PLOT statement must be specified after the BAR statement.

Global statements: AXIS, FOOTNOTE, PATTERN, SYMBOL, TITLE

Supports: Drill-down functionality

Not supported by: Java

Description

The PLOT statement specifies one plot request. This statement automatically

- scales the plot response (right) axis to include the maximum and minimum data values
- plots data points within the axis
- labels the plot response axis with the name of its variable and displays each major tick mark value.

You can use statement options to specify a plot variable, manipulate the plot response axis, modify the appearance of your graph, and describe catalog entries. You can use SYMBOL definitions to modify plot symbols for the data points, join data points, or specify other types of interpolations. For more information on the SYMBOL statement, see “SYMBOL Statement” on page 183.

In addition, you can use global statements to modify the axis, or add titles, footnotes, and notes to the plot.

Syntax

```
PLOT </options(s)>;
```

The PLOT statement is optional, but if specified, it must follow the BAR statement. If you do not specify a PLOT statement, GBARLINE generates only a bar chart and duplicates the bar response axis (left axis) as the plot response axis (right axis).

To specify a variable to plot, use the SUMVAR= option. If you do not specify a plot variable, GBARLINE uses the bar variable as the plot variable. For more information, see “About the Plot Variable” on page 745 and the description of the SUMVAR= option.

option(s) can be one or more options from any or all of the following categories:

- appearance options:
 - NOLINE
 - NOMARKER

- statistic options:
 - CFREQ
 - CPERCENT
 - FREQ
 - FREQ=*numeric-variable*
 - MEAN
 - PERCENT
 - SUM
 - SUMVAR=*plot-variable*
 - TYPE=*statistic*
- axes options:
 - ASCENDING
 - AXIS=AXIS<1...99>
 - DESCENDING
 - RAXIS=*value-list* | AXIS<1...99>
 - MINOR=*number-of-minor-ticks*
- ODS options:
 - HTML=*variable*

Options

You can specify as many options as you want and list them in any order.

ASCENDING

joins the plot points in ascending order of the value of the plot statistic. By default, data points are joined in ascending order of the midpoint value.

AXIS=AXIS<1...99>

See RAXIS= on page 767.

DESCENDING

joins the plot points in descending order of the value of the plot statistic. By default, plot points are arranged in ascending order of the midpoint value.

FREQ=*numeric-variable*

specifies a variable whose values weight the contribution of each observation in the computation of the plot statistic. Each observation is counted the number of times that is specified by the value of *numeric-variable* for that observation. If the value of *numeric-variable* is missing, 0, or negative, then the observation is not used in the statistic calculation. Non-integer values of *numeric-variable* are truncated to integers. The FREQ= option is valid with all plot statistics.

Because you cannot use TYPE=PERCENT, TYPE=CPERCENT, TYPE=FREQ, or TYPE=CFREQ with the SUMVAR= option, you must use the FREQ= option to calculate percentages, cumulative percentages, frequencies, or cumulative frequencies based on a sum.

The statistics will not be affected by applying a format to *numeric-variable*.

Not supported by: ActiveX, Java

HTML=*variable*

identifies the variable in the input data set whose values create links in the HTML file created by the ODS statement. These links are associated with the plot points. The links point to the data or graph that you wish to display when the user drills down on the plot point or area. The values of *variable* can be up to 1024 characters

long. Characters after the 1024-character limit (including any closing quotes) are truncated.

Note: This option is supported only by the ActiveX device drivers. △

MINOR=*number-of-minor-ticks*

specifies the number of minor tick marks that are drawn between each major tick mark on the plot response axis. Minor tick marks are not labeled. The MINOR= option overrides the NUMBER= suboption of the MINOR= option in an AXIS definition. You must specify a positive number.

NOLINE

suppresses the drawing of the line plot overlay, regardless of what is specified in the SYMBOL statement.

NOMARKER

suppresses the drawing of the marker symbol, regardless of what is specified in the SYMBOL statement.

RAXIS=*value-list* | AXIS<1...99>

AXIS=*value-list* | AXIS<1...99>

specifies the major tick mark values for the plot (right) response axis or assigns an AXIS definition.

The way you specify *value-list* depends on the type of variable:

- For numeric variables, *value-list* is either an explicit list of values, or a starting and an ending value with an interval increment, or a combination of both forms:

n <...*n*>

n TO *n* <BY *increment*>

n <...*n*> TO *n* <BY *increment* > <*n* <...*n*> >

If a numeric variable has an associated format, the specified values must be the *unformatted* values.

- For date-time values, *value-list* includes any SAS date, time, or datetime value described for the SAS functions INTCK and INTNX, shown here as *SAS-value*:

'*SAS-value*'*i* < ...'*SAS-value*'*i*>

'*SAS-value*'*i* TO '*SAS-value*' *i*<BY *interval*>

SUMVAR=*plot-variable*

specifies the variable to plot. *Plot-variable*, if specified, must be numeric. The GBARLINE procedure calculates the sum or, if requested, the mean of *plot-variable* for each midpoint.

When you use the SUMVAR= option, the TYPE= option must be either SUM or MEAN. With the SUMVAR= option, the default is TYPE=SUM.

Featured in: Example 1 on page 768

See also: "About the Plot Variable" on page 745

TYPE=*statistic*

specifies the plot statistic.

- If the SUMVAR= option is not used, *statistic* can be one of the following:

FREQ

frequency (the default)

CFREQ

cumulative frequency

PERCENT PCT
percentage

CPERCENT CPCT
cumulative percentage

- If SUMVAR= is used, *statistic* can be one of the following:

SUM
sum (the default)

MEAN
mean

Because you cannot use TYPE=FREQ, TYPE=CFREQ, TYPE=PERCENT, or TYPE=CPERCENT with SUMVAR=, you must use FREQ= to calculate percentages or frequencies based on a sum.

See also: “About Chart Statistics” on page 745 and “Calculating Weighted Statistics” on page 746

About SYMBOL Definitions

SYMBOL statements control the appearance of plot symbols and lines, and define interpolation methods. They can specify

- the shape, size, and color of the plot symbols that mark the data points
- plot line style, color, and width
- an interpolation method for plotting data
- how missing values are treated in interpolation calculations.

SYMBOL definitions are assigned either by default by the GBARLINE procedure or explicitly with a plot request.

If no SYMBOL definition is currently in effect, the GBARLINE procedure produces a join interpolation using the default plot symbol. The default plot symbol for ActiveX device drivers is the square. For other devices, the default symbol is the plus sign (+).

See “SYMBOL Statement” on page 183 for a complete discussion of the features of the SYMBOL statement.

Examples

Example 1: Producing a Basic Bar Line Graph with Styles

Procedure Features:

BAR statement options:

SUMVAR=
DISCRETE

PLOT statement options:

SUMVAR=

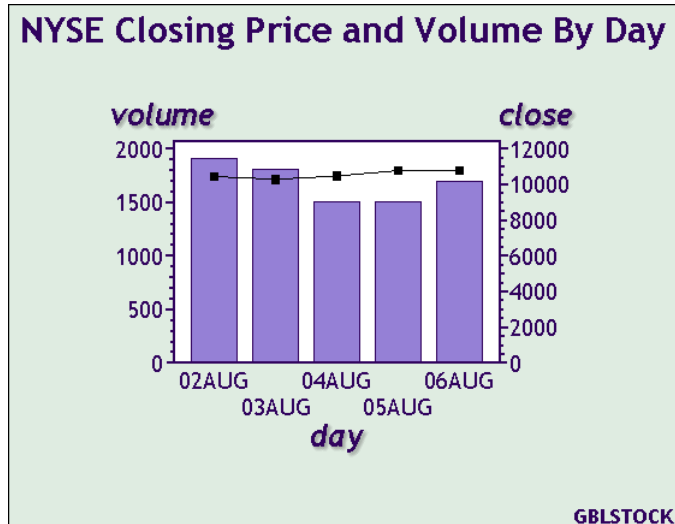
Other Features:

INFORMAT statement

FORMAT statement

STYLE= option on the ODS statement

Sample library member: GBLSTOCK



This example produces a basic bar line graph showing the volume and closing price for each of five days of trading activity on the New York Stock Exchange. The vertical bars indicate the volume, and the overlay plot graphs the closing price. It uses the ODS style ANALYSIS.

Set the graphics environment. Some graphics options may override style attributes, so if you are using a style, specify the minimum options needed by your graph.

```
options reset=all device=activex;
```

Define the odsout fileref. Specify the path of the HTML file where you want ODS to write the output.

```
filename odsout 'C:\your_web_path\';
```

Open the HTML output destination and specify the ANALYSIS style.

```
ods html file='gblstock.htm' path=odsout style=analysis;
```

Create the data set NYSE. NYSE contains one observation for each of five workdays. Each observation includes the date, closing price, and volume.

```
data nyse;
  informat day date9.;
  format day date5.;
  input day $ high low close volume;
```

```

        volume=volume/1000;
datalines;
02AUG2002 10478.76 10346.24 10426.91 1908809
03AUG2002 11042.92 10298.44 10274.65 1807543
04AUG2002 10498.22 10400.31 10456.43 1500656
05AUG2002 10694.47 10636.32 10762.98 1498403
06AUG2002 10801.12 10695.13 10759.48 1695602
;
run;

```

Define the title and footnote.

```

title1 "NYSE Closing Price and Volume By Day";
footnote j=r h=2 'GBLSTOCK';

```

Produce the bar line graph. The SUMVAR= option on the BAR statement specifies the variable whose values determine the height of the bars. The DISCRETE option creates a separate midpoint for each unique value of the bar variable. The SUMVAR= option on the PLOT statement specifies the variable whose values are used to calculate the overlay plot.

```

proc gbarline data=nyse;
    bar day / sumvar=volume discrete;
    plot / sumvar=close;
run;
quit;

```

Close the ODS HTML destination. You must close the HTML destination before you can view the output with a browser.

```
ods html close;
```

Example 2: Calculating Weighted Statistics

Procedure Features:

BAR statement options:

```

    AXIS=
    DESCENDING
    SUMVAR=

```

PLOT statement options:

```

    ASCENDING
    AXIS=
    FREQ=
    SUMVAR=

```

Other Features:

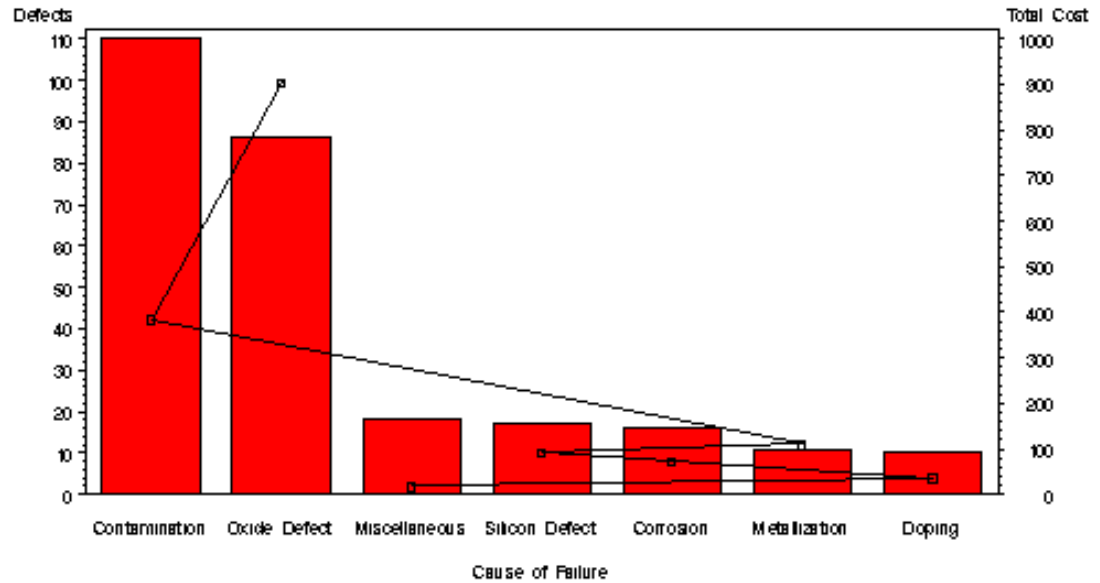
```

    AXIS statement
    SYMBOL statement

```


Sample library member: GBLWTSTA

The Cost of Defects



GBLWTSTA

This example uses the `FREQ=` option to calculate weighted statistics for the overlay plot. During the manufacture of a metal-oxide semiconductor (MOS) capacitor, two different cleaning processes were used by two manufacturing systems that were operating in parallel. Process A used a standard cleaning solution, while Process B used a different cleaning mixture that contained less particulate matter. For five consecutive days, the causes of failure with each process were recorded.

Set the graphics environment.

```
goptions reset=all gunit=pct border cback=white
      ftitle=swissb ftext=swiss htitle=5
      htext=2;
```

Create the data set FAILURE. Each observation specifies the manufacturing process that resulted in the defect, the date, the cause of the defects, and the total number of defects for that date. Each observation also contains a variable, `COST`, that specifies the cost associated with that type of defect.

```
data failure;
  label cause = 'Cause of Failure' ;
  input process $ 1-9 day $ 13-19 cause $ 23-36 count 40-41;
  datalines;
Process A   March 1   Contamination   15
Process A   March 1   Corrosion       2
Process A   March 1   Doping         1
Process A   March 1   Metallization   2
...more data lines...
Process B   March 5   Metallization   0
Process B   March 5   Miscellaneous   1
```

```

Process B   March 5   Oxide Defect      8
Process B   March 5   Silicon Defect    2
;
run;

data failure;
  set failure;
  if      cause='Contamination' then cost=3.5;
  else if cause='Metallization' then cost=10;
  else if cause='Oxide Defect'  then cost=10.5;
  else if cause='Corrosion'     then cost=4.5;
  else if cause='Doping'       then cost=3.6;
  else if cause='Silicon Defect' then cost=5.4;
  else                          cost=1.0;
  output;
run;

```

Define the title and footnote.

```

title1 "The Cost of Defects";
footnote j=r h=3 'GBLWTSTA ';

```

Define the labels for the axes.

```

AXIS1 label=("Defects");
AXIS2 label=("Total Cost");

```

Specify the symbol, color, and symbol size to use for the overlay plot.

```

symbol1 v=square c=black h=2;

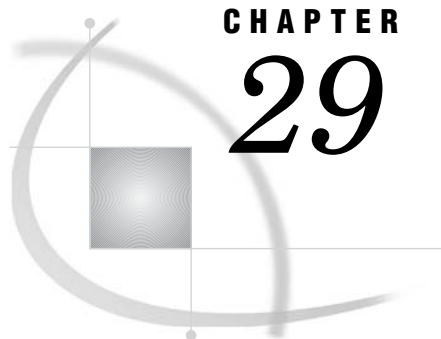
```

Produce the bar line graph. The SUMVAR= option on the BAR statement specifies the variable that determines the height of the bars. The SUMVAR= option on the PLOT statement specifies the plot variable. GBARLINE multiplies the value of the FREQ= variable by the value of the COUNT variable, and uses the result to determine the plot points.

```

proc gbarline data=failure;
  bar cause / sumvar=count
             axis=axis1
             descending;
  plot / sumvar=count
        freq=cost
        axis=axis2
        ascending;
run;
quit;

```



CHAPTER 29

The GCHART Procedure

<i>Overview</i>	774
<i>About Block Charts</i>	774
<i>About Bar Charts</i>	775
<i>About Pie, Detail Pie, and Donut Charts</i>	776
<i>About Star Charts</i>	777
<i>Concepts</i>	778
<i>About Chart Variables</i>	779
<i>Missing Values</i>	779
<i>About Midpoints</i>	780
<i>Character Values</i>	780
<i>Discrete Numeric Values</i>	780
<i>Continuous Numeric Values</i>	781
<i>Selecting and Ordering Midpoints</i>	781
<i>About Chart Statistics</i>	782
<i>Frequency</i>	782
<i>Cumulative Frequency</i>	782
<i>Percentage</i>	782
<i>Cumulative Percentage</i>	783
<i>Sum</i>	783
<i>Mean</i>	783
<i>Calculating Weighted Statistics</i>	783
<i>About Patterns</i>	784
<i>Default Patterns and Outlines</i>	784
<i>User-Defined Patterns, Outlines, and Images</i>	785
<i>Version 6 Patterns</i>	785
<i>Procedure Syntax</i>	785
<i>PROC GCHART Statement</i>	786
<i>BLOCK Statement</i>	787
<i>HBAR, HBAR3D, VBAR, and VBAR3D Statements</i>	796
<i>PIE, PIE3D, and DONUT Statements</i>	818
<i>STAR Statement</i>	833
<i>Examples</i>	842
<i>Example 1: Specifying the Sum Statistic in a Block Chart</i>	842
<i>Example 2: Grouping and Subgrouping a Block Chart</i>	844
<i>Example 3: Specifying the Sum Statistic in Bar Charts</i>	846
<i>Example 4: Subgrouping a 3D Vertical Bar Chart</i>	848
<i>Example 5: Controlling Midpoints and Statistics in a Horizontal Bar Chart</i>	850
<i>Example 6: Generating Error Bars in a Horizontal Bar Chart</i>	854
<i>Example 7: Creating Bar Charts with Drill-down for the Web</i>	856
<i>Example 8: Specifying the Sum Statistic for a Pie Chart</i>	869
<i>Example 9: Subgrouping a Donut or Pie Chart</i>	872

Example 10: Ordering and Labeling Slices in a Pie Chart 873

Example 11: Assigning Patterns and Identifying Midpoints with a Legend 875

Example 12: Grouping and Arranging Pie Charts 877

Example 13: Specifying the Sum Statistic in a Star Chart 879

Example 14: Charting a Discrete Numeric Variable in a Star Chart 880

Example 15: Creating a Detail Pie Chart 883

References 884

Overview

The GCHART procedure produces six types of charts: block charts, horizontal and vertical bar charts, pie and donut charts, and star charts. These charts graphically represent the value of a statistic calculated for one or more variables in an input SAS data set. The charted variables can be either numeric or character. The procedure calculates these statistics:

- frequency or cumulative frequency counts
- percentages or cumulative percentages
- sums
- means.

Use the GCHART procedure to

- display and compare exact and relative magnitudes
- examine the contribution of parts to the whole
- analyze where data are out of balance.

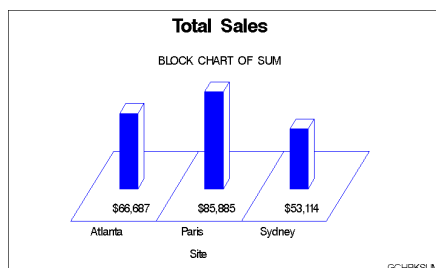
About Block Charts

Block charts display the relative magnitude of data with blocks of varying height, each set in a square that represents a category of data (midpoint). Because block charts do not use axes, they are most useful when the relative magnitude of the blocks is more significant than the exact magnitude of any particular block.

Figure 29.1 on page 774 shows a simple block chart of total sales for three manufacturing sites. Each site is a midpoint and occupies one square. The name of the site (the midpoint value) is printed below the square. Midpoint values are, by default, arranged in ascending order from left to right. The label below the midpoint grid names the chart variable.

Sales for the site (the chart statistic) are represented by the height of the block; sales amount (the formatted statistic value) is printed below the block. The heading above the blocks describes the type of statistic, in this case SUM.

Figure 29.1 Block Chart (GCHBKSUM)



The program for this chart is in Example 1 on page 842. For more information on producing block charts, see “BLOCK Statement” on page 787.

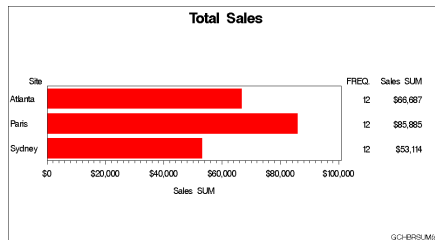
About Bar Charts

Horizontal and vertical bar charts display the magnitude of data with bars, each of which represents a category of data (midpoint). The length (or height) of the bars represents the value of the chart statistic for the corresponding midpoint.

Figure 29.2 on page 775 shows a simple horizontal bar chart of total sales for three manufacturing sites. Each site is a midpoint and is displayed as a bar. The name of the site (the midpoint value) is printed on the midpoint axis beside the bar. Midpoint values are, by default, arranged in ascending order from top to bottom of the chart and labeled with the name of the chart variable.

The chart statistics, in this case total sales for each site, are represented by the length of the bars. The response axis displays the scale of values for the chart statistic. The table of statistics to the right of the bars displays the exact statistic for each bar. Both a column in the table and the response axis are labeled with the name of the summary variable and the type of statistic.

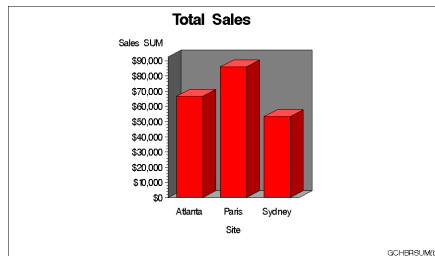
Figure 29.2 Horizontal Bar Chart (GCHBRSUM (a))



The program for this chart is Example 3 on page 846.

Figure 29.3 on page 775 shows the same data presented as a vertical bar chart. The two types of bar charts have essentially the same characteristics, except that horizontal bar charts by default display a table of statistic values to the right of the bars, while vertical bar charts can optionally display the statistic value above or inside of each bar.

Figure 29.3 Vertical Bar Chart (GCHBRSUM(b))



The program for this chart is Example 3 on page 846. For more information on producing horizontal and vertical bar charts, see “HBAR, HBAR3D, VBAR, and VBAR3D Statements” on page 796.

About Pie, Detail Pie, and Donut Charts

Pie and donut charts represent the relative contribution of parts to the whole by displaying data as wedge-shaped "slices" of a circle (either a "pie" or "donut"). Each slice represents a category of data (midpoint). The size of each slice (length of the arc) represents the contribution of the corresponding midpoint to the total chart statistic. Detail pie charts are pie charts with a second pie overlay that shows additional detail about the data that contributes to each of the outer pie's slices. Donut charts look like pie charts except that they have a hole in the middle in which you can place text.

Figure 29.4 on page 776 shows a pie chart of total sales for three manufacturing sites. Each site is a midpoint and is displayed as a slice. By default, the slices are ordered counterclockwise beginning at the 3 o'clock position.

Sales for the site (the chart statistic) are represented by the size of the slice. Both the sales amount (the formatted value of the chart statistic) and the name of the site (the midpoint value) are printed outside of the slice. You can also label pie slices with the percentage of the total statistic value that they represent. The heading above the pie describes the type of statistic (SUM), and names the summary variable (SALES) and the chart variable (SITE).

Figure 29.4 Pie Chart (GCHPISUM(a))

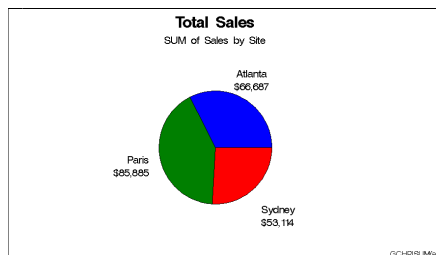


Figure 29.5 on page 776 show the three-dimensional version of the same pie chart.

Figure 29.5 3D Pie Chart (GCHPISUM(b))

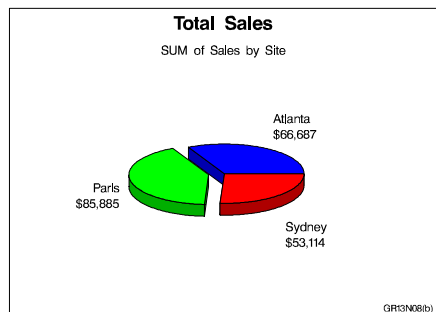
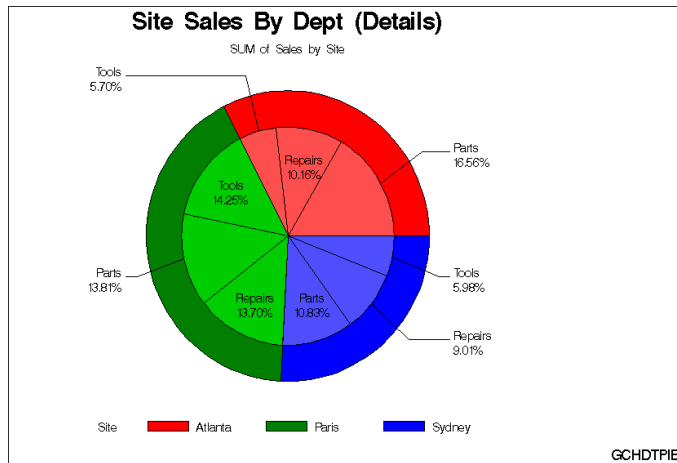


Figure 29.6 on page 777 shows a detail pie chart generated from the same data.

Figure 29.6 Detail Pie Chart (GCHDTPIE)



The programs for these charts are in Example 8 on page 869 and Example 15 on page 883. For more information on producing pie or donut charts, see “PIE, PIE3D, and DONUT Statements” on page 818.

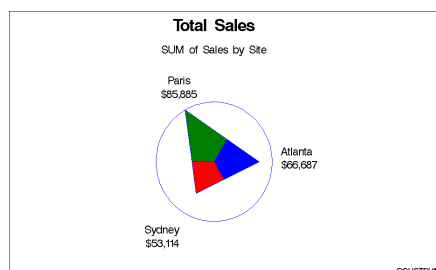
About Star Charts

Star charts display data as lines (“spines”) radiating from the center of a circle toward the perimeter. Each spine represents a category of data (midpoint). The length of a spine represents the magnitude of the chart statistic for that midpoint starting at the center of the circle, which by default represents 0. The radius of the circle is the length of the longest spine (greatest statistic value) in the chart. Instead of spines, star charts can also display the chart statistic as slices, which are enclosed areas formed by connecting the ends of the spines.

Figure 29.7 on page 777 shows the total sales for the three manufacturing sites as a star chart. Each site is a midpoint and is displayed as a spine. By default the ends of the spines are connected and they are ordered counterclockwise beginning at the 3 o’clock position.

Sales for the site (the chart statistic) are represented by the length of the spine. Both the sales amount (the formatted statistic value) and the name of the site (the midpoint value) are printed outside of the star chart. You can also label star charts with the percentage of the total statistic value that they represent. The heading above the chart describes the type of statistic (SUM), and names the summary variable (SALES) and the chart variable (SITE).

Figure 29.7 Star Chart (GCHSTSUM)



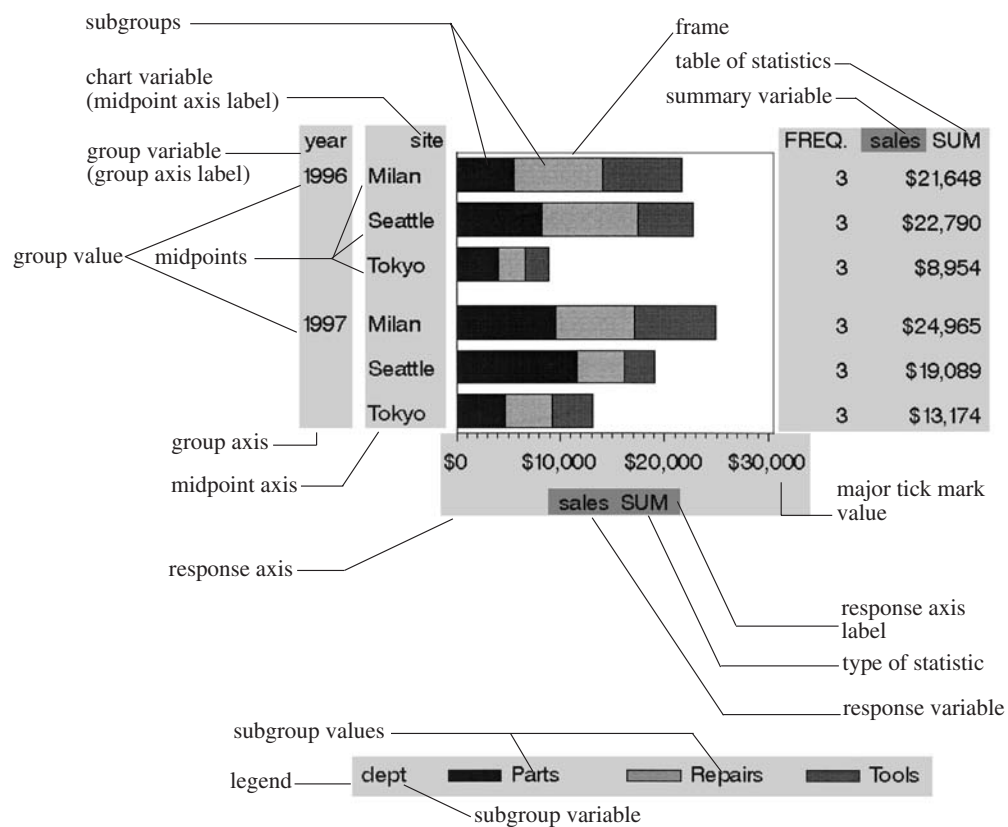
The program for this chart is Example 13 on page 879. For more information on producing star charts, see “STAR Statement” on page 833.

Concepts

The GCHART procedure produces charts based on the values of a *chart variable*. These values are represented by a set of *midpoints*. The chart itself displays information about the chart variable in the form of *chart statistics*.

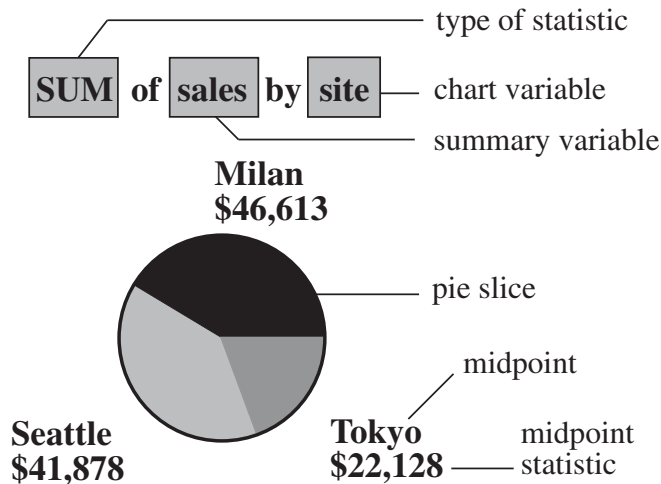
Figure 29.8 on page 778 and Figure 29.9 on page 779 illustrate these terms as well as other terms used with the GCHART procedure.

Figure 29.8 Terms Used with Bar Charts



Bar charts have two axes: a midpoint axis that shows the categories of data, and a response axis that displays the scale of values for the chart statistic. The response axis is divided into evenly spaced intervals identified with major tick marks that are labeled with the corresponding statistic value. Minor tick marks are evenly distributed between the major tick marks. Each axis is labeled with the chart variable name or label. The response axis is also labeled with the statistic type.

Figure 29.9 Terms Used with Pie and Donut Charts



Pie charts show statistics based on values of a variable called the chart variable. Generally, the values of the chart variable are represented by the slices in the chart. Next to each pie slice a number (or character string) appears that identifies the value or range of values assigned to that slice by the GCHART procedure. This number (or character string) is known as the *midpoint* for that slice. The statistic value for each midpoint is displayed beneath the midpoint. The slices in the chart represent all the values of the chart variable included in the chart. The number of degrees included in each slice represents the statistic value for the midpoint.

About Chart Variables

The *chart variable* is the variable in the input data set whose values determine the categories of data represented by the bars, blocks, slices, or spines. The chart variable generates the midpoints to which each observation in the data set contribute.

The chart variable can be either character or numeric. Character chart variables contain character values, which are always discrete. Numeric chart variables fall into two categories: discrete and continuous.

- *Discrete variables* contain a finite number of specific numeric values that are to be represented on the chart. For example, a variable that contains years, such as 1984 or 2001, is a discrete variable.
- *Continuous variables* contain a range of numeric values that are to be represented on the chart. For example, a variable of temperature data that contains real values between 0 and 212 is a continuous variable.

Numeric chart variables are always treated as continuous variables unless the DISCRETE option is used in the action statement.

Missing Values

By default, the GCHART procedure ignores missing midpoint values for the chart variable. If you specify the MISSING option, then missing values are treated as a valid midpoint and are included on the chart. Missing values for the group and subgroup variables are always treated as valid groups and subgroups.

When the value of the variable that is specified in the FREQ= option is missing, 0, or negative, the observation is excluded from the calculation of the chart statistic.

When the value of the variable specified in the SUMVAR= option is missing, the observation is excluded from the calculation of the chart statistic.

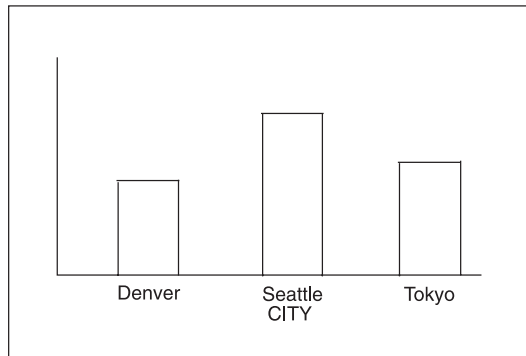
About Midpoints

Midpoints are the values of the chart variable that identify categories of data. By default, midpoints are selected or calculated by the procedure. The way the procedure handles the midpoints depends on whether the values of the chart variable are character, discrete numeric, or continuous numeric.

Character Values

A character chart variable generates a midpoint for each unique value of the variable. For example, if the chart variable CITY contains the names of three different cities, each city is a midpoint, resulting in three midpoints for the chart:

Figure 29.10 Character Midpoints

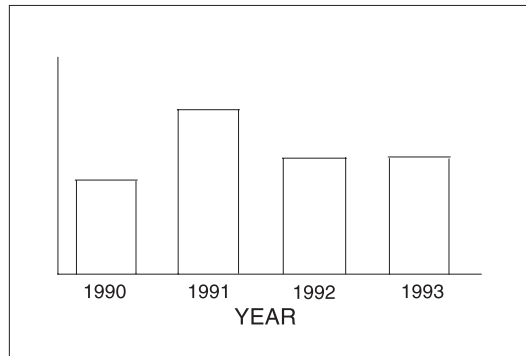


(In pie charts, midpoint values that compose a small percentage of the total for the chart may be placed in the OTHER slice and will not produce a separate midpoint.)

By default, character midpoints are arranged in alphabetic order. If a character variable has an associated format, the values are arranged in order of the formatted values.

Discrete Numeric Values

A numeric chart variable used with the DISCRETE option generates a midpoint for each unique value of the chart variable. For example, the numeric variable YEAR used with DISCRETE produces one midpoint for each year:

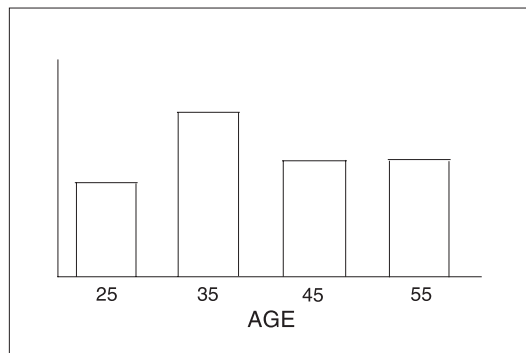
Figure 29.11 Discrete Numeric Midpoints

By default, numeric midpoints are arranged in ascending order. If the numeric variable has an associated format, each formatted value generates a separate midpoint. Formatted numeric variables are arranged in ascending order according to their unformatted numeric values.

Continuous Numeric Values

A continuous numeric variable generates midpoints that represent ranges of values. By default, the GCHART procedure determines the ranges, calculates the median value of each range, and displays the appropriate median value at each midpoint on the chart. A value that falls exactly halfway between two midpoints is placed in the higher range.

For example, the numeric variable AGE produces four midpoints, each of which represents a ten-year age range; the median value of the range is displayed at each midpoint:

Figure 29.12 Continuous Numeric Midpoints

By default, midpoints of ranges are arranged in ascending order.

Selecting and Ordering Midpoints

For character or discrete numeric values, you can use the MIDPOINTS= option to rearrange the midpoints or to exclude midpoints from the chart. For example, to change the default alphabetic order of the midpoints in Figure 29.10 on page 780, specify

```
midpoints='Tokyo' 'Denver' 'Seattle'
```

To exclude the midpoint for Denver, specify

```
midpoints='Tokyo' 'Seattle'
```

In this case, values excluded by the option are not included in the calculation of the chart statistic.

You can order or select discrete numeric midpoint values just as you do character values, but you omit the quotation marks when specifying numeric values.

For continuous numeric variables, use the LEVELS= or MIDPOINTS= option to change the number of midpoints, to control the range of values each midpoint represents, or to change the order of the midpoints. To control the range of values each midpoint represents, use the MIDPOINTS= option to specify the median value of each range. For example, to select the ranges 20–29, 30–39, and 40–49, specify

```
midpoints=25 35 45
```

Alternatively, to select the number of midpoints that you want and let the procedure calculate the ranges and medians, use the LEVELS= option.

You can also use formats to control the ranges of continuous numeric variables, but in that case the values are no longer continuous but discrete.

Note: You cannot use the MIDPOINTS= option to exclude continuous numeric values from the chart because values below or above the ranges specified by the option are automatically included in the first and last midpoints, respectively. To exclude continuous numeric values from a chart, use a WHERE statement in a DATA step or the WHERE= DATA set option. Δ

See also the description of the LEVELS= and MIDPOINTS= options for the appropriate statement.

About Chart Statistics

The *chart statistic* is the statistical value calculated for the chart variable and represented by each block, bar, or slice. The GCHART procedure calculates six chart statistics; the default statistic is frequency.

The examples given in the descriptions of these statistics assume a data set with two variables, CITY and SALES. The values of CITY are **Denver**, **Seattle**, and **Tokyo**. There are 21 observations: seven for Denver, nine for Seattle, and five for Tokyo.

Frequency

The frequency statistic is the total number of observations in the data set for each midpoint. For example, seven observations of the chart variable, CITY, contain the value **Denver**, so the frequency for the **Denver** midpoint is 7.

Cumulative Frequency

The cumulative frequency statistic adds the frequency for the current midpoint to the frequency of all of the preceding midpoints. For example, the frequency for the **Denver** midpoint is 7, and the frequency for the next midpoint, **Seattle**, is 9, so the cumulative frequency for **Seattle** is 16.

You cannot request cumulative frequency with the DONUT, PIE, PIE3D, or STAR statements.

Percentage

The percentage statistic is calculated by dividing the frequency for each midpoint by the total frequency count for all midpoints in the chart or group and multiplying it by

100. For example, the frequency count for the **Denver** midpoint is 7 and the total frequency count for the chart is 21, so the percentage statistic for **Denver** is 33.3%.

Cumulative Percentage

The cumulative percentage statistic adds the percentage for the current midpoint to the percentage for all of the preceding midpoints in the chart or group. For example, the percentage for the **Denver** midpoint is 33.3, and the percentage for the next midpoint, **Seattle**, is 42.9, so the cumulative percentage for **Seattle** is 76.2.

You cannot request cumulative percentage with the DONUT, PIE, PIE3D, or STAR statements.

Sum

The sum statistic is the total of the values for the SUMVAR= variable for each midpoint. For example, if you specify SUMVAR=SALES and the values of the SALES variable for the seven **Denver** observations are **8734, 982, 1504, 3207, 4502, 624, and 918**, the sum statistic for the **Denver** midpoint is 20,471.

You must use the SUMVAR= option to specify the variable for which you want the sum statistic.

Mean

The mean statistic is the average of the values for the SUMVAR= variable for each midpoint. For example, if TYPE=MEAN and SUMVAR=SALES, the mean statistic for the **Denver** midpoint is 2924.42.

You must use the SUMVAR= option to specify the variable for which you want the mean statistic.

Calculating Weighted Statistics

By default, each observation is counted only once in the calculation of the chart statistic. To calculate weighted statistics in which an observation can be counted more than once, use the FREQ= option. This option identifies a variable whose values are used as a multiplier for the observation in the calculation of the statistic. If the value of the FREQ= variable is missing, 0, or negative, the observation is excluded from the calculation.

If you use the SUMVAR= option, then the SUMVAR= variable value for an observation is multiplied by the FREQ= variable value for the observation for use in calculating the chart statistic.

For example, to use a variable called COUNT to produce weighted statistics, assign FREQ=COUNT. If you also assign the variable HEIGHT to the SUMVAR= option, then the following table shows how the values of COUNT and HEIGHT would affect the statistic calculation:

Value of COUNT	Value of HEIGHT	Number of times the observation is used	Value used for HEIGHT
1	55	1	55
5	65	5	325
.	63	0	-
-3	60	0	-

By default, the percentage and cumulative percentage statistics are calculated based on the frequency. If you want to chart a percentage or cumulative percentage based on a sum, you can use the `FREQ=` option to specify a variable to use for the "sum" calculation and specify the `PCT` statistic, as shown in this example:

```
freq=count type=pct
```

Because the variable that is used by `FREQ=` determines the number of times an observation is counted, the value of `COUNT` is the equivalent of the sum statistic.

See also the descriptions of the `TYPE=`, `SUMVAR=`, and `FREQ=` options for the action statements.

About Patterns

When a chart needs one or more patterns, the procedure uses either

- default patterns and outlines that are automatically generated by SAS/GRAPH or
- patterns, colors, outlines, and images that are defined by `PATTERN` statements, graphics options, and procedure options.

The following sections summarize pattern behavior for the `GCHART` procedure. For more information, see "PATTERN Statement" on page 169.

Default Patterns and Outlines

In general, the default pattern that the `GCHART` procedure uses is a solid fill that it rotates once through the colors list, skipping the foreground color. The procedure also outlines all areas in the foreground color. (Typically, the foreground color is the first color in the device's colors list.)

Specifically, the `GCHART` procedure uses default patterns and outlines when you

- do not specify *any* `PATTERN` statements, and
- do not use the `CPATTERN=` graphics option, and
- do not use the `COLORS=` graphics options (that is, you use the device's default colors list and it has more than one color), and
- do not use the `COUTLINE=` option in the action statement.

If all of these conditions are true, then the `GCHART` procedure

- selects the first default fill pattern, which is always solid, and rotates it through the colors list, generating one solid pattern for each color. If the first color in the device's colors list is black (or white), the procedure skips that color and begins generating patterns with the next color.
- uses the foreground color to outline every patterned area.

If the procedure needs additional patterns, `GCHART` selects the next default pattern fill that is appropriate to the type of chart and rotates it through the colors list, skipping the foreground color as before. The procedure continues in this fashion until it has generated enough patterns for the chart.

Changing any of these conditions may change or override the default behavior:

- If you specify a colors list with the `COLORS=` option in a `GOPTIONS` statement and the list contains more than one color, the procedure rotates the default solid pattern through that list, using every color, even if the foreground color is black (or white). The default outline color remains the foreground color.
- If you specify either `COLORS=(one-color)` or the `CPATTERN=` graphics option, the default fill pattern changes from solid to the list of appropriate hatch patterns. The procedure uses the specified color to generate one pattern definition for each hatch pattern in the list. The default outline color remains the foreground color.

- Whenever there are PATTERN definitions in effect, whether or not the GCHART procedure can use them, the default outline color for all patterns changes from foreground to SAME, as described in the following section.

For a description of these graphics options, see Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261.

User-Defined Patterns, Outlines, and Images

You can use PATTERN statements to explicitly specify patterns, including color or fill type or both. You can also specify images to fill the bars of two-dimensional bar charts. For complete information on all patterns, see “PATTERN Statement” on page 169. See also the section on controlling patterns and colors for each chart type.

When you use PATTERN statements, the procedure uses the specified patterns until all of the PATTERN definitions they generate have been used. Then, if more patterns are required, it returns to the default pattern rotation.

Whenever you specify *any* PATTERN statement, the default pattern outline changes. Instead of the foreground color, the outline color is the same as the fill color; for example, a blue bar has a blue outline. The effect is the same as specifying COUTLINE=SAME. Even when the procedure runs out of user-defined patterns and generates default patterns, the outlines continue to match the interior pattern color.

To change the outline color of any pattern, whether it’s a default or user-defined pattern, use the COUTLINE= option in the action statement that generates the chart.

Two-dimensional bar charts created with the HBAR and VBAR statements can use the PATTERN statement to fill specified bars with specified images. For details, see the IFRAME= option on page 806 and the IBACK= goption “Controlling Bar Chart Patterns, Colors, and Images” on page 816.

Other means of including images in charts include adding background images to two- and three-dimensional bar charts. For two-dimensional bar charts created with HBAR and VBAR statements, the IBACK= goption “IBACK” on page 317 specifies image files that fill the backplane frame. To fill the backplane frame of a three-dimensional image created with the HBAR3D or VBAR3D statements, use the IFRAME= option on page 806. For further information, including a listing of recognized image file types, see “Image File Types Supported by SAS/GRAPH” on page 106.

Version 6 Patterns

If you specify the V6COMP graphics option, then the procedure generates patterns by rotating the appropriate Version 6 default patterns through all of the colors in the colors list. With V6COMP, all patterns are outlined in the same color as the fill.

Procedure Syntax

Requirements: At least one BLOCK, HBAR, HBAR3D, VBAR, VBAR3D, PIE, PIE3D, DONUT, or STAR statement is required.

Global statements: AXIS, FOOTNOTE, GOPTIONS, LEGEND, PATTERN, TITLE

Reminder: The procedure can include the BY, FORMAT, LABEL, and WHERE statements as well as the SAS/GRAPHNOTE statement.

Supports:

- RUN-group processing
 - Output Delivery System (ODS)
-

```

PROC GCHART<DATA=input-data-set>
    <ANNOTATE=Annotate-data-set>
    <GOUT=<libref.>output-catalog>
    <IMAGEMAP=output-data-set>;
BLOCK chart-variable(s) </ option(s)>;
HBAR | HBAR3D | VBAR | VBAR3Dchart-variable(s) </ option(s)>;
PIE | PIE3D | DONUT chart-variable(s) </ option(s)>;
STAR chart-variable(s) </ option(s)>;

```

PROC GCHART Statement

Identifies the data set containing the chart variables. Optionally specifies annotation and an output catalog.

Requirements: An input data set is required.

Syntax

```

PROC GCHART<DATA=input-data-set>
    <ANNOTATE=Annotate-data-set>
    <GOUT=<libref.>output-catalog>
    <IMAGEMAP=output-data-set>;

```

Options

PROC GCHART statement options affect all graphs produced by the procedure.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate all graphs that are produced by the GCHART procedure. To annotate individual graphs, use ANNOTATE= in the action statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

DATA=*input-data-set*

specifies the SAS data set that contains the variable(s) to chart. By default, the procedure uses the most recently created SAS data set.

See also: “SAS Data Sets” on page 29 and “About Chart Variables” on page 779

GOUT=<*libref.*>*output-catalog*

specifies the SAS catalog in which to save the graphics output that is produced by the GCHART procedure. If you omit the libref, SAS/GRAPH looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist.

See also: “Storing Graphics Output in SAS Catalogs” on page 53

IMAGEMAP=*output-data-set*

creates a temporary SAS data set that is used to generate an image map in an HTML output file. The information in the image map data set includes the shape and coordinates of the elements in the graph and drill-down URLs that have been associated with those elements. The drill-down URLs are provided by one or two variables in the input data set. These variables are identified to the GCHART procedure with the HTML= and/or HTML_LEGEND= options.

The %IMAGEMAP macro generates the image map in the HTML output file. The macro takes two arguments, the name of the image map data set and the name or fileref of the HTML file, as shown in the following example:

```
%imagemap(imgmapds, myimgmap.html);
```

Not supported by: Java, ActiveX

BLOCK Statement

Creates block charts in which the height of the blocks represents the value of the chart statistic for each category of data.

Requirements: At least one chart variable is required.

Global statements: LEGEND, PATTERN, TITLE, FOOTNOTE

Supports: Drill-down functionality

Description

The BLOCK statement specifies the variable or variables that define the categories of data to chart. This statement automatically

- determines the midpoints
- calculates the chart statistic for each midpoint (the default is FREQ)
- scales the blocks according to the statistic value
- assigns patterns and colors to the block faces and the grid; the default block pattern is solid.

You can use statement options to select or order the midpoints (blocks), to change the type of chart statistic, and to modify the appearance of the chart. You can also specify additional variables by which to group, subgroup, or sum the data.

Block charts allow grouping, which organizes the blocks into rows based on the values of a group variable, and subgrouping, which subdivides the blocks into segments based on the values of a subgroup variable.

In addition, you can use global statements to modify the block patterns and the legend, as well as add titles, footnotes, and notes to the chart. You can also use an Annotate data set to enhance the chart.

Note: If you get a message that the chart is too large to display on your terminal or printer, try one or both of the following: Δ

- reduce the size of the character cells defined for the output device by specifying larger values for the HPOS= and VPOS= graphics options
- decrease the size of the chart text with the HTEXT= graphics option.

See “About the Graphics Output Area” on page 34 for details .

Syntax

BLOCK *chart-variable(s)* *</ option(s)>*;

option(s) can be one or more options from any or all of the following categories:

- appearance options
 - ANNOTATE=*Annotate-data-set*

- BLOCKMAX=*max-value*
- CAXIS=*grid-color*
- COUTLINE=*block-outline-color* | SAME
- CTEXT=*text-color*
- LEGEND=LEGEND<1...99>
- NOHEADING
- NOLEGEND
- PATTERNID=BY | GROUP | MIDPOINT | SUBGROUP
- WOUTLINE=*block-outline-width*
- midpoint options
 - DISCRETE
 - GROUP=*group-variable*
 - LEVELS=*number-of-midpoints*
 - MIDPOINTS=*value-list*
 - MIDPOINTS=OLD
 - MISSING
 - SUBGROUP=*subgroup-variable*
- statistic options
 - FREQ=*numeric-variable*
 - G100
 - SUMVAR=*summary-variable*
 - TYPE=*statistic*
- catalog entry description options
 - DESCRIPTION=*'entry-description'*
 - NAME=*'entry-name'*
- ODS options
 - HTML=*variable*
 - HTML_LEGEND=*variable*

Required Arguments

chart-variable(s)

specifies one or more variables that define the categories of data to chart. Each chart variable draws a separate chart. All variables must be in the input data set. Separate multiple chart variables with blanks. The values of a chart variable used with the BLOCK statement have a maximum length of 13.

See also: “About Chart Variables” on page 779

Options

Options in a BLOCK statement affect all graphs produced by that statement. You can specify as many options as you want and list them in any order. For details on specifying colors, see Chapter 6, “SAS/GRAPH Colors and Images,” on page 91. For a complete description of the graphics options, see Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261.

ANNOTATE=*Annotate-data-set*

ANNO=Annotate-data-set

specifies a data set to annotate charts produced by the BLOCK statement.

Note: Annotate coordinate systems 1, 2, 7, and 8 (data system coordinates) are not valid with block charts. △

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

BLOCKMAX=max-value

specifies the chart statistic value of the tallest block on the chart. This option lets you produce a series of block charts using the same scale. All blocks are rescaled as if *max-value* were the maximum value on the chart.

Not supported by: Java, ActiveX

CAXIS=grid-color

specifies the color for the midpoint grid. By default, the midpoint grid uses the foreground color (usually the first color in the colors list).

Featured in: Example 2 on page 844

COUTLINE=block-outline-color | SAME

outlines all blocks or all block segments and legend values in the subgroup legend (if it appears) using the specified color. SAME specifies that the outline color of a block or a block segment or a legend value is the same as the interior pattern color.

The default outline color depends on the PATTERN statement:

- If you do not specify a PATTERN statement, the default outline color is black for the Java or ActiveX devices. Otherwise, the default outline color is the foreground color (the first color in the colors list).
- If you specify the PATTERN statement or the V6COMP graphics option, the default is COUTLINE=SAME.

Note: If you specify empty patterns, (VALUE=EMPTY in a PATTERN statement) you should not change the outline color from the default value, SAME, to a single color. Otherwise all the outlines will be one color and you will not be able to distinguish between the empty areas. △

See also: “Controlling Block Chart Patterns and Colors” on page 794 and “About Patterns” on page 784

Featured in: Example 2 on page 844

Not supported by: Java (partial), ActiveX (partial)

CTEXT=text-color

specifies a color for all text on the chart. Text includes the values and labels for the midpoint grid, the subgroup legend, and the descriptive statistic values. For the Java and ActiveX devices, the default color is black. For other devices, if you omit CTEXT=, PROC GCHART searches for a color specification in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the first color in the colors list (the default).

CTEXT= is overridden by the COLOR= suboption of the LABEL= or VALUE= option in a LEGEND definition assigned to the subgroup legend. The suboption determines the color of the legend label or the color of the legend value descriptions, respectively.

DESCRIPTION='entry-description'**DES='entry-description'**

specifies the description of the catalog entry for the chart. The maximum length for *entry-description* is 256 characters. The description does not appear on the chart. By

default, the GCHART procedure assigns a description of the form BLOCK CHART OF *variable*, where *variable* is the name of the chart variable.

The *entry-description* can include the #BYLINE, #BYVAL, and #BYVAR substitution options, which work as they do when used on TITLE, FOOTNOTE, and NOTE statements. Refer to the description of the options on page 222, and “Substituting BY Line Values in a Text String” on page 226. The 256-character limit applies before the substitution takes place for these options; thus, if in the SAS program the entry-description text exceeds 256 characters, it is truncated to 256 characters, and then the substitution is performed.

The descriptive text is shown in each of the following:

- the "description" portion of the Results window
- the catalog-entry properties that you can view from the Explorer window
- the Table of Contents that is generated when you use CONTENTS= on an ODS statement (see “Linking to Output through a Table of Contents” on page 495), assuming the GCHART output is generated while the contents page is open
- the Description field of the PROC GREPLAY window
- the data tip text for web output (depending on the device driver you are using). See “Adding Data Tips to Web Presentations” on page 568 for details.

DISCRETE

treats a numeric chart variable as a discrete variable rather than as a continuous variable. The GCHART procedure creates a separate midpoint and, hence, a separate grid square and block for each unique value of the chart variable. If the chart variable has a format associated with it, each formatted value is treated as a midpoint.

The LEVELS= option is ignored when you use DISCRETE. The MIDPOINTS= option overrides DISCRETE.

FREQ=*numeric-variable*

specifies a variable whose values weight the contribution of each observation in the computation of the chart statistic. Each observation is counted the number of times specified by the value of *numeric-variable* for that observation. If the value of *numeric-variable* is missing, 0, or negative, the observation is not used in the statistic calculation. Non-integer values of *numeric-variable* are truncated to integers.

FREQ= is valid with all chart statistics.

Because you cannot use the PERCENT, CPERCENT, FREQ, or CFREQ statistics with the SUMVAR= option, you must use the FREQ= option to calculate percentages, cumulative percentages, frequencies, or cumulative frequencies based on a sum.

The statistics will not be affected by applying a format to *numeric-variable*.

See also: “Calculating Weighted Statistics” on page 783

G100

calculates the percentage and cumulative percentage statistics separately for each group. When you use G100, the individual percentages reflect the contribution of the midpoint to the group and total 100 percent for each group. G100 is ignored unless you also use the GROUP= option.

By default, the individual percentages reflect the contribution of the midpoint to the entire chart and total 100 percent for the entire chart.

GROUP=*group-variable*

organizes the data according to the values of *group-variable*. *Group-variable* can be either character or numeric and is always treated as a discrete variable. The group variable can have up to 12 different values.

GROUP= produces a group grid that contains a separate row of blocks for each unique value of the group variable. Each row contains a square for each midpoint. The groups are arranged from front to back in ascending order of the group variable

values. These values are printed to the left of each row; the group variable name or label is printed above the list of group values.

By default, each group includes all midpoints, even if no observations for the group fall within the midpoint range. Missing values for *group-variable* are treated as a valid group.

Featured in: Example 2 on page 844

HTML=*variable*

identifies the variable in the input data set whose values create links in the HTML file that is created by the ODS statement. These links are associated with an area of the chart and point to the data or graph you wish to display when the user drills down on the area. The values of *variable* can be up to 1024 characters long. Characters after the 1024-character limit (including any closing quotes) are truncated.

HTML_LEGEND=*variable*

identifies the variable in the input data set whose values create links in the HTML file that is created by the ODS statement. These links are associated with a legend value and point to the data or graph that you wish to display when the user drills down on the value. The values of *variable* can be up to 1024 characters long. Characters after the 1024-character limit (including any closing quotes) are truncated.

Not supported by: Java, ActiveX

LEGEND=LEGEND<1...99>

assigns the specified LEGEND definition to the legend generated by the SUBGROUP= option. The LEGEND= option itself does *not* generate a legend.

LEGEND= is ignored if

- SUBGROUP= is not used.
- the specified LEGEND definition is not in effect.
- the NOLEGEND option is used.
- the PATTERNID= option is set to any value other than SUBGROUP; that is, the value of PATTERNID= is BY or GROUP or MIDPOINT.

To create a legend based on the chart midpoints instead of the subgroups, use the chart variable as the subgroup variable:

```
block city / subgroup=city;
```

The Java and ActiveX devices do not support all LEGEND statement options. See “LEGEND Statement” on page 151 for more information.

See also: SUBGROUP= on page 793 and “LEGEND Statement” on page 151

Featured in: Example 2 on page 844

Not supported by: Java (partial), ActiveX (partial)

LEVELS=*number-of-midpoints*

specifies the number of midpoints for the numeric chart variable. The range for each midpoint is calculated automatically using the algorithm described in Terrell and Scott (1985). LEVELS= is ignored if

- the chart variable is character type.
- the DISCRETE option is used.
- the MIDPOINTS= option is used.

MIDPOINTS=*value-list*

specifies the midpoint values for the blocks. The way you specify *value-list* depends on the type of variable:

- For numeric chart variables, *value-list* is either an explicit list of values, or a starting and an ending value with an interval increment, or a combination of both forms:

n <...*n*>

n TO *n* <BY *increment*>

n <...*n*> TO *n* <BY *increment*> <*n* <...*n*>>

If a numeric variable has an associated format, the specified values must be the *unformatted* values.

By default, numeric variable values are treated as continuous (if you omit the DISCRETE option), and

- the lowest midpoint consolidates all data points from negative infinity to the median of the first two midpoints
- the highest midpoint consolidates all data points from the median of the last two midpoints up to infinity
- all other values in *value-list* specify the median of a range of values, and the GCHART procedure calculates the midpoint values.

If you include the DISCRETE option, each value in *value-list* specifies a unique numeric value.

- For character chart variables, *value-list* is a list of unique character values enclosed in quotation marks and separated by blanks:

'*value-1*' <...'value-*n*'>

If a character variable has an associated format, the specified values must be the *formatted* values.

For a complete description of *value-list*, see the ORDER= on page 130 option in the AXIS statement.

If *value-list* for either type of variable specifies so many midpoints that the axis values overwrite each other, the values may be unreadable. In this case the procedure writes a warning to the SAS log. On many devices, you can correct crowded values by increasing the number of cells in your graphics display using the HPOS= and VPOS= graphics options.

See also: “About Midpoints” on page 780

Featured in: Example 2 on page 844

MIDPOINTS=OLD

generates default midpoints using the Nelder algorithm (*Applied Statistics* 25:94–7, 1976). The MIDPOINTS=OLD option is ignored unless the chart variable is numeric.

MISSING

accepts a missing value as a valid midpoint for the chart variable. By default, observations with missing values are ignored. Missing values are always valid for the group and subgroup variables.

NAME='entry-name'

specifies the name of the catalog entry for the graph. The maximum length for *entry-name* is eight characters. The default name is GCHART. If the name duplicates an existing entry name, then SAS/GRAPH software adds a number to the duplicate name to create a unique name—for example, GCHART1.

NOHEADING

suppresses the heading describing the type of statistic. For the Java and ActiveX devices, NOHEADING is the default. For other devices, by default the heading is printed at the top of each block chart.

Featured in: Example 2 on page 844

Not supported by: Java, ActiveX

NOLEGEND

suppresses the legend automatically generated by the SUBGROUP= option. NOLEGEND is ignored if the SUBGROUP= option is not used.

PATTERNID=BY | GROUP | MIDPOINT | SUBGROUP

specifies the way fill patterns are assigned. By default, PATTERNID=SUBGROUP. Values for PATTERNID= are as follows:

BY

changes patterns each time the value of the BY variable changes. All blocks use the same pattern if the GCHART procedure does not include a BY statement.

GROUP

changes patterns every time the value of the group variable changes. All blocks in each group (row) use the same pattern, but a different pattern is used for each group.

MIDPOINT

changes patterns every time the midpoint value changes. If you use the GROUP= option, the respective midpoint patterns are repeated for each group.

SUBGROUP

changes patterns every time the value of the subgroup variable changes. The blocks must be subdivided by the SUBGROUP= option for the SUBGROUP value to have an effect. Without SUBGROUP=, all block faces have the same pattern.

Note: If you use the SUBGROUP= option and specify a PATTERNID= value other than SUBGROUP, the block segments use the same pattern and are indistinguishable. △

See also: “Controlling Block Chart Patterns and Colors” on page 794

Featured in: Example 7 on page 856

SUBGROUP=*subgroup-variable*

divides the blocks into segments according to the values of *subgroup-variable*. *Subgroup-variable* can be either character or numeric and is always treated as a discrete variable. SUBGROUP= creates a separate segment within each block for every unique value of the subgroup variable for that midpoint.

If PATTERNID=SUBGROUP (the default setting), each segment is filled with a different pattern, and a legend providing a key to the patterns is automatically generated. If the value of PATTERNID= is anything other than SUBGROUP, the segments are all the same color, the legend is suppressed, and the subgrouping effect is lost.

By default the legend appears at the bottom of the chart. To modify the legend, assign a LEGEND definition with the LEGEND= option. To suppress the legend, specify NOLEGEND.

See also: “LEGEND Statement” on page 151

Featured in: Example 2 on page 844

SUMVAR=*summary-variable*

specifies a numeric variable for sum or mean calculations. The GCHART procedure calculates the sum or, if requested, the mean of *numeric-variable* for each midpoint. The resulting statistics are represented by the height of the blocks in each square. The values of a summary variable used with the BLOCK statement have a maximum length of 8.

When you use SUMVAR=, the TYPE= option value must be either SUM or MEAN. With SUMVAR=, the default is TYPE=SUM.

Featured in: Example 1 on page 842

TYPE=*statistic*

specifies the chart statistic.

- If the SUMVAR= option is not used, *statistic* can be one of the following:

FREQ
frequency (the default)

CFREQ
cumulative frequency

PERCENT PCT
percentage

CPERCENT CPCT
cumulative percentage

- If SUMVAR= is used, *statistic* can be either:

SUM
sum (the default)

MEAN
mean

Because you cannot specify the statistics PERCENT, CPERCENT, FREQ, or CFREQ in conjunction with the SUMVAR= option, you must use FREQ= to calculate percentages, cumulative percentages, frequencies, or cumulative frequencies based on a sum. See also “Calculating Weighted Statistics” on page 783.

If you specify TYPE=MEAN and use the SUBGROUP= option, the height of the block represents the mean for the entire midpoint. The subgroup segments are proportional to the subgroup’s contribution to the sum for the block.

See also: “About Chart Statistics” on page 782

Featured in: Example 2 on page 844

WOULINE=*block-outline-width*

specifies the width of the block outline in pixels.

Not supported by: Java

Controlling Block Chart Patterns and Colors

Default patterns and outlines

In a block chart, only the front faces of the blocks display patterns. By default, the procedure

- fills the block faces with bar/block patterns, beginning with the default fill, SOLID, and rotating it through the colors list. When the solid patterns are exhausted, the procedure selects the next default bar/block pattern and rotates it through the colors list. It continues in this fashion until all of the required patterns have been assigned.

If you use the device’s default colors and the first color in the list is either black or white, the procedure does not create a pattern in that color. If you specify a colors list with the COLORS= graphics option, the procedure uses all of the colors in the list to generate the patterns.

- outlines blocks and block segments using the first color in the colors list.
- colors the midpoint grid with the first color in the colors list.

See “About Patterns” on page 784 for more information on how the GCHART procedure assigns default patterns and outlines.

User-defined patterns

To override the default patterns and select fills and colors for the blocks or block segments, use the PATTERN statement. Only bar/block patterns are valid; all other pattern fills are ignored. For a complete description of all bar/block patterns, see the description of PATTERN statement option VALUE= on page 171.

Whenever you use PATTERN statements, the default pattern outline color changes to SAME. That is, the outline color is the same as the fill color. To specify the outline color, use the COUTLINE= on page 789 option.

When patterns change

The PATTERNID= option controls when the pattern changes. By default, PATTERNID=SUBGROUP. Therefore, when you use the SUBGROUP= option to subdivide the blocks, the pattern automatically changes each time the subgroup value changes, and each subdivision of the block displays a different pattern. As a result, the number of values for the SUBGROUP= variable determines the number of block patterns on the chart. If you do not subdivide the blocks, all blocks use the same pattern.

Instead of changing the pattern for each subgroup, you can change the pattern for each midpoint, each group, or each BY group, by changing the value of the PATTERNID= option. See the PATTERNID= on page 793 option for details.

Axis color

By default, axis elements use the first color in the colors list. To change the grid color, use the CAXIS= option. To change the axis text color, use the CTEXT= option.

Controlling Block Chart Text

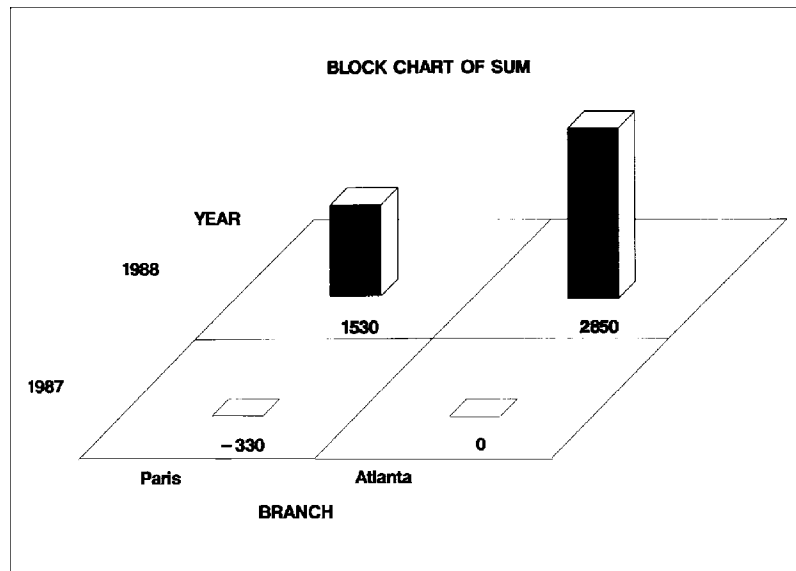
To control the font and size of text on the chart, use the FTEXT= and HTEXT= graphics options. See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for a description of these options.

Because block charts do not use AXIS statements, you must use a LABEL statement instead to suppress the label for the midpoint variable. See Example 2 on page 844.

Displaying Negative or Zero Values

The relative block heights in the chart represent the scaled value of the chart statistic value for the midpoint. If the statistic has a value of 0 or, in the case of sum and mean, a negative value, the base of the block is drawn in the square for the corresponding midpoint. Figure 29.13 on page 796 shows an example of a chart with 0 and negative statistic values.

Figure 29.13 Block Chart with 0 and Negative Statistic Values



HBAR, HBAR3D, VBAR, and VBAR3D Statements

Create horizontal or vertical bar charts in which the length or height of the bars represents the value of the chart statistic for each category of data.

Requirements: At least one chart variable is required.

Global statements: AXIS, LEGEND, PATTERN, TITLE, FOOTNOTE

Supports: Drill-down functionality

Description

The HBAR, HBAR3D, VBAR, and VBAR3D statements specify the variable or variables that define the categories of data to chart. These statements automatically

- determine the midpoints
- calculate the chart statistic for each midpoint (the default is FREQ)
- scale the response axis and the bars according to the statistic value
- determine bar width and spacing
- assign patterns to the bars; the default bar/block pattern is SOLID.
- draw a frame around the axis area using the first color in the colors list.

You can use statement options to select or order the midpoints (bars), to control the tick marks on the response axis, to change the type of chart statistic, to display specific statistics, and to modify the appearance of the chart. You can also specify additional variables by which to group, subgroup, or sum the data.

All bar charts allow grouping, which uses an additional category to organize the bars into groups, and subgrouping, which divides the bars into segments.

In addition, you can:

- use global statements to modify the axes (including requesting a logarithmic axis), the bar patterns, and the legend. See Chapter 7, “SAS/GRAPH Statements,” on page 121 for more information.

- add titles, footnotes, and notes to the chart. See “TITLE, FOOTNOTE, and NOTE Statements” on page 210 for more information.
- use an Annotate data set to enhance the chart. See Chapter 24, “Using Annotate Data Sets,” on page 587 for more information.
- display an image in the background of the chart. For HBAR3D and VBAR3D charts, see the IFRAME= option on page 806. For HBAR and VBAR charts, see the IBACK= option “IBACK” on page 317.
- display images in the bars of an HBAR or VBAR chart. See the PATTERN statement IMAGE= option on page 171.

Syntax

HBAR | HBAR3D | VBAR | VBAR3D *chart-variable(s) </option(s)>*;

option(s) can be one or more options from any or all of the following categories:

- appearance options
 - ANNOTATE=*Annotate-data-set*
 - CAUTOREF=*reference-line-color*
 - CAXIS=*axis-color*
 - CERROR=*error-bar-color*
 - CFRAME=*background-color*
 - COUTLINE=*bar-outline-color* | SAME
 - CREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*
 - CTEXT=*text-color*
 - FRAME | NOFRAME
 - GSPACE= *group-spacing*
 - IFRAME= *fileref* | '*external-file*'
 - IMAGESTYLE = TILE | FIT
 - LAUTOREF=*reference-line-type*
 - LEGEND=LEGEND<1...99>
 - LREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*
 - NOLEGEND
 - PATTERNID=BY | GROUP | MIDPOINT | SUBGROUP
 - SHAPE=3D-*bar-shape* (HBAR3D and VBAR3D only)
 - SPACE=*bar-spacing*
 - WIDTH=*bar-width*
 - WOUTLINE=*bar-outline-width*
- statistic options
 - CFREQ
 - CFREQLABEL=*'column-label'* | NONE (HBAR and HBAR3D only)
 - CLM=*confidence-level*
 - CPERCENT
 - CPERCENTLABEL=*'column-label'* | NONE (HBAR and HBAR3D only)
 - ERRORBAR=BARS | BOTH | TOP
 - FREQ
 - FREQLABEL=*'column-label'* | NONE (HBAR and HBAR3D only)
 - FREQ=*numeric-variable*
 - G100

INSIDE=*statistic*

MEAN

MEANLABEL=*'column-label'* | NONE (HBAR and HBAR3D only)

NOSTATS (HBAR and HBAR3D only)

OUTSIDE=*statistic*

PERCENT

PERCENTLABEL=*'column-label'* | NONE (HBAR and HBAR3D only)

SUM

SUMLABEL=*'column-label'* | NONE (HBAR and HBAR3D only)

SUMVAR=*summary-variable*

TYPE=*statistic*

□ midpoint options

DISCRETE

GROUP=*group-variable*

LEVELS=*number-of-midpoints* | ALL

MIDPOINTS=*value-list*

MIDPOINTS=OLD

MISSING

RANGE

SUBGROUP=*subgroup-variable*

□ axes options

ASCENDING

AUTOREF

AXIS=AXIS<1...99>

CLIPREF

DESCENDING

FRONTREF (HBAR3D and VBAR3D only)

GAXIS=AXIS<1...99>

MAXIS=AXIS<1...99>

MINOR=*number-of-minor-ticks*

NOAXIS

NOBASEREF

NOZERO

RANGE

RAXIS=*value-list* | AXIS<1...99>

REF=*value-list*

□ catalog entry description options

DESCRIPTION=*'entry-description'*

NAME=*'entry-name'*

□ ODS options

HTML=*variable*

HTML_LEGEND=*variable*

Required Arguments

chart-variable(s)

specifies one or more variables that define the categories of data to chart. Each chart variable draws a separate chart. All variables must be in the input data set. Multiple chart variables must be separated with blanks.

See also: “About Chart Variables” on page 779

Options

Options in an HBAR, HBAR3D, VBAR, or VBAR3D statement affect all graphs that are produced by that statement. You can specify as many options as you want and list them in any order. For details on specifying colors, see Chapter 6, “SAS/GRAPH Colors and Images,” on page 91. For details on specifying images, see “Specifying Images in SAS/GRAPH Programs” on page 106. For a complete description of the graphics options, see Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate charts produced by the bar chart statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

ASCENDING

arranges the bars in ascending order of the value of the chart statistic. By default, bars are arranged in ascending order of midpoint value, without regard to the lengths of the bars. ASCENDING reorders the bars from shortest to longest. In horizontal bar charts the ordering is top to bottom; in vertical bar charts the ordering is left to right.

If you also use the GROUP= option, the reordering is performed separately for each group, so the order of the midpoints may be different for each group.

ASCENDING overrides any midpoint order specified with the MIDPOINTS= option or specified in the ORDER= option in an AXIS statement assigned to the midpoint axis.

AUTOREF

draws a reference line at each major tick mark on the response axis. To draw reference lines at specific points on the response axis, use the REF= option.

By default, reference lines in 2D bar charts are drawn in front of the bars. To draw reference lines behind the bars, use the CLIPREF option.

By default, reference lines in 3D bar charts are drawn on the back plane of the axis. To draw reference lines in front of the bars, use the FRONTREF option.

Featured in: Example 5 on page 850

AXIS=AXIS<1...99>

See RAXIS= on page 812.

CAUTOREF=*reference-line-color*

specifies the color of reference lines drawn at major tick marks, as determined by the AUTOREF option. The default color is either the value of the CAXIS= option or the first color in the color list. To specify a line type for these reference lines, use the LAUTOREF= option.

CAXIS=*axis-color*

specifies a color for the response and midpoint axis lines and for the default axis area frame. If you omit the CAXIS= option, PROC GCHART searches for a color specification in this order:

- 1 the COLOR= option in AXIS definitions
- 2 the first color in the colors list (the default).

This option also specifies the default color for all reference lines.

CERROR=*error-bar-color*

specifies the color of error bars in bar charts. The default is the color of the response axis, which is controlled by the CAXIS= option.

CFRAME=*background-color*

CFR=*background-color*

specifies the color with which to fill the axis area in 2D bar charts or in the backplane in 3D bar charts.

The axis area color does not affect the frame color, which is always the same as the midpoint axis line color and controlled by the CAXIS= option. By default, the axis area in 2D bar charts is not filled.

CFRAME= is overridden by the NOFRAME and IFRAME= options.

Note: If the background color, the bar color, and the outline color are the same, you may not be able to distinguish the bars. \triangle

Featured in: Example 4 on page 848

CFREQ

displays the cumulative frequency statistic in the table of statistics and above vertical bars. Default statistics are suppressed when you request specific statistics. For vertical bar charts, this option is ignored if the bars are too narrow to avoid overlapping values or if the FREQ option is specified.

See also: “About Chart Statistics” on page 782, “Displaying Statistics in Horizontal Bar Charts” on page 815, and “Displaying Statistics in Vertical Bar Charts” on page 815

CFREQLABEL=*'column-label'* | NONE (HBAR and HBAR3D only)

specifies the text of the column label for the CFREQ statistic in the table of statistics. *Column-label* can be up to 32 characters long, but a single line of the label can be no more than 24 characters. By default, a label with more than one word will break as close to the center of the line as possible. A double space in the string forces a line break. To suppress the label, specify CFRREQLABEL=NONE.

Not supported by: Java, ActiveX

CLIPREF

clips the reference lines at the bars. This makes the reference lines appear to be behind the bars. Because CLIPREF is the default for 3D bar charts, it affects only 2D charts.

Featured in: Example 5 on page 850

CLM=*confidence-level*

specifies the confidence intervals to use when drawing error bars on a bar chart. Values for *confidence-level* must be greater than or equal to 50 and strictly less than 100. The default is 95. See ERRORBAR= for details on how error bars are computed and drawn.

Featured in: Example 6 on page 854

COUTLINE=*bar-outline-color* | SAME

outlines all bars or bar segments and legend values in the subgroup legend (if it appears) using the specified color. SAME specifies that the outline color of a bar or a bar segment or a legend value is the same as the interior pattern color.

The default outline color depends on the PATTERN statement:

- If you do not specify a PATTERN statement, the default outline color is black for the Java or ActiveX devices. Otherwise, the default outline color is the foreground color (the first color in the colors list).
- If you specify the PATTERN statement or the V6COMP graphics option, the default is COUTLINE=SAME.

Note: For 2D bar charts, if you specify empty patterns, (VALUE=EMPTY in a PATTERN statement) you should not change the outline color from the default value, SAME, to a single color. Otherwise all the outlines will be one color and you will not be able to distinguish between the empty areas. △

COUTLINE= is not valid when SHAPE=CYLINDER.

See also: “Controlling Bar Chart Patterns, Colors, and Images” on page 816 and “About Patterns” on page 784

Featured in: Example 3 on page 846, Example 5 on page 850 and Example 6 on page 854

CPERCENT

CPCT

displays the cumulative percentage statistic in the table of statistics and above vertical bars. Default statistics are suppressed when you request specific statistics. For vertical bar charts, this option is ignored if the bars are too narrow to avoid overlapping values or if the FREQ, CFREQ, or PERCENT option is specified.

See also: “About Chart Statistics” on page 782, “Displaying Statistics in Horizontal Bar Charts” on page 815, and “Displaying Statistics in Vertical Bar Charts” on page 815

CREF=reference-line-color | (reference-line-color) | reference-line-color-list

CR=reference-line-color | (reference-line-color) | reference-line-color-list

specifies colors for reference lines. Specifying a single color without parentheses applies that color to all reference lines, including lines drawn with the AUTOREF and REF= options. Note that the CAUTOREF= option overrides CREF= reference color for reference lines drawn with the AUTOREF option. Specifying a single color in parentheses applies that color only to the first reference line drawn with the REF= option. Specifying a reference color list applies colors in sequence to successive lines drawn with the REF= option. The syntax of the color list is of the form (*color1 color2 ...colorN*) or (*color1, color2 ..., colorN*). The default color for reference lines is either the value of the CAXIS= option or the first color in the color list. To specify line types for these reference lines, use the LREF= option.

CPERCENTLABEL='column-label' | NONE (HBAR and HBAR3D only)

specifies the text of the column label for the CPERCENT statistic in the table of statistics. *Column-label* can be up to 32 characters long, but a single line of the label can be no more than 24 characters. By default, a label with more than one word will break as close to the center of the line as possible. A double space in the string forces a line break. To suppress the label, specify CPERCENTLABEL=NONE.

Not supported by: Java, ActiveX

CTEXT=text-color

specifies the color of all text on the chart that is not otherwise assigned a color. Text includes axis values and axis labels in the response, midpoint, and group axes; the subgroup legend; and the displayed statistics. For the Java and ActiveX devices, the default color is black. For other devices, if you omit CTEXT=, PROC GCHART searches for a color specification in this order:

- 1 the CTEXT= option in a GOPTIONS statement

2 the first color in the colors list (the default).

CTEXT= overrides the color specification for the axis label and the tick mark values in the COLOR= option in an AXIS definition assigned to an axis.

CTEXT= is overridden by

- the COLOR= suboption of the LABEL= or VALUE= option in a LEGEND definition assigned to the subgroup legend. In this case the suboption determines the color of the legend label or the color of the legend value descriptions, respectively.
- the COLOR= suboption of a LABEL= or VALUE= option in an AXIS definition assigned to an axis. In this case the suboption determines the color of the axis label or the color of the tick mark values, respectively.

DESCENDING

arranges the bars in descending order of the value of the chart statistic. By default, bars are arranged in ascending order of midpoint value, without regard to the lengths of the bars. DESCENDING reorders the bars from longest to shortest. In horizontal bar charts the ordering is top to bottom; in vertical bar charts the ordering is left to right. If you also use the GROUP= option, the reordering is performed separately for each group, so the order of the midpoints may be different for each group.

DESCENDING overrides any midpoint order that is specified with the MIDPOINTS= option or that is specified in the ORDER= option in an AXIS statement assigned to the midpoint axis.

DESCRIPTION=*'entry-description'*

DES=*'entry-description'*

specifies the description of the catalog entry for the chart. The maximum length for *entry-description* is 256 characters. The description does not appear on the chart. By default, the GCHART procedure assigns a description of the form HBAR CHART OF *variable*, where *variable* is the name of the chart variable.

The *entry-description* can include the #BYLINE, #BYVAL, and #BYVAR substitution options, which work as they do when used on TITLE, FOOTNOTE, and NOTE statements. Refer to the description of the options on page 222, and “Substituting BY Line Values in a Text String” on page 226. The 256-character limit applies before the substitution takes place for these options; thus, if in the SAS program the *entry-description* text exceeds 256 characters, it is truncated to 256 characters, and then the substitution is performed.

The descriptive text is shown in each of the following:

- the "description" portion of the Results window
- the catalog-entry properties that you can view from the Explorer window
- the Table of Contents that is generated when you use CONTENTS= on an ODS statement (see “Linking to Output through a Table of Contents” on page 495), assuming the GCHART output is generated while the contents page is open
- the Description field of the PROC GREPLAY window
- the data tip text for web output (depending on the device driver you are using). See “Adding Data Tips to Web Presentations” on page 568 for details.

Featured in: Example 7 on page 856

DISCRETE

treats a numeric chart variable as a discrete variable rather than as a continuous variable. The GCHART procedure creates a separate midpoint and, hence, a separate bar for each unique value of the chart variable. If the chart variable has a format associated with it, each formatted value is treated as a midpoint.

The LEVELS= option is ignored when you use DISCRETE. The MIDPOINTS= option overrides DISCRETE. The ORDER= option in an AXIS statement that is assigned to the midpoint axis can rearrange or exclude discrete midpoint values.

Featured in: Example 7 on page 856

ERRORBAR=BARS | BOTH | TOP

draws confidence intervals on a horizontal or vertical bar chart for either of the following:

- the mean of the SUMVAR= variable for each midpoint if you specify TYPE=MEAN
- the percentage of observations assigned to each midpoint if you specify TYPE=PCT with no SUMVAR= option.

The ERRORBAR= option cannot be used with values of the TYPE= option other than MEAN or PCT. Valid values for ERRORBAR= are:

BARS

draws error bars as bars half the width of the main bars.

BOTH

draws error bars as two ticks joined by a line (default).

TOP

draws the error bar as a tick for the upper confidence limit that is joined to the top of the bar by a line.

By default, ERRORBAR= uses a confidence level of 95 percent. You can specify different confidence levels with the CLM= option.

When you use ERRORBAR= with TYPE=PCT, the confidence interval is based on a normal approximation. Let TOTAL be the total number of observations, and PCT be the percentage assigned to a given midpoint. The standard error of the percentage is approximated as

$$APSTDERR=100 * \text{SQRT}((PCT/100) * (1-(PCT/100)) / TOTAL);$$

Let LEVEL be the confidence level specified using the CLM= option, with a default value of 95. The upper confidence limit for the percentage is computed as

$$UCLP = PCT + APSTDERR * \text{PROBIT}(1-(1-LEVEL/100)/2);$$

The lower confidence limit for the percentage is computed as

$$LCLP = PCT - APSTDERR * \text{PROBIT}(1-(1-LEVEL/100)/2);$$

When you use ERRORBAR= with TYPE=MEAN, the sum variable must have at least two non-missing values for each midpoint. If the GROUP= option is used, each midpoint within a group must also have two non-missing values. Let N be the number of observations assigned to a midpoint, MEAN be the mean of those observations, and STD be the standard deviation of the observations. The standard error of the mean is computed as

$$STDERR = STD / \text{SQRT}(N);$$

Let LEVEL be the confidence level specified using the CLM= option, with a default value of 95. The upper confidence limit for the mean is computed as

$$UCLM = MEAN + STDERR * \text{TINV}(1-(1-LEVEL/100)/2, N-1);$$

The lower confidence limit for the mean is computed as

$$LCLM = MEAN - STDERR * \text{TINV}(1-(1-LEVEL/100)/2, N-1);$$

If you want the error bars to represent a given number C of standard errors instead of a confidence interval, and if the number of observations assigned to each

midpoint is the same, then you can find the appropriate value for the CLM= option by running a DATA step. For example, if you want error bars that represent one standard error (C=1) with a sample size of N, you can run the following DATA step to compute the appropriate value for the CLM= option and assign that value to a macro variable &LEVEL:

```
data null;
  c = 1;
  n = 10;
  level = 100 * (1 - 2 * (1 - probt( c, n-1)));
  put all;
  call symput('level',put(level,best12.));
run;
```

Then when you run the GCHART procedure, you can specify CLM=&LEVEL.

Note that this trick does not work precisely if different midpoints have different numbers of observations. However, choosing an average value for N may yield sufficiently accurate results for graphical purposes if the sample sizes are large or do not vary much.

Featured in: Example 6 on page 854

FRAME | NOFRAME

FR | NOFR

specifies whether the 2D axis area frame or the 3D backplane is drawn. The default is FRAME, which draws a frame around the axis area (in 2D bar charts) or generates a colored 3D backplane (in 3D bar charts). Specifying NOFRAME removes the axis area frame from 2D charts, including any background color or image. For 3D charts, NOFRAME removes the backplane color or image, and leaves the vertical and horizontal axis planes and axes. To remove these planes, use the NOPLANE option in the AXIS statement. To remove one or more axis elements, use either the AXIS statement or the NOAXIS option.

The NOFRAME option overrides the CFRAME= and IFRAME= options and the IBACK= option “IBACK” on page 317.

The color of the frame or backplane outline is the color of the midpoint axis, which is determined by the CAXIS= option.

If the V6COMP graphics option is in effect, the default is NOFRAME.

Featured in: Example 7 on page 856 and Example 6 on page 854

FREQ

displays the frequency statistic in the table of statistics and above vertical bars. Non-integer values are rounded down to the nearest integer. Default statistics are suppressed when you request specific statistics. For vertical bar charts, this option is ignored if the bars are too narrow to avoid overlapping values. This option overrides the CFREQ, PERCENT, CPERCENT, SUM, and MEAN options.

Featured in: Example 5 on page 850

See also: “About Chart Statistics” on page 782, “Displaying Statistics in Horizontal Bar Charts” on page 815, and “Displaying Statistics in Vertical Bar Charts” on page 815

FREQLABEL=*'column-label'* | NONE (HBAR and HBAR3D only)

specifies the text of the column label for the FREQ statistic in the table of statistics. *column-label* can be up to 32 characters long, but a single line of the label can be no more than 24 characters. By default, a label with more than one word will break as close to the center of the line as possible. A double space in the string forces a line break. To suppress the label, specify FREQLABEL=NONE.

Featured in: Example 5 on page 850 and Example 6 on page 854

Not supported by: Java, ActiveX

FREQ=*numeric-variable*

specifies a variable whose values weight the contribution of each observation in the computation of the chart statistic. Each observation is counted the number of times that is specified by the value of *numeric-variable* for that observation. If the value of *numeric-variable* is missing, 0, or negative, the observation is not used in the statistic calculation. Non-integer values of *numeric-variable* are truncated to integers. FREQ= is valid with all chart statistics.

Because you cannot use TYPE=PERCENT, TYPE=CPERCENT, TYPE=FREQ, or TYPE=CFREQ with the SUMVAR= option, you must use FREQ= to calculate percentages, cumulative percentages, frequencies, or cumulative frequencies based on a sum.

The statistics will not be affected by applying a format to *numeric-variable*.

See also: “Calculating Weighted Statistics” on page 783

FRONTREF

specifies that reference lines drawn by the AUTOREF or REF= options should be drawn in front of the bars. By default, reference lines in 3D bar charts are drawn on the back plane of the axis.

G100

calculates the percentage and cumulative percentage statistics separately for each group. When you use G100, the individual percentages reflect the contribution of the midpoint to the group and total 100 percent for each group. G100 is ignored unless you also use the GROUP= option.

By default, the individual percentages reflect the contribution of the midpoint to the entire chart and total 100 percent for the entire chart.

GAXIS=AXIS<1...99>

assigns the specified AXIS definition to the group axis. (A group axis is created when you use the GROUP= option.) You can use the AXIS definition to modify the order of the groups, the text of the labels, and appearance of the axis. GAXIS= is ignored if the specified AXIS definition does not exist.

The AXIS statement options MAJOR= and MINOR= are ignored in AXIS definitions assigned to the group axis because the axis does not use tick marks. A warning message is written to the SAS log if these options appear in the AXIS definition.

The Java and ActiveX devices do not support all AXIS statement options. See “AXIS Statement” on page 124 for more information.

To remove groups from the chart, use the ORDER= option in the AXIS statement.

To suppress the brackets drawn around the values on the group axis in vertical bar charts, use the NOBRACKETS option in the AXIS statement.

See also: “AXIS Statement” on page 124

Featured in: Example 7 on page 856

Not supported by: Java (partial), ActiveX (partial)

GROUP=*group-variable*

organizes the data according to values of *group-variable*. *Group-variable* can be either character or numeric and is always treated as a discrete variable.

GROUP= produces a separate group of bars for each unique value of the group variable. Missing values for *group-variable* are treated as a valid group. The groups are arranged in ascending order of the group variable values.

By default, each group includes all midpoints, even if no observations for the group fall within the midpoint range, meaning that no bar is drawn at the midpoint. Use the NOZERO option to suppress midpoints with no observations.

GROUP= also produces a *group axis* that lists the values that distinguish the groups. The group axis has no axis line but displays the group variable name or label. To modify the group axis, assign an AXIS definition with the GAXIS= option.

In horizontal bar charts, the group axis is to the left of the midpoint axis and the groups are arranged from top to bottom, starting with the lowest value at the top.

In vertical bar charts, the group axis is below the midpoint axis and the groups are arranged from left to right starting with the lowest value at the left. If the group label in a vertical bar chart is narrower than all the bars in the group, brackets are added to the label to emphasize which bars belong in each group. Group brackets are not displayed if the space between the group values is less than one and one-half character cells. Use the NOBRACKETS option in the AXIS statement to suppress the group brackets.

Featured in: Example 7 on page 856

GSPACE=group-spacing

specifies the amount of extra space between groups of bars. *Group-space* can be any non-negative number. Units are character cells. Use GSPACE=0 to leave no extra space between adjacent groups of bars. In this case, the same space appears between groups of bars as between the bars in the same group.

GSPACE= is ignored unless you also use the GROUP= option. By default, the GCHART procedure calculates group spacing based on size of the axis area and the number of bars in the chart.

If the requested spacing results in a chart that is too large to fit in the space available for the midpoint axis, an error message is written to the SAS log and no chart is produced.

Featured in: Example 7 on page 856

HTML=variable

identifies the variable in the input data set whose values create links in the HTML file created by the ODS statement. These links are associated with an area of the chart and point to the data or graph you wish to display when the user drills down on the area. The values of *variable* can be up to 1024 characters long. Characters after the 1024-character limit (including any closing quotes) are truncated.

HTML_LEGEND=variable

identifies the variable in the input data set whose values create links in the HTML file created by the ODS statement. These links are associated with a legend value and point to the data or graph you wish to display when the user drills down on the value. The values of *variable* can be up to 1024 characters long. Characters after the 1024-character limit (including any closing quotes) are truncated.

Not supported by: Java, ActiveX

IFRAME=fileref | 'external-file'

identifies the image file you wish to fill the backplane frame of your three-dimensional bar charts. See also the IMAGESTYLE= option and “Placing a Backplane Image on Graphs with Frames” on page 115.

This option is overridden by the NOIMAGEPRINT goption “IMAGEPRINT” on page 318 .

To fill the backplane frame of two-dimensional bar charts, see the IBACK= goption “IBACK” on page 317 .

Not supported by: Java

IMAGESTYLE= TILE | FIT

for three-dimensional bar charts, specifies whether to use multiple instances of an image to fill the backplane frame (TILE) or to stretch a single instance of an image to fill the backplane frame (FIT). The TILE value is the default. See also the IFRAME= option. Java supports only TILE.

Not supported by: Java (partial)

INSIDE=statistic

displays the values of the specified statistic inside the bars. For the Java and ActiveX devices, this option is valid for both horizontal and vertical bar charts. For other devices, this option is only valid for vertical bar charts.

Statistic can be one of the following:

- FREQ**
- CFREQ**
- CPERCENT | CPCT**
- MEAN**
- PERCENT | PCT**
- SUM**

If the bars are subgrouped, only the following statistics are valid:

- FREQ**
- PERCENT | PCT**
- SUBPCT**
- SUM**

With subgroups, PERCENT displays the percent contribution of each subgroup to the midpoint value of the bar, based on frequency. The PERCENT values for each subgroup total the percent contribution of the bar to the whole. For example, if the percent contribution of the whole bar is 60%, the PERCENT statistic for all the subgroups in that bar will total 60%. To calculate PERCENT based on the SUMVAR= variable, use the FREQ= and TYPE= options. For details, see “Calculating Weighted Statistics” on page 783.

SUBPCT displays the percent contribution of each subgroup to the total bar. The SUBPCT values for each subgroup total the percent contribution to the whole bar. Because of rounding, the total of the percents may not equal 100.

Featured in: Example 4 on page 848 Example 7 on page 856

See also: “About Chart Statistics” on page 782, “Displaying Statistics in Horizontal Bar Charts” on page 815, and “Displaying Statistics in Vertical Bar Charts” on page 815

LAUTOREF=reference-line-type

specifies the line type for reference lines at major tick marks, as determined by the AUTOREF option. Line types are specified as whole numbers from 1 to 46, with 1 representing a solid line and the other values representing dashed lines. The default value is 1. To specify a color for these reference lines, use the CAUTOREF= option.

LEGEND=LEGEND<1...99>

assigns the specified LEGEND definition to the legend generated by the SUBGROUP= option. The LEGEND= option itself does *not* generate a legend.

LEGEND= is ignored if

- SUBGROUP= is not used.
- the specified LEGEND definition is not in effect.
- the NOLEGEND option is used.
- the PATTERNID= option is set to any value other than SUBGROUP; that is, the value of PATTERNID= is BY or GROUP or MIDPOINT.

To create a legend based on the chart midpoints instead of the subgroups, use the chart variable as the subgroup variable:

```
hbar city / subgroup=city;
```

The Java and ActiveX devices do not support all LEGEND statement options. See “LEGEND Statement” on page 151 for more information.

See also: “LEGEND Statement” on page 151 and SUBGROUP= on page 813 option

Featured in: Example 4 on page 848

LEVELS=*number-of-midpoints* | ALL

specifies the number of midpoints for a numeric chart variable. The range for each midpoint is calculated automatically, using the algorithm in Terrell and Scott (1985).

If you specify LEVELS=ALL, then all unique midpoint values are graphed. If your data contains a large number of unique midpoint values (over 200), you can use the XPIXELS and YPIXELS GOPTIONS to allow the device driver to render a larger (and more readable) graph.

LEVELS= is ignored if

- the chart variable is character type
- the DISCRETE option is used
- the MIDPOINTS= option is used.

LREF=*reference-line-type* | (*reference-line-type* | *reference-line-type-list*)

LR=*reference-line-type* | (*reference-line-type* | *reference-line-type-list*)

specifies line types for reference lines. Line types are specified as whole numbers from 1 to 46, with 1 representing a solid line and the other values representing dashed lines. Specifying a line type without parentheses applies that type to all reference lines drawn with the AUTOREF and REF= options. Note that the LAUTOREF= option overrides LREF=*reference-line-type* for reference lines drawn with the AUTOREF option. Specifying a single line type in parentheses applies that line type to the first reference line drawn with the REF= option. Specifying a line type list applies line types in sequence to successive reference lines drawn with the REF= option. The syntax of the line-type list is of the form (*type1 type2 ...typeN*). The default line type is specified by the AXIS statement’s STYLE= option. By default, STYLE=1, a solid line. To specify colors for these reference lines, use the CREF= option.

Not supported by: Java

MAXIS=AXIS<1...99>

assigns the specified AXIS definition to the midpoint axis. The MAXIS= option is ignored if the specified AXIS definition does not exist.

The Java and ActiveX devices do not support all AXIS statement options. See “AXIS Statement” on page 124 for more information.

See also: “AXIS Statement” on page 124 and “About Midpoints” on page 780

Featured in: Example 4 on page 848

Not supported by: Java (partial), ActiveX (partial)

MEAN

displays the mean statistic in the table of statistics and above vertical bars. By default, the column heading in the table includes the name of the variable for which the mean is calculated. Default statistics are suppressed when you request specific statistics. For vertical bar charts, this option is ignored if the bars are too narrow to avoid overlapping values or if the FREQ, CFREQ, PERCENT, CPERCENT, or SUM option is specified. MEAN is ignored unless you also use the SUMVAR= option.

See also: “About Chart Statistics” on page 782, “Displaying Statistics in Horizontal Bar Charts” on page 815, and “Displaying Statistics in Vertical Bar Charts” on page 815

MEANLABEL=*'column-label'* | NONE (HBAR and HBAR3D only)

specifies the text of the column label for the MEAN statistic in the table of statistics. *column-label* can be up to 32 characters long, but a single line of the label can be no more than 24 characters. By default, a label with more than one word will break as close to the center of the line as possible. A double space in the string forces a line break. To suppress the label, specify MEANLABEL=NONE.

Featured in: Example 6 on page 854

Not supported by: Java, ActiveX

MIDPOINTS=*value-list*

specifies the midpoint values for the bars. The way you specify *value-list* depends on the type of the chart variable.

- For numeric chart variables, *value-list* is either an explicit list of values, or a starting and an ending value with an interval increment, or a combination of both forms:

n <...*n*>

n TO *n* <BY *increment*>

n<...*n*> TO *n* <BY *increment*> <*n* <...*n*>>

If a numeric variable has an associated format, the specified values must be the *unformatted* values.

By default, numeric variable values are treated as continuous (if you omit the DISCRETE option), and

- the lowest midpoint consolidates all data points from negative infinity to the median of the first two midpoints
- the highest midpoint consolidates all data points from the median of the last two midpoints up to infinity
- all other values in *value-list* specify the median of a range of values, and the GCHART procedure calculates the midpoint values.

If you include the DISCRETE option, each value in *value-list* specifies a unique numeric value.

- For character chart variables, *value-list* is a list of unique character values enclosed in quotation marks and separated by blanks:

'*value-1*' <...'value-*n*'>

If a character variable has an associated format, the specified values must be the *formatted* values.

For a complete description of *value-list*, see the ORDER= on page 130 option in the AXIS statement.

If the *value-list* for either type of variable specifies so many midpoints that the axis values overwrite each other, the values may be unreadable. In this case the procedure writes a warning to the SAS log. On many devices, this problem can be corrected by either adjusting the size of the text with the HTEXT= graphics option or by increasing the number of cells in your graphics display using the HPOS= and VPOS= graphics options.

The ORDER= option in the AXIS statement overrides the order specified in the MIDPOINTS= option. The bar chart statement options ASCENDING and DESCENDING also override both MIDPOINTS= and ORDER= in the AXIS statement.

See also: “About Midpoints” on page 780

Featured in: Example 5 on page 850

MIDPOINTS=OLD

generates default midpoints using the Nelder algorithm (*Applied Statistics* 25:94–7, 1976). The MIDPOINTS=OLD option is ignored unless the chart variable is numeric.

MINOR=number-of-minor-ticks

specifies the number of minor tick marks between each major tick mark on the response axis.

MINOR= in a bar chart statement overrides the number of minor tick marks specified in the MINOR= option in an AXIS definition assigned to the response axis with the RAXIS= option.

MISSING

accepts a missing value as a valid midpoint for the chart variable. By default, observations with missing values are ignored. Missing values are always valid for group and subgroup variables.

NAME='entry-name'

specifies the name of the catalog entry for the graph. The maximum length for *entry-name* is eight characters. The default name is GCHART. If the name duplicates an existing entry name, then SAS/GRAPH software adds a number to the duplicate name to create a unique name—for example, GCHART1.

Featured in: Example 7 on page 856

NOAXIS

suppresses all axes, including axis lines, axis labels, axis values, and all major and minor tick marks. If you specify an axis definition with the GAXIS, MAXIS=, or RAXIS= options, then the axes are generated as defined in the AXIS statement, but then all lines, labels, values, and tick marks are suppressed. Therefore, axis statement options such as ORDER=, LENGTH, and OFFSET= will still be used.

To remove only selected axis elements such as lines, values or labels, use specific AXIS statement options.

NOAXIS does not suppress either the default frame or an axis area fill requested by the CFRAME= option. To remove the axis frame or the 3D backplane, use the NOFRAME option in the procedure. To remove the horizontal or vertical axis planes, use the NOPLANE option in the AXIS statement.

NOBASEREF

suppresses the zero reference line when the SUM or MEAN chart statistic has negative values.

NOLEGEND

suppresses the legend that is automatically generated by the SUBGROUP= option. NOLEGEND is ignored if the SUBGROUP= option is not used.

NOSTATS (HBAR and HBAR3D only)

suppresses the table of statistics. NOSTATS suppresses both the default statistics and specific statistics requested by the FREQ, CFREQ, PERCENT, CPERCENT, SUM, and MEAN options.

Not supported by: Java

NOZERO

suppresses any midpoints for which there are no corresponding values of the chart variable and, hence, no bar. NOZERO usually is used with the GROUP= option to suppress midpoints when not all values of the chart variable are present for every group or if the chart statistic for the bar is 0.

Note: If a bar is omitted and if you have also specified bar labels with the VALUE= option in an AXIS statement, the labels may be shifted and not displayed with the correct bar. \triangle

Featured in: Example 7 on page 856

Not supported by: Java

OUTSIDE=statistic

displays the values of the specified statistic above the bars. For the Java and ActiveX devices, this option is valid for both horizontal and vertical bar charts. For other devices, this option is only valid for vertical bar charts.

Statistic can be one of the following:

- FREQ
- CFREQ
- PERCENT | PCT
- CPERCENT | CPCT
- SUM
- MEAN

Featured in: Example 4 on page 848 and Example 7 on page 856

See also: “About Chart Statistics” on page 782, “Displaying Statistics in Horizontal Bar Charts” on page 815, and “Displaying Statistics in Vertical Bar Charts” on page 815

PATTERNID=BY | GROUP | MIDPOINT | SUBGROUP

specifies the way fill patterns are assigned. By default, PATTERNID=SUBGROUP. Values for PATTERNID= are as follows:

BY

changes patterns each time the value of the BY variable changes. All bars use the same pattern if the GCHART procedure does not include a BY statement.

GROUP

changes patterns every time the value of the group variable changes. All bars in each group use the same pattern, but a different pattern is used for each group.

MIDPOINT

changes patterns every time the midpoint value changes. If you use the GROUP= option, the respective midpoint patterns are repeated for each group.

SUBGROUP

changes patterns every time the value of the subgroup variable changes. The bars must be subdivided by the SUBGROUP= option for the SUBGROUP value to have an effect. Without SUBGROUP=, all bars have the same pattern.

Note: If you use the SUBGROUP= option and specify a PATTERNID= value other than SUBGROUP, the bar segments use the same pattern and are indistinguishable. \triangle

See also: “Controlling Bar Chart Patterns, Colors, and Images” on page 816

Featured in: Example 4 on page 848 and Example 7 on page 856

PERCENT

PCT

prints the percentages of observations having a given value for the chart variable in the table of statistics and above vertical bars. Default statistics are suppressed when you request specific statistics. For vertical bar charts, this option is ignored if the bars are too narrow to avoid overlapping values or if the FREQ or CFREQ option is specified.

See also: “About Chart Statistics” on page 782, “Displaying Statistics in Horizontal Bar Charts” on page 815, and “Displaying Statistics in Vertical Bar Charts” on page 815

PERCENTLABEL=*column-label* | NONE (HBAR and HBAR3D only)

specifies the text of the column label for the PERCENT statistic in the table of statistics. *column-label* can be up to 32 characters long, but a single line of the label can be no more than 24 characters. By default, a label with more than one word will break as close to the center of the line as possible. A double space in the string forces a line break. To suppress the label, specify PERCENTLABEL=NONE.

Not supported by: Java, ActiveX

RANGE

displays on the axis of the chart the range of numeric values represented by each bar. In the graphics output, the starting value of each range is indicated with the less-than symbol (<), and the ending value is indicated with the greater-than-or-equal-to symbol (>=). The RANGE option has no effect on axes that represent character data. By default, the values shown on the axis are determined by the value of the MIDPOINTS= option on page 809. If specified, the DISCRETE option on page 802 overrides the RANGE option.

RAXIS=*value-list* | AXIS<1...99>**AXIS=*value-list* | AXIS<1...99>**

specifies values for the major tick mark divisions on the response axis or assigns the specified AXIS definition to the axis. See the MIDPOINTS= option on page 809 for a description of *value-list*. By default, the GCHART procedure scales the response axis automatically and provides an appropriate number of tick marks.

You can specify negative values, but negative values are reasonable only when TYPE=SUM or TYPE=MEAN and one or more of the sums or means are less than 0. Frequency and percentage values are never less than 0.

For lists of values, a separate major tick mark is created for each individual value. A warning message is written to the SAS log if the values are not evenly spaced.

If the values represented by the bars are larger than the highest tick mark value, the bars are truncated at the highest tick mark.

If you use a BY statement with the PROC GCHART statement, the same response axes are produced for each BY group when RAXIS=*value-list* is used or if there is an ORDER= list in the AXIS statement assigned to the response axis.

The Java and ActiveX devices do not support all AXIS statement options. See “AXIS Statement” on page 124 for more information.

See also: “AXIS Statement” on page 124

Featured in: Example 4 on page 848 and Example 7 on page 856

Not supported by: Java (partial), ActiveX (partial)

REF=*value-list*

draws reference lines at the specified points on the response axis. See the MIDPOINTS= option on page 809 for a description of *value-list*.

Values can be listed in any order, but should be within the range of values represented by the response axis. A warning is written to the SAS log if any of the points are off of the axis, and no reference line is drawn for such points. You can use the AUTOREF option to draw reference lines automatically at all of the major tick marks.

By default, reference lines in 3D bar charts are drawn on the back plane of the axis. To draw the reference lines in front of the bars, use the FRONTREF option.

SHAPE=*3D-bar-shape* (HBAR3D and VBAR3D only)

specifies the shape of the bars in charts that are produced with the HBAR3D and VBAR3D statements. *3D-bar-shape* can be one of the following:

- BLOCK | B (the default)
- CYLINDER | C

- HEXAGON | H
- PRISM | P
- STAR | S

The COUTLINE= option is not valid when SHAPE=CYLINDER.

Featured in: Example 7 on page 856

SPACE=bar-spacing

specifies the amount of space between individual bars or between the bars within each group if you also use the GROUP= option. *Bar-space* can be any non-negative number, including decimal values. Units are character cells. By default, the GCHART procedure calculates spacing based on the size of the axis area and the number of bars on the chart. Use SPACE=0 to leave no space between adjacent bars.

SPACE= is ignored if

- you specify the WIDTH= option and are using the Java or ActiveX devices.
- the specified spacing requests a chart that is too large to fit in the space available for the midpoint axis. In this case, a warning message is issued.

Featured in: Example 4 on page 848 and Example 7 on page 856

SUBGROUP=subgroup-variable

divides the bars into segments according to the values of *subgroup-variable*. *Subgroup-variable* can be either character or numeric and is always treated as a discrete variable. SUBGROUP= creates a separate segment within each bar for every unique value of the subgroup variable for that midpoint.

If PATTERNID=SUBGROUP (the default setting), each segment is filled with a different pattern and a legend that provides a key to the patterns is automatically generated. If the value of PATTERNID= is anything other than SUBGROUP, the segments are all the same color, the legend is suppressed, and the subgrouping effect is lost.

By default the legend appears at the bottom of the chart. To modify the legend, assign a LEGEND definition with the LEGEND= option. To suppress the legend, specify NOLEGEND.

See also: “LEGEND Statement” on page 151

Featured in: Example 4 on page 848, Example 7 on page 856 and Example 5 on page 850

SUM

displays the sum statistic in the table of statistics and above vertical bars. By default, the column heading in the table includes the name of the variable for which the sum is calculated. Default statistics are suppressed when you request specific statistics. For vertical bar charts, this option is ignored if the bars are too narrow to avoid overlapping values or if the FREQ, CFREQ, PERCENT, or CPERCENT option is specified. SUM is ignored unless you also use the SUMVAR= option.

See also: “About Chart Statistics” on page 782, “Displaying Statistics in Horizontal Bar Charts” on page 815, and “Displaying Statistics in Vertical Bar Charts” on page 815

SUMLABEL='column-label' | NONE (HBAR and HBAR3D only)

specifies the text of the column label for the SUM statistic in the table of statistics. *Column-label* can be up to 32 characters long, but a single line of the label can be no more than 24 characters. By default, a label with more than one word will break as close to the center of the line as possible. A double space in the string forces a line break. To suppress the label, specify SUMLABEL=NONE.

Not supported by: Java, ActiveX

SUMVAR=summary-variable

specifies a numeric variable for sum or mean calculations. The GCHART procedure calculates the sum or, if requested, the mean of *summary-variable* for each midpoint. The resulting statistics are represented by the length of the bars along the response axis, and they are displayed at major tick marks.

When you use SUMVAR=, the TYPE= option must be either SUM or MEAN. With SUMVAR=, the default is TYPE=SUM.

Featured in: Example 3 on page 846 and Example 6 on page 854

TYPE=statistic

specifies the chart statistic.

- If the SUMVAR= option is not used, *statistic* can be one of the following:

FREQ
frequency (the default)

CFREQ
cumulative frequency

PERCENT PCT
percentage

CPERCENT CPCT
cumulative percentage

- If SUMVAR= is used, *statistic* can be:

SUM
sum (the default)

MEAN
mean

Because you cannot use TYPE=FREQ, TYPE=CFREQ, TYPE=PERCENT, or TYPE=CPERCENT with the SUMVAR= option, you must use FREQ= to calculate percentages, cumulative percentages, frequencies, or cumulative frequencies based on a sum. See also “Calculating Weighted Statistics” on page 783.

If you specify TYPE=MEAN and use the SUBGROUP= option, the height or length of the bar represents the mean for the entire midpoint. The subgroup segments are proportional to the subgroup’s contribution to the sum for the bar. See also SUBGROUP= on page 813.

See also: “About Chart Statistics” on page 782 for a complete description of statistic types

Featured in: Example 6 on page 854

WIDTH=bar-width

specifies the width of the bars. By default, the GCHART procedure selects a bar width that accommodates the midpoint values displayed on the midpoint axis using a hardware font and a height of one cell. Units for *bar-width* are character cells. The value for *bar-width* must be greater than 0, but it does not have to be an integer, for example,

```
vbar site / width=1.5;
```

If the requested bar width results in a chart that is too large to fit in the space available for the midpoint axis, the procedure issues a warning in the log and ignores the WIDTH= specification. If the specified width is too narrow, the procedure may display the midpoint values vertically.

Featured in: Example 4 on page 848

WOULINE=bar-outline-width

specifies the width of the outline in pixels. WOUTLINE= affects both the bar and the subgroup outlines.

Not supported by: Java

The Chart Statistic and the Response Axis

In bar charts, the scale of values of the chart statistic is displayed on the response axis. By default, the response axis is divided into evenly spaced intervals identified with major tick marks that are labeled with the corresponding statistic value. Minor tick marks are evenly distributed between the major tick marks unless a log axis has been requested. For sum and mean statistics, the major tick marks are labeled with values of the SUMVAR= variable (formatted if the variable has an associated format). The response axis is also labeled with the statistic type.

Specifying Logarithmic Axes

Logarithmic axes can be specified with the AXIS statement. See “AXIS Statement” on page 124 for a complete discussion.

Displaying Statistics in Horizontal Bar Charts

For graphs generated with the Java and ActiveX devices, default statistics are not generated, but you can display one statistic at the end of each bar. To specify the statistic, specify the FREQ, CFREQ, PERCENT, CPERCENT, SUM, or MEAN option.

For graphs generated with other devices, the HBAR and HBAR3D statements print a table of statistic values to the right of the bars. When the value of TYPE= is FREQ, CFREQ, PERCENT, or CPERCENT, the frequency, cumulative frequency, percentage, and cumulative percentage statistics are printed next to the bars by default. When TYPE=SUM, the frequency and sum statistic values are printed by default. When TYPE=MEAN, the frequency and mean statistic values are printed by default. However, if you use the FREQ, CFREQ, PERCENT, CPERCENT, SUM, or MEAN options to request specific statistics, the default statistics are not printed.

For sum and mean, the name of the SUMVAR= variable is added to the heading for the column of values.

Specifying the Table of Statistics

You can use the FREQ, CFREQ, PERCENT, CPERCENT, SUM, and MEAN options to select only certain statistics. Without the SUMVAR= option, only the frequency, cumulative frequency, percentage, and cumulative percentage statistics can be printed. With SUMVAR=, all statistics, including the sum and mean, can be printed. You can suppress all statistics with the NOSTATS option.

To change the column labels for any statistic in the table, use one or more of the statistic column label options: FREQLABEL=, CFREQLABEL=, PERCENTLABEL=, CPERCENTLABEL=, SUMLABEL=, and MEANLABEL=.

To control the font and size of the text in the table of statistics, use the HTEXT= and FTEXT= graphics options.

Displaying Statistics in Vertical Bar Charts

Statistic values on vertical bar charts are not printed by default, so you must explicitly request a statistic with the FREQ, CFREQ, PERCENT, CPERCENT, SUM, MEAN, INSIDE=, or OUTSIDE= option.

For graphs generated with the Java and ActiveX devices, you can display one statistic for each bar. For graphs generated with other devices, you can display up to two statistics. Statistics can be displayed either above the bars or inside the bars.

To specify a statistic that you want to display above the bars, specify the statistic option (FREQ, CFREQ, PERCENT, CPERCENT, SUM, or MEAN) or specify `OUTSIDE=statistic`. To specify a statistic that you want to display inside the bars, specify `INSIDE=statistic`.

For graphs generated with the Java and ActiveX devices, the `OUTSIDE=` option overrides `INSIDE=`, and `INSIDE=` overrides the `FREQ`, `CFREQ`, `PERCENT`, `CPERCENT`, `SUM`, and `MEAN` options. For graphs generated with other devices, the individual statistic options override the `OUTSIDE=` option.

If more than one statistic option is specified, only the highest priority statistic is displayed. The priority order, from highest to lowest, is as follows:

- 1 FREQ
- 2 CFREQ
- 3 PERCENT
- 4 CPERCENT
- 5 SUM
- 6 MEAN

The bars must be wide enough to accommodate the text. You can adjust the width of the bars with the `WIDTH=` option. To control the font and size of the text, use the `HTEXT=` and `FTEXT=` graphics options.

Ordering and Selecting Midpoints

To rearrange character or discrete numeric midpoint values or to select ranges for numeric values, use the `MIDPOINTS=` option. Remember that although changing the number of midpoints for numeric variables may change the range of values for individual midpoints, it does not change the range of values for the chart as a whole. For details, see “About Midpoints” on page 780.

Like `MIDPOINTS=`, the `ORDER=` option in the `AXIS` statement can rearrange the order of the midpoints or suppress the display of discrete numeric or character values. However, `ORDER=` cannot calculate the midpoints for a continuous numeric variable, or exclude values from the calculations. For details, see the description of the `ORDER=` on page 130 option.

Controlling Bar Chart Patterns, Colors, and Images

Default Patterns and Outlines

Each bar in a bar chart is filled with a pattern. By default, the procedure

- fills the bars with bar/block patterns, beginning with the default fill, `SOLID`, and rotating it through the colors list. When the solid patterns are exhausted, the procedure selects the next default bar/block pattern and rotates it through the colors list. It continues in this fashion until all of the required patterns have been assigned.

Note: 3D bar charts always uses solid patterns. \triangle

If you use the device’s default colors and the first color in the list is either black or white, the procedure does not create a pattern in that color. If you specify a colors list with the `COLORS=` graphics option, the procedure uses all the colors in the list to generate the patterns.

- outlines bars and bar segments using the first color in the colors list.

See “About Patterns” on page 784 for more information on how the `GCHART` procedure assigns default patterns and outlines.

User-Defined Patterns

To override the default patterns and select fills and colors for the bars or bar segments, use the PATTERN statement. Only bar/block patterns are valid; all other pattern fills are ignored. For a complete description of all bar/block patterns, see VALUE= on page 171 in “PATTERN Statement” on page 169.

Whenever you use PATTERN statements, the default pattern outline color changes to SAME. That is, the outline color is the same as the fill color. To specify the outline color, use the COUTLINE= option (see COUTLINE= on page 800).

When Patterns Change

The PATTERNID= option controls when the pattern changes. By default, PATTERNID=SUBGROUP. Therefore, when you use the SUBGROUP= option to subdivide the bars, the pattern automatically changes each time the subgroup value changes, and each subdivision of the bar displays a different pattern. As a result, the number of values for the SUBGROUP= variable determines the number of bar patterns on the chart. If you do not subdivide the bars, all bars use the same pattern.

Instead of changing the pattern for each subgroup, you can change the pattern for each midpoint, each group, or each BY group by changing the value of PATTERNID=. See the PATTERNID= on page 811 option for details.

Axis Color

By default, axis elements use the first color in the colors list or the colors that are specified by AXIS statement color options. However, action statement options can also control the color of the axis lines, text, and frame.

To change the color of...	Use this option...
the axis text	CTEXT=
the axis lines	CAXIS=
the area within the frame	CFRAME=

Adding Images to Bar Charts

You can apply images to the bars and to the backplane frame of two-dimensional bar charts developed with the HBAR and VBAR statements. In three-dimensional bar charts, you can apply images to the backplane frame. For details, see “Specifying Images in SAS/GRAPH Programs” on page 106.

PIE, PIE3D, and DONUT Statements

Create pie or donut charts in which the size of a pie slice represents the value of the chart statistic for that category of data in relation to the total chart statistic for all categories.

Requirements: At least one chart variable is required.

Global statements: LEGEND, PATTERN, TITLE, FOOTNOTE

Supports: Drill-down functionality

Description

The PIE, PIE3D, and DONUT statements specify the variable or variables that define the categories of data to chart. These statements automatically

- determine the midpoints.
- calculate the chart statistic for each midpoint (the default is FREQ).
- scale each slice to represent its chart statistic. No slice is drawn if the chart statistic for the midpoint is 0.
- order the slices by midpoint value in ascending order starting at the three o'clock position and proceeding counterclockwise around the pie.
- print the slice name (midpoint value) and slice value (chart statistic) beside each slice.
- assign patterns and colors to the slices. The default pie pattern is PSOLID.

You can use statement options to select or order the midpoints (slices), to change the type of chart statistic, and to modify the appearance of the chart, including the content and position of the slice labels, and patterns used by the slices. You can also specify additional variables by which to group, subgroup, or sum the data. Statement options can also produce special effects, such as exploded or invisible slices.

Donut and pie charts allow grouping and subgrouping. Grouping creates two or more separate pie or donut charts that display in rows or columns on one graph. Subgrouping creates a separate ring of slices within the circle for each value of the subgroup variable. The concentric rings of the subgrouped pie or donut chart make it easy to compare slice values between subgroups.

In addition, you can use global statements to modify patterns and legends, as well as add titles, footnotes, and notes to the chart. You can also use an Annotate data set to enhance the chart.

Syntax

PIE | PIE3D | DONUT *chart-variable(s) </option(s)>;*

option(s) can be one or more options from any or all of the following categories:

- appearance options
 - ANNOTATE=*Annotate-data-set*
 - CFILL=*fill-color*
 - COUTLINE=*slice-outline-color* | SAME
 - DETAIL_RADIUS=*percent* (PIE and DONUT only)
 - EXPLODE=*value-list*
 - FILL=SOLID | X
 - INVISIBLE=*value-list*

- NOHEADING
 WOUTLINE=*slice-outline-width*
- statistic options
 - FREQ=*numeric-variable*
 - SUMVAR=*summary-variable*
 - TYPE=*statistic*
 - midpoint options
 - DISCRETE
 - LEVELS=*number-of-midpoints* | ALL
 - MIDPOINTS=*value-list*
 - MIDPOINTS=OLD
 - MISSING
 - OTHER=*percent-of-total*
 - detail pie options (PIE and DONUT only)
 - DETAIL=*variable*
 - DETAIL_THRESHOLD=*percent*
 - grouping and subgrouping options
 - ACROSS=*number-of-columns*
 - DOWN=*number-of-rows*
 - GROUP=*group-variable*
 - NOGROUPHEADING
 - SUBGROUP=*subgroup-variable*
 - slice-ordering options
 - ANGLE=*degrees*
 - ASCENDING
 - CLOCKWISE
 - DESCENDING
 - JSTYLE
 - slice-labeling options
 - CTEXT=*text-color*
 - LEGEND | LEGEND=LEGEND<1...99>
 - MATCHCOLOR
 - NOLEGEND
 - OTHERLABEL=*'text-string'*
 - PERCENT=ARROW | INSIDE | NONE | OUTSIDE
 - SLICE=ARROW | INSIDE | NONE | OUTSIDE
 - VALUE=ARROW | INSIDE | NONE | OUTSIDE
 - detail pie slice-labeling options (PIE and DONUT only)
 - DETAIL_PERCENT=BEST | NONE
 - DETAIL_SLICE=BEST | NONE
 - DETAIL_VALUE=BEST | NONE
 - donut-labeling options (DONUT only):
 - DONUTPCT=*percent*
 - LABEL=(*text argument(s)*)
 - catalog entry description options

DESCRIPTION=*'entry-description'*

NAME=*'entry-name'*

- ODS options

HTML=*variable*

HTML_LEGEND=*variable*

Required Arguments

chart-variable(s)

specifies one or more variables that define the categories of data to chart. Each chart variable draws a separate chart. All variables must be in the input data set.

Separate multiple chart variables with blanks.

See also: “About Chart Variables” on page 779

Options

Options in a PIE, PIE3D, or DONUT statement affect all graphs that are produced by that statement. You can specify as many options as you want and list them in any order. For details on specifying colors, see Chapter 6, “SAS/GRAPH Colors and Images,” on page 91. For a complete description of the graphics options, see Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261.

ACROSS=*number-of-columns*

draws *number-of-columns* pies across the procedure output area. ACROSS is ignored unless you also use the GROUP= option.

If *number-of-columns* calls for more pies than fit horizontally in the graphics output area, no pies are drawn and an error message is written to the SAS log.

If the DOWN= option also is used, the pies are drawn in left-to-right and top-to-bottom order.

Featured in: Example 11 on page 875

ANGLE=*degrees*

starts the first slice at the specified angle. A value of 0 for *degrees* corresponds to the 3 o'clock position. *Degrees* can be either positive or negative. Positive values move the starting position in the counterclockwise direction; negative values move the starting position clockwise. By default, ANGLE=0. Successive slices are drawn counterclockwise from the starting slice.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate charts produced by the PIE, PIE3D, or DONUT statement.

Note: Annotate coordinate systems 1, 2, 7, and 8 (data system coordinates) are not valid with pie or donut charts. △

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

ASCENDING

arranges the slices in ascending order of the value of the chart statistic. By default, slices are arranged in ascending order of midpoint value, without regard to size.

ASCENDING reorders the slices from smallest to largest. The OTHER slice is still last regardless of its size.

If you also use the GROUP= option, the reordering is performed separately for each group, so the order of the midpoint values may be different for each pie or donut.

ASCENDING overrides any midpoint order that is specified with the MIDPOINTS= option.

CFILL=*fill-color*

specifies one color for all patterns in the chart, regardless of whether the fill is solid or hatch. For the PIE3D statement, the fill is always solid. For the PIE and DONUT statements, if no pattern is specified on the PATTERN statement or with the FILL= option, the procedure starts with the default solid fill and then, beginning with P2N0, uses each default pie hatch pattern with the specified color. For the outline color, the procedure uses the foreground color, which is the first color in the colors lists. Use COUTLINE= to specify a different outline color. CFILL= overrides any other pattern color specification and controls the color of all slices.

See also: “Controlling Bar Chart Patterns, Colors, and Images” on page 816 and “About Patterns” on page 784

Featured in: Example 10 on page 873

CLOCKWISE

draws the slices clockwise starting at the 12 o'clock position. Although this position implies ANGLE=90, you can use ANGLE= to specify a different starting angle.

Featured in: Example 11 on page 875

COUTLINE=*slice-outline-color* | SAME

outlines all slices, rings (subgroups), and legend values (if a legend appears) in the specified color. SAME specifies that the outline color of a slice or a slice segment or a legend value is the same as the interior pattern color.

The default outline color depends on the PATTERN statement:

- If you do not specify a PATTERN statement, the default outline color is black for the Java or ActiveX devices. Otherwise, the default outline color is the foreground color (the first color in the colors list).
- If a PATTERN statement or the V6COMP graphics options is specified, the default is COUTLINE=SAME.

Note: If you specify empty patterns (VALUE=PEMPTY in a PATTERN statement), you should not change the outline color from the default value, SAME, to a single color. Otherwise, all of the outlines will be one color and you will not be able to distinguish between the empty areas. Δ

See also: “Controlling Slice Patterns and Colors” on page 831 and “About Patterns” on page 784

Featured in: Example 8 on page 869, Example 9 on page 872 and Example 11 on page 875

CTEXT=*text-color*

specifies the color for all text on the chart that is not otherwise assigned a color. Text includes all slice labels, the chart heading, and group headings if grouping is used. CTEXT= also affects the color of the slice label arrows. See “Selecting and Positioning Slice Labels” on page 830.

For the Java and ActiveX devices, the default color is black. For other devices, if you omit CTEXT=, PROC GCHART searches for a color specification in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the first color in the colors list (the default).

The MATCHCOLOR option overrides the CTEXT= option for slice labels.

Featured in: Example 9 on page 872 and Example 11 on page 875

DESCENDING

arranges the slices in descending order of the value of the chart statistic. By default, slices are arranged in ascending order of midpoint value, without regard to size. DESCENDING reorders the slices from largest to smallest. The OTHER slice is still last, regardless of its size.

If you also use the GROUP= option, the reordering is performed separately for each group, so the order of midpoint values may be different for each pie or donut.

DESCENDING overrides any midpoint order that is specified with the MIDPOINTS= option.

Featured in: Example 11 on page 875

DESCRIPTION=*entry-description*

DES=*entry-description*

specifies the description of the catalog entry for the chart. The maximum length for *entry-description* is 256 characters. The description does not appear on the chart. By default, the GCHART procedure assigns a description of the form PIE (or PIE3D or DONUT) CHART OF *variable*, where *variable* is the name of the chart variable.

The *entry-description* can include the #BYLINE, #BYVAL, and #BYVAR substitution options, which work as they do when used on TITLE, FOOTNOTE, and NOTE statements. Refer to the description of the options on page 222, and “Substituting BY Line Values in a Text String” on page 226. The 256-character limit applies before the substitution takes place for these options; thus, if in the SAS program the entry-description text exceeds 256 characters, it is truncated to 256 characters, and then the substitution is performed.

The descriptive text is shown in each of the following:

- the "description" portion of the Results window
- the catalog-entry properties that you can view from the Explorer window
- the Table of Contents that is generated when you use CONTENTS= on an ODS statement (see “Linking to Output through a Table of Contents” on page 495), assuming the GCHART output is generated while the contents page is open
- the Description field of the PROC GREPLAY window
- the data tip text for web output (depending on the device driver you are using). See “Adding Data Tips to Web Presentations” on page 568 for details.

DETAIL=*variable* (PIE and DONUT only)

produces an inner pie overlay whose slices show the major components that comprise the outer pie’s slice. *Variable* is the variable whose values are used to construct the detail pie. If you specify the DETAIL= option and either GROUP= or SUBGROUP=, then the DETAIL= option is ignored.

DETAIL_PERCENT=BEST|NONE (PIE and DONUT only)

specifies the algorithm to use for displaying the percentage values for the detail pie slices. NONE turns off the display of the percentage values.

DETAIL_RADIUS=*percent* (PIE and DONUT only)

determines the size of the detail pie. *Percent* specifies the percent of the outer pie radius to use as the detail pie radius. The valid range is 25 to 90. The default is 75.

DETAIL_SLICE=BEST|NONE (PIE and DONUT only)

specifies the algorithm to use for displaying the detail variable labels for the inner pie slices. NONE turns off the display of the detail variable labels.

DETAIL_THRESHOLD=*percent* (PIE and DONUT only)

determines if a detail slice is included in the inner pie. Any detail slice comprising *percent* or more percent of the whole pie is included. The valid range for *percent* is 0 to 75. The default is 4.

DETAIL_VALUE=BEST|NONE (PIE and DONUT only)

specifies the algorithm to use for displaying the data values for the detail pie slices. NONE turns off the display of the data values.

DISCRETE

treats a numeric chart variable as a discrete variable rather than as a continuous variable. The GCHART procedure creates a separate midpoint and, hence, a separate slice for each unique value of the chart variable. If the chart variable has a format associated with it, each formatted value is treated as a midpoint.

The LEVELS= option is ignored when you use DISCRETE. The MIDPOINTS= option overrides DISCRETE.

DONUTPCT=*percent* (DONUT only)

specifies the size of the donut hole in percent of the radius of the whole chart. Values of *percent* range from 0 to 99. By default, DONUTPCT=25.

Featured in: Example 9 on page 872

DOWN=*number-of-rows*

draws *number-of-rows* pies vertically in the procedure output area. The DOWN= option is ignored unless you also use the GROUP= option.

If *number-of-rows* calls for more pies than fit vertically in the graphics area of the output device, no pies are drawn and an error message is written to the SAS log.

If you also use the ACROSS= option, the pies are drawn in left-to-right and top-to-bottom order.

EXPLODE=*value-list*

pulls the specified slices slightly out from the rest of the pie for added emphasis. *Value-list* is the list of midpoint values for the slices to be exploded. See the MIDPOINTS= on page 826 option for a description of *value-list*.

The values in the value list must match the existing midpoints exactly, including the case of character midpoints. Any values in the list that do not correspond to existing midpoints are ignored.

When you use EXPLODE=, the radius is reduced to allow room for exploded slices.

EXPLODE= does not work with subgroups.

Featured in: Example 8 on page 869

FILL=SOLID | X

specifies the fill pattern for *all* slices in the chart:

SOLID S

rotates a solid fill through the colors list as many times as necessary. This is the default.

X

rotates a single hatch pattern through the colors list as many times as necessary.

The Java and ActiveX devices and PIE3D do not support FILL=X.

If you use default device colors (the COLORS= option is omitted), the fill skips the first color in the colors list.

FILL= overrides any pattern that is specified in PATTERN statements.

By default, the outline color is the first color in the colors list. If PATTERN statements are used to specify colors, the slice outline color matches the slice fill color.

By default, the fill patterns take the colors from the current colors list in rotation. If any PATTERN statements have been defined, the colors in the PATTERN definitions are used, in order, before the default color rotation.

See also: “Controlling Bar Chart Patterns, Colors, and Images” on page 816 and “PATTERN Statement” on page 169

Not supported by: Java (partial), ActiveX (partial)

FREQ=*numeric-variable*

specifies a variable whose values weight the contribution of each observation in the computation of the chart statistic. Each observation is counted the number of times specified by the value of *numeric-variable* for that observation. If the value of *numeric-variable* is missing, 0, or negative, the observation is not used in the statistic calculation. Non-integer values of *numeric-variable* are truncated to integers.

FREQ= is valid with all chart statistics.

Because you cannot use TYPE=PERCENT or TYPE=FREQ with the SUMVAR= option, you must use FREQ= to calculate percentages and frequencies based on a sum.

The statistics will not be affected by applying a format to *numeric-variable*.

See also: “Calculating Weighted Statistics” on page 783

GROUP=*group-variable*

organizes the data according to values of *group-variable* and produces a separate pie (or donut) chart for each unique value of *group-variable*. *Group-variable* can be either character or numeric and is always treated as a discrete variable. Missing values for *group-variable* are treated as a valid group. By default, each group includes only those midpoints with nonzero chart statistic values.

By default, the charts are produced in ascending order of group variable value and each is drawn on a separate page or display. Therefore, the effect of GROUP= is essentially the same as using a BY statement except that GROUP= causes the midpoints with the same value to use the same color and fill pattern. To place more than one pie on a page or display, use the ACROSS= or DOWN= options, or both.

See also: “BY Statement” on page 141

Featured in: Example 12 on page 877

HTML=*variable*

identifies the variable in the input data set whose values create links in the HTML file that is created by the ODS statement. These links are associated with an area of the chart and point to the data or graph that you wish to display when the user drills down on the area. The values of *variable* can be up to 1024 characters long. Characters after the 1024-character limit (including any closing quotes) are truncated.

HTML LEGEND=*variable*

identifies the variable in the input data set whose values create links in the HTML file created by the ODS statement. These links are associated with a legend value and point to the data or graph that you wish to display when the user drills down on the value. The values of *variable* can be up to 1024 characters long. Characters after the 1024-character limit (including any closing quotes) are truncated.

Not supported by: Java, ActiveX

INVISIBLE=*value-list*

makes the specified slices invisible, as if they had been removed from the pie. Labels are not printed for invisible slices. *Value-list* is the list of midpoint values for the invisible slices. See the MIDPOINTS= option on page 826 for a description of *value-list*.

The values in the value list must match the existing midpoints exactly, including the case of character midpoints. Any values in the list that do not correspond to existing midpoints are ignored.

JSTYLE

arranges the midpoints in descending order of the statistic value and draws the slices clockwise starting at the 12 o'clock position. The JSTYLE option has the same effect as specifying both the DESCENDING and CLOCKWISE options.

LABEL=(*text argument(s)***) (DONUT only)**

defines the text that is displayed in the donut hole. *Text-argument(s)* defines the text or the appearance of the label, or both. *Text-argument(s)* can be one or more of the following:

'text-string'

provides the text of the label. Enclose each string in quotation marks. Separate multiple strings with blanks.

text-description-suboption

modifies a characteristic such as the font, color, or size of the text string(s) that follows it. *Text-description-suboption* can be

ANGLE=*degrees*

COLOR=*color*

FONT=*font*

HEIGHT=*text-height* <*units*>

JUSTIFY=LEFT | CENTER | RIGHT

ROTATE=*degrees*

The Java and ActiveX devices do not support all of the suboptions. See “Text Description Suboptions” on page 829 for a complete description.

Specify as many text strings and text description suboptions as you want, but enclose them all in one set of parentheses.

Featured in: Example 9 on page 872

Not supported by: Java (partial), ActiveX (partial)

LEGEND | LEGEND=LEGEND<1...99>

generates a legend for the slice names (midpoint values) instead of printing them beside the slices. The legend displays each slice name and its associated pattern. This option also suppresses the display of the chart statistic values. To display the chart statistics, use the VALUE= option.

If you use the SUBGROUP= option, the legend is automatically generated. However, because patterning is always by midpoint, the legend still describes the midpoint values, not the subgroups.

Note: If you request a legend and the slices use hatch patterns, the patterns in the slices are oriented to be visually equivalent to the legend. \triangle

Specifying LEGEND=LEGEND n assigns the specified LEGEND statement to the legend. The Java and ActiveX devices do not support all LEGEND statement options. See “LEGEND Statement” on page 151 for more information.

See also: “LEGEND Statement” on page 151 and SUBGROUP= option on page 828

Featured in: Example 9 on page 872 Example 11 on page 875

Not supported by: Java (partial), ActiveX (partial)

LEVELS=number-of-midpoints | ALL

specifies the number of midpoints for a numeric chart variable. The range for each midpoint is calculated automatically.

If you specify LEVELS=ALL, then all unique midpoint values are graphed. If your data contains a large number of unique midpoint values (over 200), you can use the XPIXELS and YPIXELS GOPTIONS to allow the device driver to render a larger (and more readable) graph.

LEVELS= is ignored if

- the chart variable is character type
- the DISCRETE option is used
- the MIDPOINTS= option is used.

MATCHCOLOR

uses the slice pattern color for all slice labels. MATCHCOLOR overrides the color that is specified in the CTEXT= option.

MIDPOINTS=*value-list*

specifies the midpoint values for the slices. The way you specify *value-list* depends on the type of variable:

- For numeric chart variables, *value-list* is either an explicit list of values, or a starting and an ending value with an interval increment, or a combination of both forms:

n <...*n*>

n TO *n* <BY *increment*>

<*n*...> *n* TO *n* <BY *increment*> <*n* <...*n*>>

If a numeric variable has an associated format, the specified values must be the *unformatted* values.

By default, numeric variable values are treated as continuous (if you omit the DISCRETE option), and

- the lowest midpoint consolidates all data points from negative infinity to the median of the first two midpoints
- the highest midpoint consolidates all data points from the median of the last two midpoints up to infinity
- all other values in *value-list* specify the median of a range of values, and the GCHART procedure calculates the midpoint values.

If you include the DISCRETE option, each value in *value-list* specifies a unique numeric value.

- For character chart variables, *value-list* is a list of unique character values enclosed in quotation marks and separated by blanks:

'*value-1*' <...'value-*n*'>

If a character variable has an associated format, the specified values must be the *formatted* values.

For a complete description of *value-list*, see the ORDER= option on page 130 in the AXIS statement.

Midpoints that represent small percentages are collected into a generic midpoint named OTHER. See the OTHER= option on page 827 and the OTHERLABEL= option on page 827 for more information.

See also: “About Midpoints” on page 780

Featured in: Example 10 on page 873

MIDPOINTS=OLD

generates default midpoints using the Nelder algorithm (*Applied Statistics* 25:94–7, 1976). The MIDPOINTS=OLD option is ignored unless the chart variable is numeric

MISSING

accepts a missing value as a valid midpoint for the chart variable. By default, observations with a missing value are ignored. Missing values are always valid for the group and subgroup variable.

NAME='*entry-name***'**

specifies the name of the catalog entry for the graph. The maximum length for *entry-name* is eight characters. The default name is GCHART. If the name duplicates an existing entry name, then SAS/GRAPH software adds a number to the duplicate name to create a unique name—for example, GCHART1.

NOGROUPHEADING

suppresses the headings that are normally printed above each pie when you use the GROUP= option.

NOHEADING

suppresses the heading that is normally printed at the top of each page or display of output for all devices except Java and ActiveX. For the Java and ActiveX devices, NOHEADING is the default.

Featured in: Example 9 on page 872

Not supported by: Java, ActiveX

NOLEGEND

suppresses the legend that is automatically generated by the SUBGROUP= option. NOLEGEND is ignored if the SUBGROUP= option is not used.

OTHER=*percent-of-total*

collects all midpoints with chart statistic values less than or equal to *percent-of-total* into a generic midpoint named OTHER. The value of *percent-of-total* can be 0 to 100; the default value is 4. Therefore, any slice that represents 4 percent or less of the total is put in the OTHER category.

Note: If you specify a small value for *percent-of-total*, the GCHART procedure may not be able to label all of the small slices. Δ

The OTHER slice is the last slice in the pie, regardless of the order of the slices. (In other words, it is the slice immediately before the starting slice.)

If only one midpoint falls into the OTHER category, its slice is displayed in its normal position in the pie and retains its original label. For example, suppose a pie has these slices and percent values: Coal 35%, Gas 15%, Hydro 5%, and Oil 45%. If you specify OTHER=5, Hydro remains the third slice instead of becoming the last slice.

Featured in: Example 11 on page 875 and Example 12 on page 877

OTHERCOLOR=*color*

specifies the color to use for the OTHER slice. If you omit the OTHERCOLOR= option, GCHART searches for a color specification in this order:

- 1 the CFILL= option
- 2 the COLOR= option in a PATTERN statement
- 3 the COLOR= in a GOPTIONS statement
- 4 the default color list.

For more information, see “Controlling Slice Patterns and Colors” on page 831.

OTHERLABEL=*'text-string'*

specifies a text string up to 16 characters for the label for the OTHER slice. The default label is OTHER.

Featured in: Example 11 on page 875

PERCENT=ARROW | INSIDE | NONE | OUTSIDE

prints the percentage represented by each slice using the specified labeling method. For a description of the option values, see “Selecting and Positioning Slice Labels” on page 830. By default, PERCENT=NONE (percentage is not displayed).

Whether the slice percent displays with or without decimal places, depends on the range of values across the chart. The only way to control the appearance of these values is to calculate the percentage with a DATA step or statistical procedure and use the resulting data set as input to the GCHART procedure. Assign the variable that contains the calculated percentages to the SUMVAR= option.

Featured in: Example 10 on page 873 and Example 12 on page 877

SLICE=ARROW | INSIDE | NONE | OUTSIDE

controls the position and style of the slice name (midpoint value) for each slice. For a description of the option values, see “Selecting and Positioning Slice Labels” on page 830. By default, SLICE=OUTSIDE (the name is outside of the slice).

Featured in: Example 10 on page 873 and Example 12 on page 877

SUBGROUP=subgroup-variable

divides the chart into concentric rings according to the values of *subgroup-variable*. For DEVICE=JAVA, subgroups are implemented using drill-down functionality instead of concentric rings. In the resulting graph, you can select a pie slice to display subgroup information. *Subgroup-variable* can be either character or numeric and is always treated as a discrete variable.

The width of the rings, which is the same for each subgroup, is determined by the radius of the pie and the size of the donut hole, if any.

By default, the subgroup rings are ordered from the outside in, alphabetically (if character) or numerically (if numeric). If the JSTYLE option is also used, the order of the slices within the subgroups is determined by the outermost subgroup. Any inner subgroup that contains a value that is not in the outer subgroup, places the new slice for that value either last or just before the "other" slice, if one is present. That slice order is continued for any remaining subgroups.

Each ring is labeled with its subgroup value; labels are placed to the right of the chart. If the GROUP= option is also used and if all groups contain the same subgroups, then only the first (upper left) chart on each page is labeled. If any group differs in the number of subgroups it contains, then all charts are labeled.

By default the subgroups are outlined in the foreground color. To specify an outline color, use the COUTLINE= option.

SUBGROUP= automatically generates a legend for the midpoint values (not the subgroup values) and suppresses display of the chart statistic. By default the legend appears at the bottom of the chart. To modify the legend, assign a LEGEND definition. To suppress the legend, specify NOLEGEND. To display the chart statistic, use the VALUE= option.

If EXPLODE is also used, it is ignored.

See also: “Controlling Bar Chart Patterns, Colors, and Images” on page 816 and “LEGEND Statement” on page 151

Featured in: Example 9 on page 872 and Example 10 on page 873

SUMVAR=summary-variable

specifies a numeric variable for sum or mean calculations. The GCHART procedure calculates the sum or, if requested, the mean of *numeric-variable* for each midpoint. The resulting statistics are represented by the size of the slice and displayed beside of each slice.

When you use SUMVAR=, the TYPE= option must be either SUM or MEAN. With SUMVAR=, the default is TYPE=SUM.

Featured in: Example 8 on page 869

TYPE=statistic

specifies the chart statistic.

- If the SUMVAR= option is not used, *statistic* can be one of the following:

FREQ

frequency (the default)

PERCENT PCT

percentage

- If SUMVAR= is used, *statistic* can be one of the following:

SUM

sum (the default)

MEAN
mean

Because you cannot use TYPE=FREQ or TYPE=PERCENT with the SUMVAR= option, you must use FREQ= to calculate percentages or frequencies based on a sum.

See also: “About Chart Statistics” on page 782 and “Calculating Weighted Statistics” on page 783

VALUE=ARROW | INSIDE | NONE | OUTSIDE

controls the position and style of the slice value (chart statistic) for each slice. For a description of the option values see “Selecting and Positioning Slice Labels” on page 830. By default, VALUE=OUTSIDE (the value is outside the slice).

Featured in: Example 10 on page 873 and Example 11 on page 875

WOUTLINE=slice-outline-width

specifies the width of the outline in pixels. WOUTLINE= affects both the slice and the subgroup outlines.

Not supported by: Java, ActiveX

Text Description Suboptions

The LABEL= option in the DONUT statement uses text description suboptions to change the color, height, justification, font, and angle of the following text string(s).

ANGLE=*degrees*

A=*degrees*

specifies the angle at which the baseline of the text string(s) is rotated with respect to the horizontal. A positive value for *degrees* moves the baseline counterclockwise; a negative value moves it clockwise. By default, ANGLE=0 (horizontal).

Not supported by: Java

COLOR=*color*

C=*color*

specifies the color for the text string(s). The COLOR= suboption stays in effect until another COLOR= specification is encountered. If you omit COLOR=, LABEL= uses the first color in the colors list. It ignores the CTEXT= graphics option. See Chapter 6, “SAS/GRAPH Colors and Images,” on page 91 for details on specifying *color*.

FONT=*font*

F=*font*

specifies the font for the text string(s). If you omit FONT=, LABEL= uses the font that is specified by the FTEXT= graphics option. If no font is specified, it uses the default hardware font, NONE. See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for details on specifying *font*. The Java and ActiveX devices do not support all fonts.

Not supported by: Java (partial), ActiveX (partial)

HEIGHT=*text-height* <*units*>

H=*text-height* <*units*>

specifies the height of the text string(s). *Text-height* is the number of units. If you omit HEIGHT=, LABEL= uses the height that is specified by the HTEXT= graphics option. If no text height is specified and if the default text height is too large for the donut hole, the size of the label is reduced to fit. *Units* can be CELLS | CM | IN | PCT | PT. If you omit *units*, HEIGHT= uses the unit that is specified by the GUNIT= graphics option, or the default unit, CELLS.

JUSTIFY=LEFT | CENTER | RIGHT

J=L | C | R

specifies the alignment of the text string(s). By default, JUSTIFY=CENTER.

Not supported by: Java, ActiveX

ROTATE=*degrees*

specifies the angle at which each character is rotated with respect to the baseline of the text string. The angle is measured from the current text baseline angle specified by the ANGLE= suboption. A positive value for *degrees* rotates the character counterclockwise; a negative value rotates it clockwise. By default, ROTATE=0 (parallel to the baseline).

Not supported by: Java

Selecting and Positioning Slice Labels

By default, each slice is labeled with its midpoint value (slice name) and its chart statistic value (slice value), which are printed outside of the slice. You can control where and how these labels are displayed with the SLICE= and VALUE= options, respectively. In addition, each slice can display the percentage its midpoint contributes to the total chart statistic (slice percent). Use the PERCENT= option to request slice percent.

The SLICE=, VALUE=, and PERCENT= options use the same values:

ARROW

places the text outside the slice and connects the text to the slice with a line. This labeling method reduces the radius of the pie. The arrow uses the color that is specified by CTEXT= in the PIE, PIE3D, or DONUT statement. If CTEXT= is omitted, the arrow uses the first color in the colors list.

INSIDE

places the text inside the slice. The label overlays the slice fill patterns. This labeling method increases the radius of the pie.

NONE

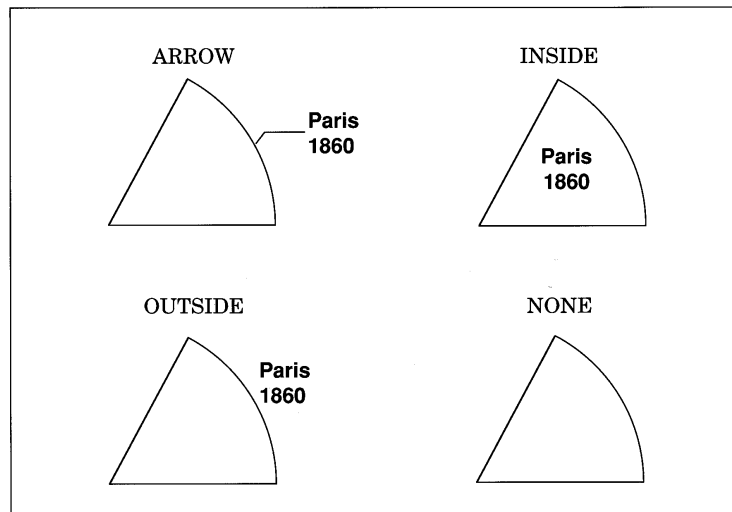
suppresses the text.

OUTSIDE

places the text outside of the slice.

Figure 29.14 on page 831 illustrates these values.

Figure 29.14 Slice Labeling Methods



The SLICE= and VALUE= options are dependent on each other. If you specify only VALUE= or only SLICE=, the other option automatically uses the same labeling method. PERCENT= is independent of these two.

Be careful about the combinations that you specify. For example, if you specify PERCENT=ARROW and VALUE=OUTSIDE, the line that connects the percentage information to each slice may overlay the statistic value.

If your pie has many slices, the labels may overlap, particularly if there are several small slices together. You can correct the overlapping labels by using

- FTEXT= graphics option to decrease the size of the labels.
- the Graphics Editor to adjust the labels by moving or resizing the text.
- ANGLE= to change the orientation of the pie.
- MIDPOINTS= to rearrange slices so that small slices are not together.
- OTHER= to group more midpoints into the OTHER category.
- the HPOS= and VPOS= graphics options to increase the number of cells in your display. (See “About the Graphics Output Area” on page 34 for details.)

Controlling Slice Patterns and Colors

Pie and donut charts are always patterned by midpoint. Even when you specify subgrouping, the patterning method does not change from midpoint to subgroup.

Default patterns and outlines

Each slice in a pie or donut chart is filled with a pattern. By default, the procedure

- fills the slices with pie patterns, beginning with the default fill, PSOLID, and rotating through the colors in the colors list. When the solid patterns are exhausted, the procedure selects the next default pie pattern and rotates it through the colors list. It continues in this fashion until all of the required patterns have been assigned.

Note: PIE3D always uses solid patterns. Δ

If you use the device’s default colors and the first color in the list is either black or white, the procedure does not create a pattern in that color. If you specify a colors list with the COLORS= graphics option, the procedure uses all the colors in the list to generate the patterns.

- outlines slices and subgroup segments using the first color in the colors list. To change the outline color, use the `COUTLINE=` option.

See “About Patterns” on page 784 for more information on how the `GCHART` procedure assigns default patterns and outlines.

Controlling patterns

You can control slice patterns and their outlines in several ways.

- To select a different fill for the slices, such as empty or hatched, you can
 - request a single hatched fill pattern for all slices by specifying the `FILL=X` option on the `PIE` or `DONUT` statement. The pattern specified by `FILL=X` rotates through the colors list as many times as needed to generate all of the patterns that are required by the chart. If you specify a single color with either `CFILL=` or the graphics option, `CPATTERN=`, all slices use the same color as well as the same pattern.
 - specify a pattern with the `VALUE=` option in the `PATTERN` statement. Only pie patterns are valid; all other pattern specifications are ignored. For a complete description of all pie patterns, see `VALUE=` on page 174 in “`PATTERN` Statement” on page 169.

If no color options are specified, the procedure rotates each specified fill once through the colors list. Otherwise the `PATTERN` statement generates one pattern definition for the specified pattern and color. When all of the specified patterns are exhausted, the procedure starts rotating through the default pie patterns, beginning with `PSOLID`.

- To select colors for the slices, you can
 - specify a single pattern color with the `CFILL=` option, or with the `CPATTERN=` graphics option, or with a colors list of one color. For the `PIE` and `DONUT` statements, `CFILL=` starts with the default solid color and uses the foreground color for outlines, whereas `CPATTERN=` and a colors list of one color skip the solid pattern and, beginning with `P2N0`, use each pie hatch pattern with the specified color, and use the fill color for the outline color.
 - specify only `COLOR=` in one or more `PATTERN` statements. In this case, the procedure creates a solid pattern for each specified color. When it runs out of `PATTERN` statements, it returns to the default patterns, beginning with `PSOLID`, and rotates them each through the colors list. Whenever you specify a `PATTERN` statement, the default outline color is `SAME`.
- To define specific patterns and colors for the slices, use `PATTERN` statements and specify both the `VALUE=` and `COLOR=` options. If you provide fewer `PATTERN` definitions than the chart requires, the `GCHART` procedure uses the default pattern rotation for the slices that are drawn after all of the defined patterns are exhausted.

Whenever you use `PATTERN` statements, the default outline color changes to `SAME`. That is, the outline color is the same as the fill color. To change the outline color, use the `COUTLINE=` option on page 821.

See “About Patterns” on page 784 for more information on how the `GCHART` procedure uses patterns and outlines. See “`PATTERN` Statement” on page 169 for a description of default pie patterns.

Modifying the Statistic Heading and the Group Heading

By default, the procedure prints a heading at the top of each pie (or donut) chart that indicates the type of statistic charted and the name of the chart variable— for example, *SUM of SALES by SITE*. You can suppress this heading with the `NOHEADING` option.

When you use the GROUP= option, a heading is printed above each pie indicating the name of the group variable and its value for the particular pie— for example, *SITE=Paris*. You can suppress these headings with the NOGROUPHEADING option. You can also suppress the variable name *SITE=* so that only the value *Paris* remains. To do this, use a LABEL statement and assign a null value to the variable name, for example,

```
label site='00'x;
```

Because the AXIS statement cannot be used by the PIE, PIE3D, and DONUT statements, you should use the FTEXT= and HTEXT= graphics options to control the font and height of text on the chart. Increasing the value of the HTEXT= graphics option decreases the size of the pie if any slice labels are positioned outside.

STAR Statement

Creates star charts in which the length of the spines represents the value of the chart statistic for each category of data or midpoint.

Requirements: At least one chart variable is required.

Global statements: FOOTNOTE, PATTERN, TITLE,

Supports: Drill-down functionality (slices only)

Not supported by: Java, ActiveX

Description

The STAR statement specifies the variable or variables that define the categories of data to chart. This statement automatically

- determines the midpoints.
- calculates the chart statistic for each midpoint (the default is `FREQ`).
- scales each spine or slice to represent the chart statistic. Slices or spines are drawn for all midpoints where the value of the chart statistic is greater than the value that is specified in the `STARMIN=` option.
- arranges the spines or slice counterclockwise around the star in ascending order of midpoint value, starting at the three o'clock position.
- prints the midpoint value and chart statistic beside each spine or slice.
- assigns patterns to the slices.

If all the data to be charted with the STAR statement are positive, the center of the star represents 0 and the outside circle represents the maximum value. If negative values are calculated for the chart statistic, the center represents the minimum value in the data. You can specify other values for the center and outside of the circle with the `STARMIN=` and `STARMAX=` options.

You can also use statement options to select or order the midpoints, to change the type of chart statistic, and to modify the appearance of the chart, including the content and position of the spine or slice labels, and patterns that fill the slice. You can specify additional variables by which to group or sum the data.

Star charts allow grouping, which creates two or more separate charts that display in rows or columns on one graph.

In addition, you can use global statements to modify patterns as well as add titles, footnotes, and notes to the chart. You can also use an Annotate data set to enhance the chart.

Syntax

STAR *chart-variable(s) </option(s)>*;

option(s) can be one or more options from any or all of the following categories:

- appearance options
 - ANGLE=*degrees*
 - ANNOTATE=*Annotate-data-set*
 - CFILL=*fill-color*
 - COUTLINE=*star-outline-color* | SAME
 - FILL=SOLID | X
 - NOCONNECT
 - STARMAX=*max-value*
 - STARMIN=*min-value*
 - WOOUTLINE=*slice-outline-width*
- statistic options
 - FREQ=*numeric-variable*
 - SUMVAR=*summary-variable*
 - TYPE=*statistic*
- midpoint options
 - DISCRETE
 - LEVELS=*number-of-midpoints*
 - MIDPOINTS=*value-list*
 - MIDPOINTS=OLD
 - MISSING
- grouping options
 - ACROSS=*number-of-columns*
 - DOWN=*number-of-rows*
 - GROUP=*group-variable*
- labeling options
 - CTEXT=*text-color*
 - MATCHCOLOR
 - NOGROUPHEADING
 - NOHEADING
 - PERCENT=ARROW | INSIDE | NONE | OUTSIDE
 - SLICE=ARROW | INSIDE | NONE | OUTSIDE
 - VALUE=ARROW | INSIDE | NONE | OUTSIDE
- catalog entry description options
 - DESCRIPTION=*'entry-description'*
 - NAME=*'entry-name'*
- ODS options
 - HTML=*variable*

Required Arguments

chart-variable(s)

specifies one or more variables that define the categories of data to chart. Each chart variable draws a separate chart. All variables must be in the input data set. Separate multiple chart variables with blanks.

See also: “About Chart Variables” on page 779

Options

Options in a STAR statement affect all of the graphs that are produced by that statement. You can specify as many options as you want and list them in any order. For details on specifying colors, see Chapter 6, “SAS/GRAPH Colors and Images,” on page 91.

ACROSS=*number-of-columns*

draws *number-of-columns* stars across the procedure output area. ACROSS= is ignored unless you also use the GROUP= option. If *number-of-columns* calls for more stars than fit horizontally in the graphics area of the output device, no stars are drawn and an error message is written to the SAS log.

If you also use the DOWN= option, the star charts are drawn in left-to-right and top-to-bottom order.

ANGLE=*degrees*

starts the first slice at the specified angle. A value of 0 for *degrees* corresponds to the 3 o'clock position. *Degrees* can be either positive or negative. Positive values move the starting position counterclockwise; negative values move the starting position clockwise.

If the star chart uses spines instead of slices, *degrees* specifies the angle of the position halfway between the first spine and the last spine.

By default, ANGLE=0, which places the first spine or the center of the first slice of the star at the 0 degree position. Successive star spines or slices are drawn counterclockwise from the starting position.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate charts that are produced by the STAR statement.

Note: Annotate coordinate systems 1, 2, 7, and 8 (data system coordinates) are not valid with star charts. △

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

CFILL=*fill-color*

specifies one color for all slices in the chart, regardless of whether the fill is solid or hatch. If no pattern is specified on the PATTERN statement or with the FILL= option, the procedure starts with the default solid fill and then, beginning with P2N0, uses each default star hatch pattern with the specified color. For the outline color, the procedure uses the foreground color, which is the first color in the colors lists. Use COUTLINE= to specify a different outline color. CFILL= overrides any other pattern color specification and controls the color of all slices.

COUTLINE=*star-outline-color* | **SAME**

specifies the color for the circle that surrounds the star chart and for the slice outlines or spines.

SAME specifies that the outline color of a slice is the same as the interior pattern color. Specifying COUTLINE=SAME affects only slice outlines and has no effect on the color of the circle.

The default circle color is the first color in the colors list (the foreground color). The default slice outline color depends on the PATTERN statement:

- If you do not specify the PATTERN statement, the default outline color is the foreground color (the first color in the colors list).
- If you do not specify the PATTERN statement or the V6COMP graphics options, the default is COUTLINE=SAME.

Note: If you specify empty patterns, (VALUE=PEMPTY in a PATTERN statement) you should not change the outline color from the default value, SAME, to a single color. Otherwise all the outlines will be one color and you will not be able to distinguish between the empty areas. △

See also: “Selecting Patterns for the Star Charts” on page 840 and “About Patterns” on page 784

Featured in: Example 14 on page 880

CTEXT=*text-color*

specifies a color for all text on the chart. Text includes all slice labels, the chart heading, and group headings if grouping is used.

If you omit CTEXT=, PROC GCHART searches for a color specification in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the first color in the colors list (the default).

The MATCHCOLOR option overrides the CTEXT= option for star slice labels.

DESCRIPTION=*'entry-description'*

DES=*'entry-description'*

specifies the description of the catalog entry for the chart. The maximum length for *entry-description* is 256 characters. The description does not appear on the chart. By default, the GCHART procedure assigns a description of the form STAR CHART OF *variable*, where *variable* is the name of the chart variable.

The *entry-description* can include the #BYLINE, #BYVAL, and #BYVAR substitution options, which work as they do when used on TITLE, FOOTNOTE, and NOTE statements. Refer to the description of the options on page 222, and “Substituting BY Line Values in a Text String” on page 226. The 256-character limit applies before the substitution takes place for these options; thus, if in the SAS program the entry-description text exceeds 256 characters, it is truncated to 256 characters, and then the substitution is performed.

The descriptive text is shown in each of the following:

- the "description" portion of the Results window
- the catalog-entry properties that you can view from the Explorer window
- the Table of Contents that is generated when you use CONTENTS= on an ODS statement (see “Linking to Output through a Table of Contents” on page 495), assuming the GCHART output is generated while the contents page is open
- the Description field of the PROC GREPLAY window.

DISCRETE

treats a numeric chart variable as a discrete variable rather than as a continuous variable. The GCHART procedure creates a separate midpoint and, hence, a separate star slice for each unique value of the chart variable. If the variable has a format associated with it, each format value is treated as a separate value.

The LEVELS= option is ignored when you use the DISCRETE option. The MIDPOINTS= option overrides the DISCRETE option.

Featured in: Example 14 on page 880

DOWN=number-of-rows

draws *number-of-rows* stars vertically in the procedure output area. The DOWN= option is ignored unless you also use the GROUP= option. If *number-of-rows* calls for more stars than fit vertically in the graphics area of the output device, no stars are drawn and an error message is written to the SAS log.

If you also use the ACROSS= option, the stars are drawn in left-to-right and top-to-bottom order.

FILL=SOLID | X

specifies the fill pattern for *all* slices in the star chart:

SOLID

S

rotates a solid fill through the colors list as many times as necessary. This is the default.

X

rotates a single hatch pattern through the colors list as many times as necessary. If you use default device colors (the COLORS= option is omitted), the fill skips the first color in the colors list.

FILL= overrides any patterns that are specified in PATTERN statements.

By default, the fill patterns take the colors from the current colors list in rotation. If any PATTERN statements have been defined, the colors in the PATTERN definitions are used, in order, before the default color rotation.

Featured in: Example 14 on page 880

FREQ=numeric-variable

specifies a variable whose values weight the contribution of each observation in the computation of the chart statistic. Each observation is counted the number of times that are specified by the value of *numeric-variable* for that observation. If the value of *numeric-variable* is missing, 0, or negative, the observation is not used in the statistic calculation. Non-integer values of *numeric-variable* are truncated to integers.

FREQ= is valid with all chart statistics.

Because you cannot use TYPE=PERCENT or TYPE=FREQ with the SUMVAR= option, you must use FREQ= to calculate percentages and frequencies based on a sum.

The statistics will not be affected by applying a format to *numeric-variable*.

See also: “Calculating Weighted Statistics” on page 783

GROUP=variable

organizes the data according to values of *group-variable* and produces a separate star chart for each unique value of *group-variable*. *Group-variable* can be either character or numeric and is always treated as a discrete variable. Missing values for *group-variable* are treated as a valid group.

By default, the charts are produced in ascending order of group variable value and each is drawn on a separate page or display. Therefore, the effect of GROUP= is essentially the same as using a BY statement except that GROUP= causes the midpoints with the same value to use the same color and fill pattern. To place more than one star chart on a page or display, use the ACROSS= or DOWN= options, or both.

HTML=variable

identifies the variable in the input data set whose values create links in the HTML file that is created by the ODS statement. These links are associated with an area of the chart and point to the data or graph that you wish to display when the user drills down on the area. Only star charts with slices support drill-down functionality. The values of *variable* can be up to 1024 characters long. Characters after the 1024-character limit (including any closing quotes) are truncated.

LEVELS=number-of-midpoints

specifies number of midpoints for a numeric chart variable. The range for each midpoint is calculated automatically using the algorithm described by Terrell and Scott (1985). LEVELS= is ignored if

- the chart variable is character type
- the DISCRETE option is used
- the MIDPOINTS= option is used.

MATCHCOLOR

uses the slice pattern color for all slice labels. MATCHCOLOR overrides the color that is specified in the CTEXT= option. If the chart uses spines instead of slices, the spine color is used for the slice label and value text.

MIDPOINTS=value-list

specifies the midpoint values for the slices. The way you specify *value-list* depends on the type of variable:

- For numeric chart variables, *value-list* is either an explicit list of values, or a starting and an ending value with an interval increment, or a combination of both forms:

n <...*n*>

n TO *n* <BY *increment*>

n <...*n*> TO *n* <BY *increment*> <*n* <...*n*>>

If a numeric variable has an associated format, the specified values must be the *unformatted* values.

By default, numeric variable values are treated as continuous (if you omit the DISCRETE option), and

- the lowest midpoint consolidates all data points from negative infinity to the median of the first two midpoints
- the highest midpoint consolidates all data points from the median of the last two midpoints up to infinity
- all other values in *value-list* specify the median of a range of values, and the GCHART procedure calculates the midpoint values.

If you include the DISCRETE option, each value in *value-list* specifies a unique numeric value.

- For character chart variables, *value-list* is a list of unique character values enclosed in quotation marks and separated by blanks:

'*value-1*' <...'value-*n*'>

If a character variable has an associated format, the specified values must be the *formatted* values.

For a complete description of *value-list*, see the ORDER= option on page 130 in the AXIS statement.

See also: "About Midpoints" on page 780

MIDPOINTS=OLD

generates default midpoints using the Nelder algorithm (*Applied Statistics* 25:94–7, 1976). The MIDPOINTS=OLD option is ignored unless the chart variable is numeric

MISSING

accepts a missing value as a valid midpoint for the chart variable. By default, observations with a missing value are ignored. Missing values are always valid for the group variable.

NAME='entry-name'

specifies the name of the catalog entry for the graph. The maximum length for *entry-name* is eight characters. The default name is GCHART. If the name duplicates an existing entry name, then SAS/GRAPH software adds a number to the duplicate name to create a unique name—for example, GCHART1.

NOCONNECT

draws only star spines without connecting lines. By default, the spines are connected to form slices.

Featured in: Example 14 on page 880

NOGROUPHEADING

suppresses the headings normally printed above each star when you use the GROUP= option.

NOHEADING

suppresses the heading normally printed at the top of each page or display of star chart output.

Featured in: Example 14 on page 880

PERCENT=ARROW | INSIDE | NONE | OUTSIDE

prints the percentage represented by each slice using the specified labeling method. For a description of the option values see “Selecting and Positioning Spine and Slice Labels” on page 840. By default, PERCENT=NONE (percentage is not displayed).

SLICE=ARROW | INSIDE | NONE | OUTSIDE

controls the position and style of the slice name (midpoint value) for each slice. For a description of the option values, see “Selecting and Positioning Spine and Slice Labels” on page 840. By default, SLICE=OUTSIDE (the name is outside the slice).

STARMAX=*max-value*

scales the chart so that the outside (or edge) of the circle represents the value that is specified by *max-value*. By default, the value for STARMAX= is the maximum chart statistic value.

STARMIN=*min-value*

scales the chart so that the center of the circle represents the value that is specified by *min-value*. By default, STARMIN=0. If the chart statistic has negative values, by default the value for STARMIN= is the minimum chart statistic value.

SUMVAR=*summary-variable*

specifies a numeric variable for sum or mean calculations. The GCHART procedure calculates the sum or, if requested, the mean of the value of *numeric-variable* for each midpoint. The resulting statistics are represented by the size of the slice and displayed beside each slice.

When you use SUMVAR=, the TYPE= option must be either SUM or MEAN. With SUMVAR=, the default is TYPE=SUM.

Featured in: Example 13 on page 879

TYPE=*statistic*

specifies the chart statistic.

- If the SUMVAR= option is not used, *statistic* can be one of the following:

FREQ

frequency (the default)

PERCENT PCT

percentage

If SUMVAR= is used, *statistic* can be one of the following:

SUM

sum (the default)

MEAN
mean

Because you cannot use TYPE=FREQ or TYPE=PERCENT with the SUMVAR= option, you must use FREQ= to calculate percentages or frequencies based on a sum.

See also: “About Chart Statistics” on page 782 and “Calculating Weighted Statistics” on page 783

VALUE=ARROW | INSIDE | NONE | OUTSIDE

controls the position and style of the slice value (chart statistic) for each slice. For a description of the option values, see “Selecting and Positioning Spine and Slice Labels” on page 840. By default, VALUE=OUTSIDE (the value is outside of the slice).

WOUTLINE=*slice-outline-width*

specifies the width of the outline in pixels. WOUTLINE= affects the slice outlines.

Selecting and Positioning Spine and Slice Labels

By default, each spine or slice is labeled with its midpoint value and its chart statistic value, which are printed outside of the circle. You can control where and how these labels are displayed with the SLICE= and VALUE= options, respectively. In addition, each spine can display the percentage that its midpoint contributes to the total chart statistic (spine percent). Use the PERCENT= option to request spine percent.

The SLICE=, VALUE=, and PERCENT= options use the same values:

ARROW

places the text outside of the star circle and connects the text to the circle with a line. The line points to the spine or the center of the slice. The arrow uses the color that is specified by CTEXT= in the STAR statement. If you omit CTEXT=, the arrow uses the first color in the colors list.

INSIDE

places the text inside the star circle.

NONE

suppresses the text.

OUTSIDE

places the text outside the star circle.

Figure 29.14 on page 831 illustrates these values.

The SLICE= and VALUE= options are dependent on each other. If you specify only VALUE= or only SLICE=, the other option automatically uses the same labeling method. PERCENT= is independent of these two.

Be careful about the combinations that you specify. For example, if you specify PERCENT=ARROW and VALUE=OUTSIDE, the line that connects the percentage information to each spine may overlay the statistic value.

Selecting Patterns for the Star Charts

Star charts are always patterned by midpoint.

Default patterns and outlines

Each slice in a star chart is filled with a pattern. By default, the procedure

- fills the slices with star patterns, beginning with the default fill, PSOLID, and rotating through the colors in the colors list. When the solid patterns are exhausted, the procedure selects the next default star pattern and rotates it

through the colors list. It continues in this fashion until all the required patterns have been assigned.

If you use the device's default colors and the first color in the list is either black or white, the procedure does not create a pattern in that color. If you specify a colors list with the `COLORS=` graphics option, the procedure uses all of the colors in the list to generate the patterns.

- outlines slices using the first color in the colors list. To change the outline color, use the `COUTLINE=` option.

See “About Patterns” on page 784 for more information on how the GCHART procedure assigns default patterns and outlines.

Controlling patterns

You can control slice patterns and their outlines in several ways.

- To select a different fill for the slices, such as empty or hatched, you can
 - request a single hatched fill pattern for all slices by specifying the `FILL=X` option on the STAR statement. The pattern that is specified by `FILL=X` rotates through the colors list as many times as needed to generate all the patterns required by the chart. If you specify a single color with either `CFILL=` or the graphics option, `CPATTERN=`, all slices use the same color as well as the same pattern.
 - specify a pattern with the `VALUE=` option in the `PATTERN` statement. Only star patterns are valid; all other pattern specifications are ignored. For a complete description of all star patterns, see `VALUE=` on page 174 in “PATTERN Statement” on page 169.

If no color options are specified, the procedure rotates each specified fill once through the colors list. Otherwise the `PATTERN` statement generates one pattern definition for the specified pattern and color. When all of the specified patterns are exhausted, the procedure starts rotating through the default star patterns, beginning with `PSOLID`.

- To select colors for the slices, you can
 - specify a single pattern color with the `CFILL=` option, or with the `CPATTERN=` graphics option, or with a colors list of one color. If you use `CFILL=`, the procedure starts with the default solid color and uses the foreground color for outlines. If you use `CPATTERN=` or a colors list of one color, the procedure skips the default solid fill and, beginning with `P2N0`, uses each default star hatch pattern with the specified color, and uses the fill color for the outline color.
 - specify only `COLOR=` in one or more `PATTERN` statements. In this case, the procedure creates a solid pattern for each specified color. When it runs out of `PATTERN` statements, it returns to the default patterns, beginning with `PSOLID`, and rotates them each through the colors list. Whenever you specify a `PATTERN` statement, the default outline color is `SAME`.
- To define specific patterns and colors for the slices, use `PATTERN` statements and specify both the `VALUE=` and `COLOR=` options. If you provide fewer `PATTERN` definitions than the chart requires, the GCHART procedure uses the default pattern rotation for the slices that are drawn after all defined patterns are exhausted.

Whenever you use `PATTERN` statements, the default outline color changes to `SAME`. That is, the outline color is the same as the fill color. To change the outline color, use the `COUTLINE=` option on page 821.

See “About Patterns” on page 784 for more information on how the GCHART procedure uses patterns and outlines. See “PATTERN Statement” on page 169 for a description of default star patterns.

Modifying the Statistic Heading and the Group Heading

By default, the procedure prints a heading at the top of each chart indicating the type of statistic charted and the name of the chart variable— for example, **SUM of SALES by SITE**. You can suppress this heading with the NOHEADING option.

When you use the GROUP= option, a heading is printed above each star indicating the name of the group variable and its value for the particular star— for example, **SITE=Paris**. You can suppress these headings with the NOGROUPHEADING option. You can also suppress the variable name *SITE=* so that only the value *Paris* remains. To do this, use a LABEL statement and assign a null value to the variable name, as shown in this example:

```
label site='00'x;
```

Because the AXIS statement cannot be used by the STAR statement, you should use the FTEXT= and HTEXT= graphics options to control the font and height of text on the chart. Increasing the value of HTEXT= decreases the size of the star if any slice labels are positioned outside. For a description of these graphics options, see Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261.

Examples

Example 1: Specifying the Sum Statistic in a Block Chart

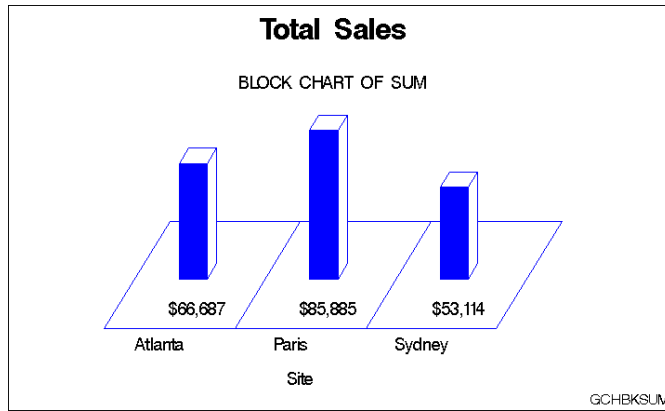
Procedure features:

BLOCK statement option:

SUMVAR=

Other features: FORMAT statement

Sample library member: GCHBKSUM



This example produces a block chart of total sales for three sites by charting the values of the character variable `SITE` and calculating the sum of the variable `SALES` for each site. It prints formatted values of the sales statistics below the blocks.

The chart uses default patterns and colors. The block faces use the default pattern fill, which is solid. Because a colors list is specified in the `GOPTIONS` statement, the default fill color is the first color in the list, blue. The midpoint grid and the block outlines also use the first color in the list.

All the blocks use the same pattern because by default patterns change for subgroups and in this chart subgroups are not specified.

Set the graphics environment. `CTEXT=` specifies the color for all text on the output. `COLORS=` specifies the colors list, which is used by the default patterns and outlines.

```
goptions reset=global gunit=pct border cback=white
        ctext=black colors=(blue green red)
        ftext=swiss ftitle=swissb
        htitle=6 htext=3.5;
```

Create data set TOTALS. `TOTALS` contains quarterly sales data for three manufacturing sites for one year. Sales figures are broken down by department.

```
data totals;
  length dept $ 7 site $ 8;
  input dept site quarter sales;
  datalines;
Parts   Sydney  1 7043.97
Parts   Atlanta 1 8225.26
Parts   Paris   1 5543.97
...more data lines...
Tools   Sydney  4 1775.74
Tools   Atlanta 4 3424.19
Tools   Paris   4 6914.25
;
```

Define title and footnote.

```
title 'Total Sales';
footnote j=r 'GCHBKSUM ';
```

Produce the block chart. The BLOCK statement produces a block chart. SUMVAR= calculates the sum of SALES for each value of the chart variable SITE. With SUMVAR= the default statistic is SUM. The summary variable SALES is assigned a dollar format.

```
proc gchart data=reflib.totals;
  format sales dollar8.;
  block site / sumvar=sales;
run;
quit;
```

Example 2: Grouping and Subgrouping a Block Chart

Procedure features:

BLOCK statement options:

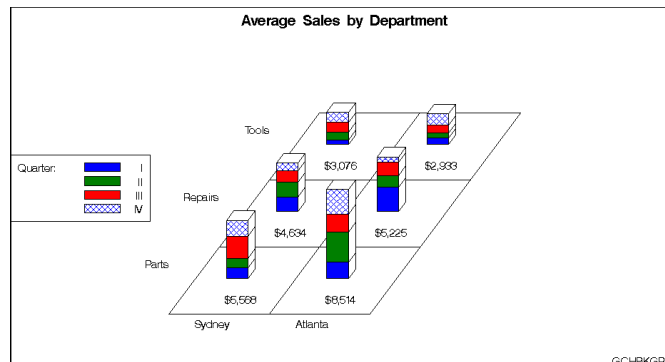
CAXIS=
 COUTLINE=
 GROUP=
 LEGEND=
 MIDPOINTS=
 NOHEADING
 SUBGROUP=
 TYPE=

Other features:

LABEL statement
 LEGEND statement
 Default pattern rotation

Data set: TOTALS

Sample library member: GCHBKGRP



This example shows average quarterly sales for each department at two of the three manufacturing sites in the TOTALS data set; it excludes the Paris site from the chart.

The program groups the chart data (sites) by department, and subgroups department sales data by quarter. Each site is a midpoint. Because the sites are grouped by department, each midpoint has a separate square for each department and the height of the block represents total sales for that department.

The blocks are subgrouped to show how quarterly sales contribute to total sales; each segment represents sales for a quarter. A legend explaining the subgroup patterns appears below the midpoint grid.

The subgroups use four default patterns. The first three patterns are created by rotating the first default fill, solid, through the three colors in the colors list defined in the GOPTIONS statement. The fourth default pattern is created by using the second default pattern fill, X1, with the first color in the colors list, blue.

Because the first color in the colors list is also the default color for several other elements, the program includes several options that override the default: CTEXT= colors all text, CAXIS= colors the midpoint grid, COUTLINE= colors the pattern outline. For more information on patterns and colors, see “Controlling Block Chart Patterns and Colors” on page 794.

Assign the libref and set the graphics environment. COLORS= specifies a colors list that is used by the default patterns. CTEXT= specifies black for all text.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(blue green red) ctext=black
          ftitle=swissb ftext=swiss htitle=4 htext=3;
```

Define title and footnote.

```
title 'Average Sales by Department';
footnote j=r 'GCHBKGRP ';
```

Define legend characteristics. LABEL= assigns new text to the legend label. CBORDER= draws a black frame around the legend.

```
legend1 cborder=black
        label=('Quarter:')
        position=(middle left outside)
        mode=protect
        across=1;
```

Produce the block chart. The LABEL statement suppresses the midpoint and group labels by assigning a null hexadecimal string to each variable name.

```
proc gchart data=reflib.totals;
  format quarter roman.;
  format sales dollar8.;
  label site='00'x dept='00'x;
```

TYPE= specifies the chart statistic as the mean value of the summary variable SALES for each site. MIDPOINTS= selects the two sites and the order in which they appear. GROUP= creates a separate row of blocks for each different value of DEPT. SUBGROUP= divides each block into separate segments for the four quarters. LEGEND= assigns the LEGEND1 statement to the graph. NOHEADING suppresses the default heading that would otherwise appear above the chart. CAXIS= colors the grid black. COUTLINE= specifies the outline color for the blocks.

```

block site / sumvar=sales
              type=mean
              midpoints='Sydney' 'Atlanta'
              group=dept
              subgroup=quarter
              legend=legend1
              noheading
              coutline=black
              caxis=black;

run;
quit;

```

Example 3: Specifying the Sum Statistic in Bar Charts

Procedure features:

HBAR statement options:

SUMVAR=

VBAR3D statement options:

SUMVAR=

COUTLINE=

Other features:

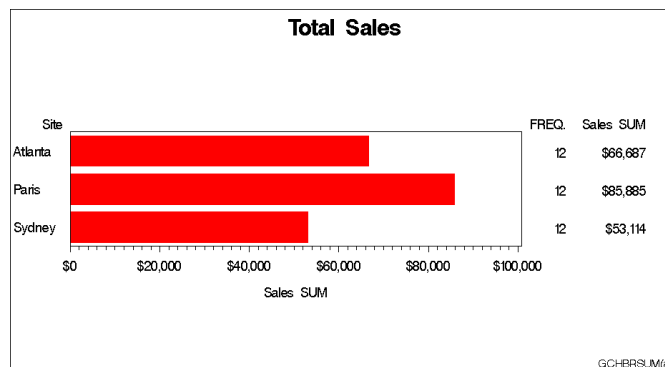
FORMAT statement

PATTERN statement

RUN-group processing

Data set: TOTALS

Sample library member: GCHBRSUM



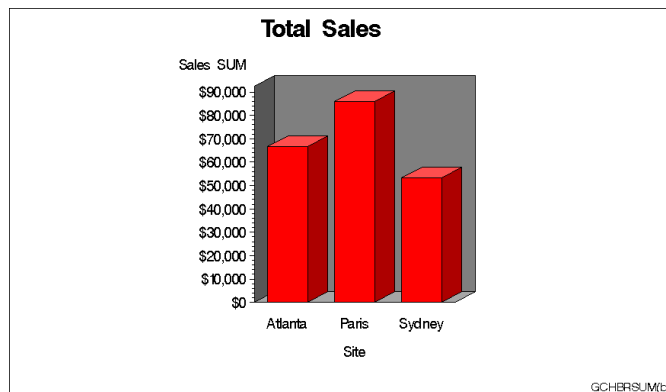
This example produces two bar charts that show total sales for three sites by charting the values of the character variable `SITE` and calculating the sum of the variable `SALES` for each site.

In the horizontal bar chart shown above, the summary statistics are printed by default to the right of the bars and display the formatted values of `SALES`.

The bars use the default pattern fill, which is solid. Because a colors list is specified in the `GOPTIONS` statement, the first default pattern color is the first color in the list. To avoid having black bars, the program uses a `PATTERN` statement to specify the pattern color. Using a `PATTERN` statement causes the default bar outline color to match the fill color. All the bars display the same pattern because by default patterns change for subgroups and in this chart subgroups are not specified.

The output also shows the frame that is drawn by default around the axis area.

The second bar chart is a 3D vertical bar chart, shown in the following output. Vertical bar charts do not generate a table of statistics and by default do not print any chart statistics. This chart uses the same pattern as the horizontal bar chart, but the `VBAR3D` statement specifies a black outline for the bars.



Assign the libref and set the graphics environment. `COLORS=` specifies the colors list, which is used by the default patterns and outlines.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border
          colors=(black red blue green)
          cback=white ftitle=swissb ftext=swiss
          htitle=6 htext=3.5;
```

Define title and footnote for the first chart.

```
title1 'Total Sales';
footnote1 h=3 j=r 'GCHBRSUM(a) ';
```

Specify a color for the pattern. The `PATTERN` statement explicitly defines `RED` as the color for the first solid pattern.

```
pattern1 color=red;
```

Produce the horizontal bar chart. The HBAR statement produces a two-dimensional bar chart. SUMVAR= calculates the sum of SALES for each value of the chart variable SITE. The default statistic for SUMVAR= is SUM. The summary variable SALES is assigned a dollar format.

```
proc gchart data=reflib.totals;
  format sales dollar8.;
  hbar site / sumvar=sales;
run;
```

Produce the vertical bar chart. Because the procedure supports RUN-group processing, you do not have to repeat the PROC GCHART statement to generate the second chart. The VBAR3D statement produces a three-dimensional vertical bar chart. The FOOTNOTE1 statement replaces the previous footnote. COUTLINE= assigns a black outline to the bars.

```
footnote1 h=3 j=r 'GCHBRSUM(b) ';
vbar3d site / sumvar=sales
        coutline=black;

run;
quit;
```

Example 4: Subgrouping a 3D Vertical Bar Chart

Procedure features:

VBAR statement options:

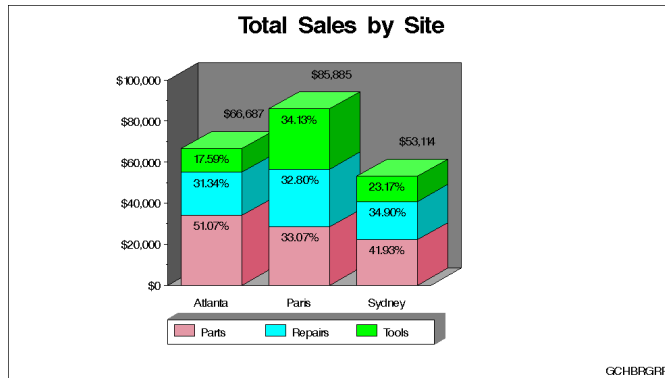
```
CFRAME=
INSIDE=SUBPCT
LEGEND=
MAXIS=
OUTSIDE=SUM
RAXIS=
SPACE=
SUBGROUP=
WIDTH=
```

Other features:

```
AXIS statement
FORMAT statement
GOPTIONS statement
  OFFSHADOW=
LEGEND statement
PATTERN statement
```

Data set: TOTALS

Sample library member: GCHBRGRP



This example subgrouped by department the 3D vertical bar chart of total sales for each site that is shown in Example 3 on page 846. In addition to subdividing the bars to show the amount of sales for each department for each site, the chart displays statistics both inside and outside of the bars. OUTSIDE=SUM prints the total sales for the site above each bar. INSIDE=SUBPCT prints the percent each department contributed to the total sales for its site inside of each subgroup segment.

The legend has a block-effect shadow whose color matches the backplane. The graphics option OFFSHADOW= defines the size and position of the block shadow. Both the LEGEND statement and the AXIS statement use the ORIGIN= option to line up the legend and the chart by explicitly positioning their lower left corners.

Assign the libref and set the graphics environment. OFFSHADOW= defines the depth of the block around the legend box. The positive values position the shadow above and to the right of the legend.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
         colors=(black red green blue) ftitle=swissb
         ftext=swiss htitle=6 htext=4
         offshadow=(1.5,1.5);
```

Define title and footnote.

```
title1 'Total Sales by Site';
footnote1 h=3 j=r 'GCHBRGRP ';
```

Modify the midpoint axis. LABEL= suppresses the axis label. ORIGIN= positions the left end of the horizontal axis at a point that is 25% of the width of the graphics output area.

```
axis1 label=none
      origin=(24,);
```

Modify the response axis. ORDER= specifies the major tick values for the response axis. OFFSET= moves the top tick mark to the end of the axis line.

```
axis2 label=none
      order=(0 to 100000 by 20000)
      minor=(number=1)
```

```
offset=(,0);
```

Modify the legend. LABEL= suppresses the legend label. SHAPE= defines the size of the legend values. CBORDER= draws a black frame around the legend. CBLOCK= specifies a gray block that matches the 3D planes. ORIGIN= specifies the same position as in the AXIS1 statement.

```
legend1 label=none
      shape=bar(3,3)
      cborder=black
      cblock=gray
      origin=(24,);
```

Define pattern characteristics. PATTERN statements define the colors that are assigned to subgroups. Light colors allow the black labels to show up. Default pattern fill is solid.

```
pattern1 color=lipk;
pattern2 color=cyan;
pattern3 color=lime;
```

Produce the vertical bar chart. SUBGROUP= creates a separate bar segment for each department. INSIDE= prints the subgroup percent statistic inside each bar segment. OUTSIDE= prints the sum statistic above each bar. WIDTH= makes the bars wide enough to display the statistics. SPACE= controls the space between the bars. MAXIS= assigns the AXIS1 statement to the midpoint axis. RAXIS= assigns the AXIS2 statement to the response axis. LEGEND= assigns the LEGEND1 statement to the subgroup legend. CFRAME= specifies the color for the 3D planes.

```
proc gchart data=reflib.totals;
  format quarter roman.;
  format sales dollar8.;
  vbar3d site / sumvar=sales
          subgroup=dept
          inside=subpct
          outside=sum
          width=9
          space=4
          maxis=axis1
          raxis=axis2
          cframe=gray
          coutline=black
          legend=legend1;
run;
quit;
```

Example 5: Controlling Midpoints and Statistics in a Horizontal Bar Chart

Procedure features:

HBAR statement options:

AUTOREF
 COUTLINE=
 CLIPREF
 SUBGROUP=

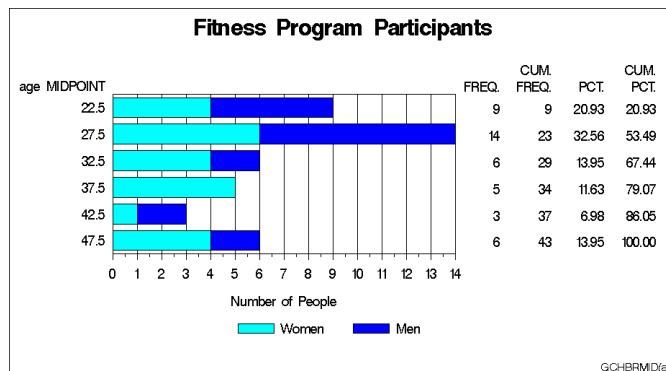
HBAR3D statement options:

FREQ
 FREQLABEL=
 MIDPOINTS=

Other features:

AXIS statement
 LEGEND statement
 PATTERN statement
 RUN-group processing

Sample library member: GCHBRMID



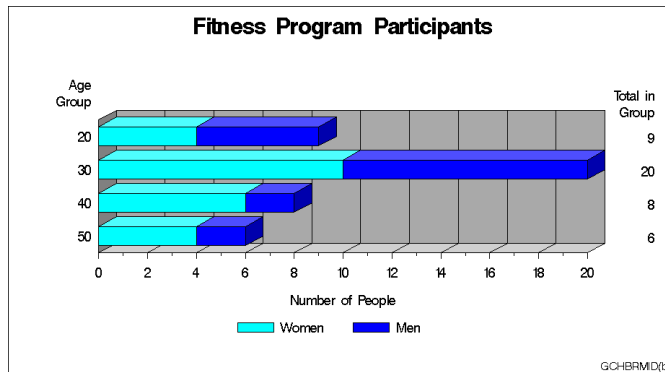
This example uses the FITNESS data set to produce a horizontal bar chart that shows the number of people in each age group in a fitness program.

It charts the numeric variable AGE, with the frequency statistic. Because the values of AGE are continuous, the procedure automatically divides the ages into ranges and displays the midpoint of each age range. The frequency statistic calculates the number of observations in each range. The chart statistic defaults to FREQ because the SUMVAR= and TYPE= options are omitted. The table of statistics displays all the statistic values.

The program also subgroups each age group bar to show the number of men and women in the group. Because the default value for the PATTERNID= option is SUBGROUP, the procedure automatically assigns a different pattern to each subgroup and the PATTERNID= option is unnecessary.

PATTERN statements specify the colors for the subgroups. Whenever the GCHART procedure uses PATTERN statements, the default outline color of the bars changes from black to the color of the bar. Because this program uses PATTERN statements, it also uses COUTLINE= to specify a black outline for the bars.

The second part of this example modifies the midpoint axis and the table of statistics, and uses RUN-group processing to produce the following chart. This part of the program specifies the midpoint value for each bar and requests only the FREQ statistic for the table.



Assign the libref and set the graphics environment. Black is the first color in the colors list and, by default, is used for all text and for the axis lines and frame. Therefore, it is not necessary to use CTEXT= (GOPTIONS statement) and CAXIS= (HBAR statement) to specify a color.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(black blue green red) ftext=swiss
          ftitle=swissb htitle=6 htext=3.5;
```

Create the data set FITNESS. FITNESS contains age and sex of participants, as well as the number of times they exercise each week and their resting heart rate and aerobic power.

```
data reflib.fitness;
  input age sex $ heart exer aero;
  datalines;
28 M 86 2 36.6
41 M 76 3 26.7
30 M 78 2 33.8
...more data lines...
29 M 54 3 44.8
48 F 66 2 28.9
36 F 66 2 33.2
;
```

Define the title and footnote.

```
title1 'Fitness Program Participants';
footnote h=3 j=r 'GCHBRMID(a) ';
```

Modify the response axis. OFFSET= moves the first and last tick marks to the ends of the axis line.

```
axis1 label=('Number of People')
      minor=(number=1)
      offset=(0,0);
```

Modify the legend. VALUE= specifies the text that describes the values.

```
legend1 label=none
       value=('Women' 'Men');
```

Define pattern colors for the subgroups. The procedure automatically assigns a pattern to each subgroup, using the default fill, SOLID, with the specified color.

```
pattern1 color=cyan;
pattern2 color=blue;
```

Produce the first horizontal bar chart. Because neither MIDPOINTS= nor DISCRETE is used, the procedure automatically selects the midpoints. SUBGROUP= divides the bars according to the values of SEX and automatically generates a legend. AUTOREF adds reference lines to the chart at each major tick mark. CLIPREF positions the reference lines behind the bars. COUTLINE= specifies the outline color for the bars.

```
proc gchart data=reflib.fitness;
  hbar age / subgroup=sex
        legend=legend1
        autoref
        clipref
        coutline=black
        raxis=axis1;
run;
```

Define the footnote for the second chart.

```
footnote h=3 j=r 'GCHBRMID(b) ';
```

Modify the response axis for the second chart. ORDER= places major tick marks on the response axis at intervals of 2.

```
axis1 order=(0 to 20 by 2)
      label=('Number of People')
      minor=(number=1)
      offset=(0,0);
```

Modify the midpoint axis label for the second chart.

```
axis2 label=('Age ' j=r 'Group');
```

Produce the second horizontal bar chart with modified midpoints. MIDPOINTS= specifies the middle value of the range of values represented by each bar. FREQ requests that only the frequency statistic appears in the table. FREQLABEL= specifies the text for the column header in the table of statistics.

```

hbar3d age / midpoints=(20 30 40 50)
           freq
           freqlabel='Total in Group'
           subgroup=sex
           autoref
           maxis=axis2
           raxis=axis1
           legend=legend1
           coutline=black;

run;
quit;

```

Example 6: Generating Error Bars in a Horizontal Bar Chart

Procedure features:

HBAR statement options:

```

CLM=
COUTLINE=
ERRORBAR=
FREQLABEL=
MEANLABEL=
NOFRAME
SUMVAR=
TYPE=

```

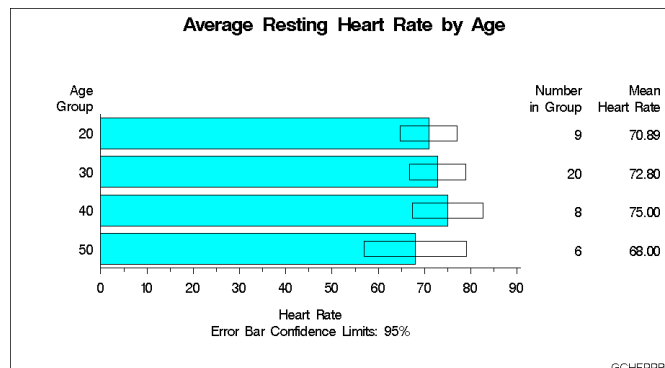
Other features:

AXIS statement

PATTERN statement

Data set: FITNESS

Sample library member: GCHERRBR



This example uses the FITNESS data set to chart the mean heart rate for each age group with error bars showing the confidence limits for the average. The response axis label describes the confidence limit for the error bars. To make the error bars easier to read, the program suppresses the frame that the procedure draws around the axis area.

Descriptive column head labels in the table of statistics replace the statistic names that appear by default.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red) ftext=swiss
        ftitle=swissb htitle=5 htext=3.5;
```

Define the title and footnote.

```
title1 'Average Resting Heart Rate by Age';
footnote h=3 j=r 'GCHERRBR ';
```

Modify the axis labels. AXIS1 is assigned to the response axis and AXIS2 is assigned to the midpoint axis.

```
axis1 label=('Heart Rate' j=c
            'Error Bar Confidence Limits: 95%')
        minor=(number=1);
axis2 label=('Age' j=r 'Group');
```

Define a color for the bars. The PATTERN statement uses the default fill, SOLID, with the specified color to create a pattern for the bars. Using a PATTERN statement causes the default bar outline color to be the same as the fill color. COUTLINE= in the HBAR statement assigns a black outline.

```
pattern1 color=cyan;
```

Produce the horizontal bar chart. SUMVAR= calculates the mean of the variable HEART for all the observations in each midpoint group. TYPE= specifies the mean statistic for the summary variable, HEART. FREQLABEL= and MEANLABEL= specify new column labels for the frequency and mean statistics. ERRORBAR= draws the error bars as empty bars and CLM= specifies the confidence level. COUTLINE= outlines the bars in black. NOFRAME suppresses the axis area frame.

```
proc gchart data=reflib.fitness;
  hbar age / type=mean
        sumvar=heart
        freqlabel='Number in Group'
        meanlabel='Mean Heart Rate'
        errorbar=bars
        clm=95
        midpoints=(20 30 40 50)
        raxis=axis1
        maxis=axis2
        noframe
        coutline=black;
run;
```

```
quit;
```

Example 7: Creating Bar Charts with Drill-down for the Web

Procedure Features:

VBAR3D statement

ODS Features:

ODS HTML statement:

```
ANCHOR=
BODY=
CONTENTS=
FRAME=
NEWFILE
NOGTITLE
PATH=
```

Other Features:

```
AXIS statement
BY statement
FORMAT statement
GOPTIONS statement
LEGEND statement
PATTERN statement
RUN-group processing
TITLE statement
WHERE statement
```

Sample library member: GCHDDOWN

This example shows how to create 3D bar charts with drill-down functionality for the Web. In addition to showing how to use the ODS HTML statement and the HTML options to create the drill-down, the example also illustrates other VBAR3D statement options.

For creating output with drill-down for the Web, the example shows how to

- explicitly name the HTML files and open and close them throughout the program
- specify names and destination for the GIF files created by the ODS HTML statement and the GIF device driver
- assign anchor names to the graphics output
- use the HTML= and HTML_LEGEND= procedure options to assign link targets
- use BY-group processing to store multiple graphs in one file or in individual files
- use incremented anchor names and incremented file names.

For more information, see “ODS HTML Statement” on page 164 in Chapter 7, “SAS/GRAPH Statements,” on page 121.

For creating 3D bar charts, the example shows how to

- group the midpoints, including patterning bars by group, modifying the group axis, adjusting the space between groups of bars
- identify midpoint values with a legend instead of labeling each bar

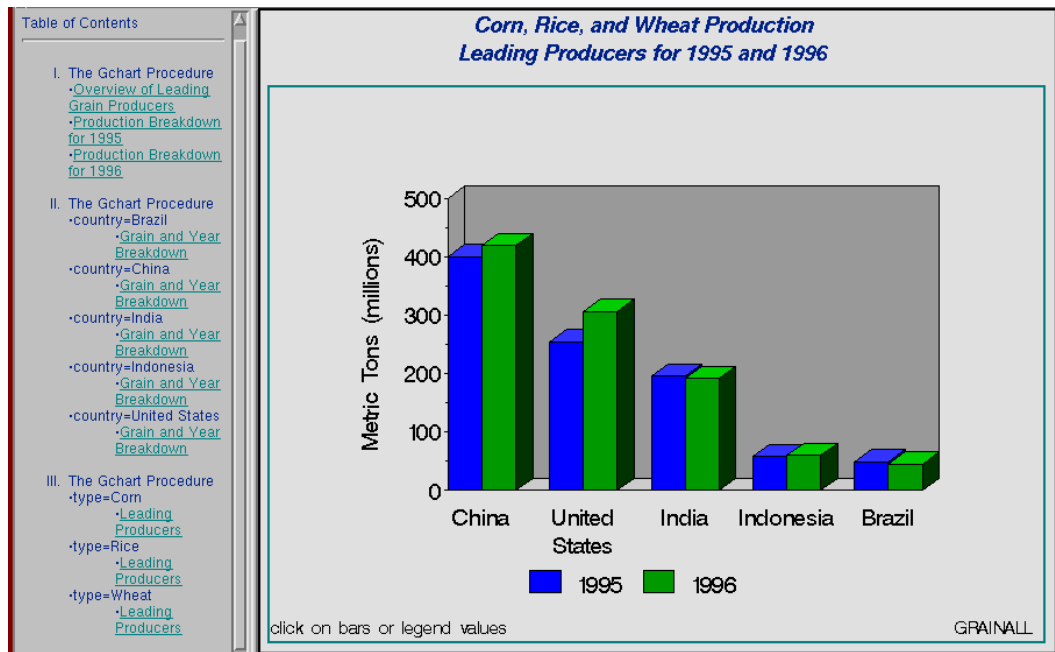
- subgroup bars
- remove an axis and its axis plane
- add reference lines.

The introduction to each part lists the VBAR3D options that it features.

The program generates twelve linked bar charts that display data about the world's leading grain producers. The data contain the amount of grain produced by five countries in 1995 and 1996. Each of these countries is one of the three leading producers of wheat, rice, or corn, worldwide.

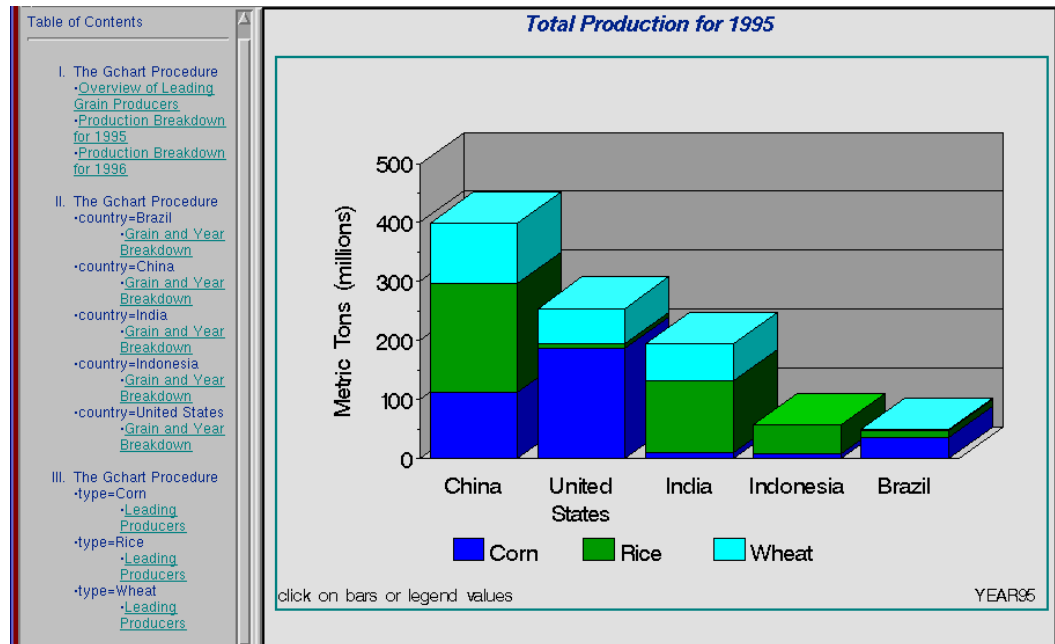
The first chart, shown in Figure 29.15 on page 857 as it appears in a browser, is an overview of the data that shows the total grain production for the five countries for both years.

Figure 29.15 Browser View of Overview Graph



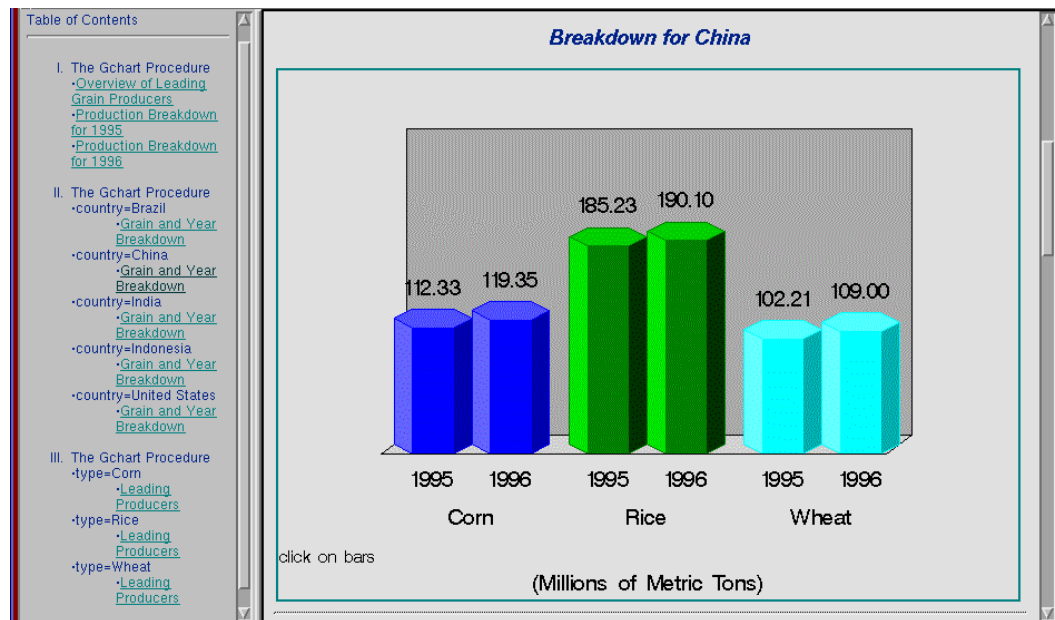
The next two charts break down grain production by year. These charts are linked to the legend values in Figure 29.15 on page 857. For example, when you select the legend value for 1995, the graph in Figure 29.16 on page 858 appears.

Figure 29.16 Browser View of Year Breakdown for 1995



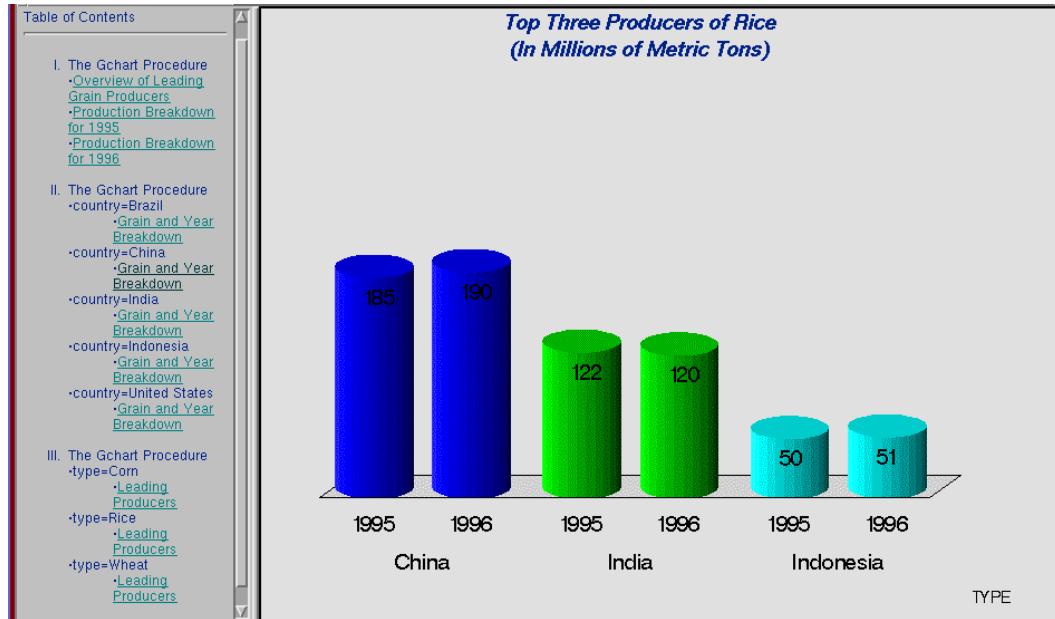
Another group of charts breaks down the data by country. These charts are linked to the bars. For example, when you drill down on the bar for China in either Figure 29.15 on page 857 or Figure 29.16 on page 858, the graph in Figure 29.17 on page 858 appears.

Figure 29.17 Browser View of Breakdown for China



Finally the data is charted by grain type. These graphs are linked to the bars in Figure 29.17 on page 858. If you select the legend value or bar for **Rice**, Figure 29.18 on page 859 appears.

Figure 29.18 Browser View of Breakdown for Rice



This program is divided into four parts:

- “Example 7, Part A” on page 859 generates the graph shown in Figure 29.15 on page 857.
- “Example 7, Part B” on page 864 generates the pair of graphs represented by Figure 29.16 on page 858.
- “Example 7, Part C” on page 866 generates the five graphs represented by Figure 29.17 on page 858.
- “Example 7, Part D” on page 868 generates the three graphs represented by Figure 29.18 on page 859.

Example 7, Part A

Features:

VBAR3D options:

DES=
 DISCRETE
 GROUP=
 GSPACE=
 HTML=
 HTML_LEGEND=
 NAME=
 SUBGROUP=

ODS HTML options:

BODY=
 CONTENTS=
 FRAME=
 GPATH=
 NOGTITLE

The first part of the program, which includes setting the graphics environment and creating the data set, does the following:

- Adds three HTML variables to the data set. The variables contain the link targets for all of the graphs that support drill-down functionality. The HREF values for the HTML variables in the data set contain this information about the link targets:
 - the name of the body file that is the target. BODY= in the ODS HTML statement names the body file.
 - the anchor name of the output if the target file contains more than one graph. By default, all output is assigned a unique anchor name unless you specify a name with ANCHOR= in the ODS HTML statement.
- Opens the HTML destination for the frame and contents files and the first body file.
- Creates one grouped 3D vertical bar chart (shown in Figure 29.15 on page 857) with drill-down on the bars and legend values. The bars, which represent total production for each year for each country, are grouped and labeled by COUNTRY. Instead of displaying the year below each bar, the program suppresses the midpoint values with an AXIS statement and creates a legend that associates bar color and year. To create the legend, the chart variable YEAR is assigned to the SUBGROUP= option. Because the chart variable and the subgroup variable are the same, each bar contains only one "subgroup." As a result, the subgroup legend has an entry for each value of YEAR, thereby creating a legend for the midpoints. The values of COUNTRY label each group of bars.
- Assigns the HTML variables that contain link information for the bars and for the legend values to the HTML= and HTML_LEGEND= options, respectively.

Assign the Web-server path. FILENAME assigns the fileref ODSOUT, which specifies a destination for the HTML and GIF files produced by the example program. To assign that location as the HTML destination for program output, ODSOUT is specified later in the program on the ODS HTML statement's PATH= option. ODSOUT must point to a Web-server location if procedure output is to be viewed on the Web.

```
filename odsout 'path-to-Web-server-space';
```

Close the ODS listing destination for procedure output, and set the graphics environment. To conserve system resources, the graphics output is not displayed in the GRAPH window, although it is written to the graphics catalog and to the GIF files.

```
ods listing close;
goptions reset=global gunit=pct
         htitle=6 htext=4 ftitle=zapfb ftext=swiss;
```

Create the data set GRAINLDR. GRAINLDR contains data about grain production in five countries for 1995 and 1996. The quantities in AMOUNT are in thousands of metric tons. MEGTONS converts these quantities to millions of metric tons.

```
data grainldr;
  length country $ 3 type $ 5;
  input year country $ type $ amount;
  megtons=amount/1000;
```

```

datalines;
1995 BRZ  Wheat    1516
1995 BRZ  Rice     11236
1995 BRZ  Corn     36276
1995 CHN  Wheat   102207
1995 CHN  Rice    185226
1995 CHN  Corn    112331
1995 IND  Wheat    63007
1995 IND  Rice    122372
1995 IND  Corn     9800
1995 INS  Wheat    .
1995 INS  Rice    49860
1995 INS  Corn     8223
1995 USA  Wheat   59494
1995 USA  Rice     7888
1995 USA  Corn   187300
1996 BRZ  Wheat    3302
1996 BRZ  Rice     10035
1996 BRZ  Corn    31975
1996 CHN  Wheat   109000
1996 CHN  Rice    190100
1996 CHN  Corn    119350
1996 IND  Wheat    62620
1996 IND  Rice    120012
1996 IND  Corn     8660
1996 INS  Wheat    .
1996 INS  Rice    51165
1996 INS  Corn     8925
1996 USA  Wheat   62099
1996 USA  Rice     7771
1996 USA  Corn   236064
;

```

Add three HTML variables to GRAINLDR to create the NEWGRAIN data set. Each HTML variable is assigned the targets for a certain variable value. These targets are specified by the HREF attribute within an AREA element in the HTML file. Each HREF value specifies the HTML body file and, optionally, the name of the anchor within the body file that identifies the target graph. The HTML variable YEARDRILL contains the targets for the values of the variable YEAR.

```

data newgrain;
  set grainldr;
  length yeardrill typedrill countrydrill $ 40;
  if year=1995 then
    yeardrill='HREF="year95_body.html"';
  else if year=1996 then
    yeardrill='HREF="year96_body.html"';

```

The HTML variable COUNTRYDRILL contains the targets for the values of the variable COUNTRY. Because the graphs of COUNTRY are in one file, the targets must include the anchor name.

```

if country='BRZ' then
  countrydrill='HREF="country_body.html#country"';
else if country='CHN' then
  countrydrill='HREF="country_body.html#country1"';
else if country='IND' then
  countrydrill='HREF="country_body.html#country2"';
else if country='INS' then
  countrydrill='HREF="country_body.html#country3"';
else if country='USA' then
  countrydrill='HREF="country_body.html#country4"';

```

The HTML variable TYPEDRILL contains the names of the files that are the targets for the values of the variable TYPE.

```

if type='Corn' then
  typedrill='HREF="type1_body.html"';
else if type='Rice' then
  typedrill='HREF="type2_body.html"';
else if type='Wheat' then
  typedrill='HREF="type3_body.html"';
run;

```

Create a format for the values of COUNTRY.

```

proc format;
  value $country 'BRZ' = 'Brazil'
                'CHN' = 'China'
                'IND' = 'India'
                'INS' = 'Indonesia'
                'USA' = 'United States';
run;

```

Define pattern colors for all graphs. To avoid solid black patterns (BLACK is the first color in the colors list), explicitly assign the pattern colors.

```

pattern1 color=blue;
pattern2 color=green;
pattern3 color=cyan;

```

Define legend characteristics for all legends. OFFSET= moves the legend down.

```

legend1 label=none
  shape=bar(4,4)
  position=(bottom center)
  offset=(-3);

```

Assign the GOPTIONS for ODS HTML destination. DEVICE= generates the SAS/GRAPH output as GIF files. TRANSPARENCY makes the background of the graphs the same as the Web-page background. NOBORDER turns off the border around the graphics output area.

```
goptions transparency device=gif noborder;
```

Open the ODS HTML destination for the ODS graphics output. BODY= names the file for storing the HTML output. CONTENTS= names the HTML file that contains the table of contents to the HTML procedure output. The contents file links to each of the body files written to the HTML destination. FRAME= names the HTML file that integrates the contents and body files. PATH= specifies the ODSOUT fileref as the HTML destination for all the HTML and GIF files. NOGTITLE suppress the graph titles from the SAS/GRAPH output and displays them through the HTML page.

```
ods html body='grain_body.html'
        frame='grain_frame.html'
        contents='grain_contents.html'
        path=odsout
        nogtitle;
```

Suppress the label and values for the midpoint axis. The midpoint values 1995 and 1996 do not appear below each bar.

```
axis1 label=none value=none;
```

Modify the response axis. ANGLE=90 prints the axis label vertically.

```
axis2 label=(angle=90 'Metric Tons (millions)')
      minor=(n=1)
      order=(0 to 500 by 100)
      offset=(0,0);
```

Suppress the label and order the values for the group axis. Because the values of COUNTRY are formatted, ORDER= must specify their formatted value.

```
axis3 label=none
      order=('China' 'United States' 'India'
            'Indonesia' 'Brazil')
      split=' ';
```

Define titles and footnote. The footnote uses the catalog entry name to identify the graph.

```
title1 'Corn, Rice, and Wheat Production';
title2 h=2 'Leading Producers for 1995 and 1996';
footnote1 j=1 h=3 'click on bars or legend values' j=r h=3 'GRAINALL ';
```

Generate the vertical bar chart that summarizes all grain production for all countries for both years. DISCRETE creates a separate bar for each unique value of YEAR. GROUP= groups the bars by country. To create a legend for midpoint values, SUBGROUP= is assigned the chart variable. GSPACE= controls the space between the groups of bars.

```

proc gchart data=newgrain;
  format country $country.;
  vbar3d year / discrete
          sumvar=megtons
          group=country
          subgroup=year
          legend=legend1
          space=0
          width=4
          gspace=5
          maxis=axis1
          raxis=axis2
          gaxis=axis3
          cframe=grayaa
          coutline=black

```

HTML= specifies COUNTRYDRILL as the variable that contains the targets for the bars. HTML_LEGEND= specifies YEARDRILL as the variable that contains the targets for the legend values. Specifying HTML variables causes SAS/GRAPH to add an image map to the HTML body file. NAME= specifies the name of the catalog entry. Because the PATH= destination is a file storage location and not a specific file name, the catalog entry name GRAINALL is automatically assigned to the GIF file. DES= specifies the description that is stored in the graphics catalog and used in the Table of Contents.

```

          html=countrydrill
          html_legend=yeardrill
          name='grainall'
          des='Overview of leading grain producers';
run;

```

Example 7, Part B

Features: VBAR3D options:
 AUTOREF
 HTML=
 HTML_LEGEND=
 SUBGROUP=
 SPACE=
 NAME=
 ODS HTML options:
 BODY=

In the second part, the PROC GCHART step continues, using RUN-group processing and WHERE statements to produce two graphs of grain production for each year, one of which is shown in Figure 29.16 on page 858. Each bar represents a country and is subgrouped by grain type. As before, both the bars and the legend values are links to other graphs. The bars link to targets stored in COUNTRYDRILL and the legend values link to targets in TYPEDRILL. These two graphs not only *contain* links, they *are* the link targets for the legend values in Figure 29.15 on page 857. Before each graph is generated, the ODS HTML statement opens a new body file in which to store the output. Because each of these graphs is stored in a separate file, the HREF attributes that are stored in the variable YEARDRILL point only to the file. The name of the file

is specified by the BODY= option in the ODS HTML statement. For example, this is the HREF attribute that points to the graph of 1995 and is stored in the variable YEARDRILL:

```
    HREF=year95_body.html
```

YEARDRILL is assigned to the HTML_LEGEND= option in Part A.

Open a new body file for the graph of 1995 production. Assigning a new body file closes GRAIN_BODY.HTML. The contents and frame files, which remain open, will provide links to all body files.

```
ods html body='year95_body.html' path=odsout;
```

Define the title and footnote for the chart.

```
title1 'Total Production for 1995';
footnote1 j=1 h=3 'click on bars or legend values' j=r h=3 'YEAR95 ';
```

Subset the data for 1995 and generate the vertical bar chart for 1995. AUTOREF draws a reference line on the backplane for every major tick mark value. SUBGROUP= creates a separate bar segment for each department. SPACE= controls the space between the bars. HTML= names the variable that contains the targets for the bars. HTML_LEGEND= names the variable that contains the targets for the legend values. The GIF files use the catalog entry name specified by NAME=.

```
where year=1995;
vbar3d country / sumvar=megtons
              subgroup=type
              autoref
              html=countrydrill
              html_legend=typedrill
              legend=legend1
              cframe=grayaa
              space=3
              coutline=black
              maxis=axis3
              raxis=axis2
              name='year95'
              des='Production Breakdown for 1995';

run;
```

Open a new body file for the graph of 1996 production. Assigning a new body file closes YEAR95_BODY.HTML.

```
ods html body='year96_body.html' path=odsout;
```

Define title and footnote for the second graph.

```
title1 'Total Production for 1996';
footnote1 j=1 h=3 'click on bars or legend values' j=r h=3 'YEAR96 ';
```

Subset the data for 1996 and generate the vertical bar chart for 1996.

```
where year=1996;
vbar3d country / sumvar=megtons
              subgroup=type
```

```

                                autoref
                                html=countrydrill
                                html_legend=typedrill
                                legend=legend1
                                cframe=grayaa
                                space=3
                                coutline=black
                                maxis=axis3
                                raxis=axis2
                                name='year96'
                                des='Production Breakdown for 1996';

run;
quit;

```

Example 7, Part C

Features: VBAR3D options:

```

DES=
GAXIS=
GROUP=
HTML=
NAME=
OUTSIDE=
PATTERNID=
RAXIS=
SHAPE=

```

ODS HTML options:

```

BODY=
ANCHOR=

```

The third part produces the five graphs that show the breakdowns by country. These graphs are generated with BY-group processing and are all stored in one body file. When the file is displayed in the browser, all the graphs appear in one frame that can be scrolled. Because the graphs are stored in one file, the links to them must explicitly point to the location of each graph in the file, not just to the file. This location is defined by an anchor. ODS HTML assigns anchor names by default, but you can specify anchor names with the ANCHOR= option. When the procedure uses BY-group processing to generate multiple pieces of output, ODS automatically increments the anchor name to produce a unique name for each graph. This example assigns the base name {mono country} to ANCHOR=. The graphs created by this part are referenced by the COUNTRYDRILL variable. With BY-group processing the catalog entry name also increments automatically. NAME= specifies COUNTRY as the base name for the graphics output. Because you cannot specify a different description for each graph, DES= specifies a generic description for the HTML Table of Contents.

Sort the data set for the graphs of production by country.The data must be sorted in order of the BY variable before running PROC GCHART with BY-group processing.

```

proc sort data=newgrain out=country;
by country;
run;

```


Open a new body file and specify the base anchor name for the graphs of individual countries. Assigning a new body file closes YEAR96_BODY.HTML. Because all the graphs generated by the BY-group processing are stored in one file, each one is automatically assigned an anchor name. ANCHOR= specifies a base name for these anchors.

```
ods html body='country_body.html'
      anchor='country'
      gfootnote
      path=odsout;
```

Redefine AXIS2 to change the range of values and suppress all axis elements. Setting all the label and tick mark options to NONE and assigning a line style of 0 removes the response axis. NOPLANE removes the 3D axis plane. Specifying ORDER= makes all the graphs use the same range of values.

```
axis2 order=(0 to 250 by 50)
      label=none
      value=none
      style=0
      major=none
      minor=none
      noplane;
```

Suppress the axis label for the midpoint axis.

```
axis4 label=none;
```

Suppress the default BY-line and define a title that includes the BY-value. #BYVAL inserts the value of the BY variable COUNTRY into the title of each report.

```
options nobyline;
title1 'Breakdown for #byval(country)';
footnote1 j=1 h=3 'click on bars';
footnote2 j=c '(Millions of Metric Tons)';
```

Generate the vertical bar chart of production for each country. GROUP= groups the bars by country. PATTERNID= assigns patterns by group value. SHAPE= assigns the bar shape. OUTSIDE= displays the SUM statistic above the bars. RAXIS= assigns the AXIS statement that removes all axis elements. GAXIS= assigns the AXIS statement that removes the label. HTML= specifies TYPEDRILL as the variable that contains the targets for the bars. NAME= specifies the name of the catalog entry. The graphics catalog entry name increments so the GIF files are named sequentially from COUNTRY to COUNTRY4. The DES= option specifies a general description that appears in the table of contents for all five graphs.

```
proc gchart data=country;
  format country $country.;
  by country;
  vbar3d year / discrete
        sumvar=megtons
        patternid=group
```

```

                                group=type
                                shape=hexagon
                                outside=sum
                                html=typedrill
                                width=9
                                gspace=3
                                space=0
                                cframe=grayaa
                                raxis=axis2
                                gaxis=axis4
                                maxis=axis4
                                name='country'
                                des='Grain and Year Breakdown';
run;

```

Example 7, Part D

Features: VBAR3D options:
 INSIDE=
 NOZERO

ODS HTML options:
 BODY=
 NEWFILE=TABLE

Like Part C, this part uses BY-group processing to generate three graphs that show the three leading producers for each type of grain. The program subsets the data and suppresses midpoints with no observations. Instead of storing all of the output in one body file, it stores each graph in a separate file. To do this, the program uses the ODS HTML option NEWFILE=TABLE. When NEWFILE=TABLE is used with BY-group processing, each new piece of output automatically generates a new body file and simply increments the name of the file that is specified by BODY=. Because each graph is stored in a separate file, the links to these graphs reference only the file name and do not require an anchor name. The graphs created by this part are referenced by the TYPEDRILL variable.

Sort the data set for the graphs of leading producers of each grain type.

```

proc sort data=grainldr out=type;
by type;
run;

```

Open a new body file. Assigning a new body file closes COUNTRY_BODY.HTML. NEWFILE=TABLE opens a new body file for each piece of output generated by the procedure. Each new file increments the name specified by BODY= using the number within the body file name as the starting number.

```

ods html body='type1_body.html'
newfile=table
path=odsout;

```

Modify the group axis. Because SPLIT= assigns a blank as the split character, the value **United States** automatically prints on two lines.

```
axis5 label=none
      split=' ';
```

Define title and footnote. #BYVAL inserts the value of the BY variable TYPE into the title of each report.

```
title1 'Top Three Producers of #byval(type)';
title2 '(In Millions of Metric Tons)';
footnote j=r h=3 'TYPE ';
```

Generate the vertical bar chart of leading producers for each grain type. BY-group processing generates a separate graph for each value TYPE. Each new graph generates a new body file. NOZERO suppresses the midpoints that do not have any observations. INSIDE= displays the SUM statistic inside the bars.

```
proc gchart data=type (where=(megtons gt 31));
  format country $country.;
  by type;
  vbar3d year / discrete
          sumvar=megtons
          group=country
          nozero
          shape=cylinder
          noframe
          patternid=group
          inside=sum
          width=8
          maxis=axis4
          raxis=axis2
          gaxis=axis5
          cframe=grayaa
          name='type'
          des='Leading Producers';

run;
quit;
```

Close the ODS HTML destination, and open the ODS Listing destination. You must close the HTML destination before you can view the output with a browser.

```
ods html close;
ods listing;
```

Example 8: Specifying the Sum Statistic for a Pie Chart

Procedure features:

PIE statement options:

COUTLINE=

SUMVAR=

PIE3D statement options:

COUTLINE=

EXPLODE=

SUMVAR=

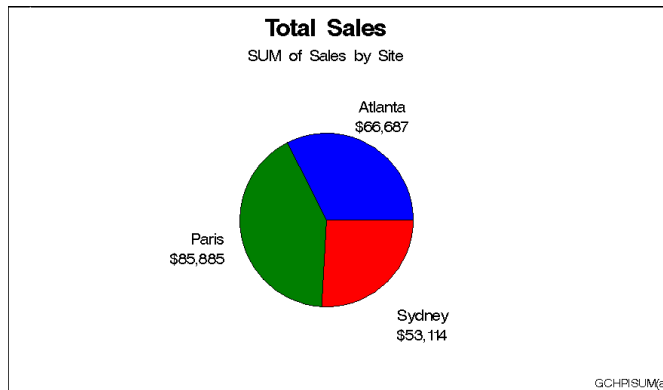
Other features:

FORMAT statement

RUN-group processing

Data set: TOTALS

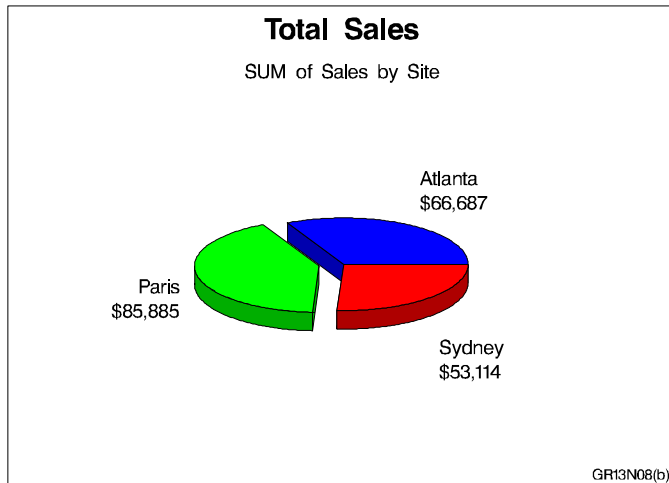
Sample library member: GCHPISUM



This example produces two pie charts that show total sales for three sites by charting the values of the character variable SITE and calculating the sum of the variable SALES for each site. It represents the statistics as slices of the pie. By default, the midpoint value and the summary statistic are printed beside each slice.

The pie slices use the default pattern fill, which is solid. Because a colors list is specified in the GOPTIONS statement, the default solid patterns rotate through the colors in the list, beginning with the first color, blue. Each slice displays a different color because, by default, pie charts are patterned by midpoint. Because the default outline color is also the first color in the list, the example uses the COUTLINE= option to assign black to the outlines.

The second pie chart is a 3D pie chart with an exploded slice, as shown in the following output.



Assign the libref and set the graphics environment. CTEXT= specifies the color for all text on the output. COLORS= specifies the colors list, which is used by the default patterns and outlines.

```
libname reflib 'SAS-data-library';
options reset=global gunit=pct border cback=white
        colors=(blue green red) ctext=black
        ftitle=swissb ftext=swiss htitle=6 htext=4;
```

Define title and footnote.

```
title 'Total Sales';
footnote j=r 'GCHPISUM(a) ';
```

Produce the first pie chart. The PIE statement produces a two-dimensional pie chart. SUMVAR= calculates the sum of SALES for each value of the chart variable SITE. The default statistic for SUMVAR= is SUM. The summary variable SALES is assigned a dollar format. COUTLINE= specifies the outline color for the slices.

```
proc gchart data=reflib.totals;
  format sales dollar8.;
  pie site / sumvar=sales
        coutline=black;
run;
```

Define footnote for second pie chart.

```
footnote j=r 'GCHPISUM(b) ';
```

Produce the second pie chart. The PIE3D statement produces a three-dimensional pie chart. EXPLODE= separates the slice for Paris from the rest of the pie.

```

pie3d site / sumvar=sales
           coutline=black
           explode='Paris';

run;
quit;

```

Example 9: Subgrouping a Donut or Pie Chart

Procedure features:

DONUT statement options:

```

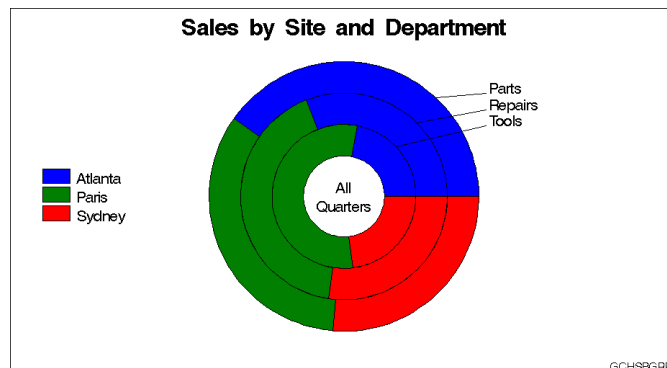
COUTLINE=
CTEXT=
DONUTPCT=
LABEL=
LEGEND=
NOHEADING
SUBGROUP=

```

Other features: LEGEND statement

Data set: TOTALS

Sample library member: GCHSBGRP



This example produces a donut chart that is similar to the pie chart in Example 8 on page 869 in that each slice represents total sales for a site and each slice is a different color. However, in this donut chart the sites are subgrouped by department, so that each department is represented as a concentric ring with slices.

Subgrouping suppresses the chart statistic and the midpoint labels. Instead it automatically labels the rings with the subgroup values and generates a legend that shows how the patterns are associated with the midpoint values. Subgrouping a pie chart produces the same results but without the hole in the center.

To allow the donut chart to be as large as possible, the program suppresses the default heading and moves the legend into the space at the left of the chart.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(blue green red) ctext=black
          ftext=swissb ftext=swiss htitle=6 htext=4;
```

Define title and footnote.

```
title 'Sales by Site and Department';
footnote h=3 j=r 'GCHSBGRP ';
```

Modify the subgroup legend. LABEL= suppresses the legend label. SHAPE= defines the shape of the legend values. POSITION=, OFFSET=, and ACROSS= arrange the legend entries in a column to the left of the pie chart. MODE= allows the legend to occupy the procedure output area.

```
legend1 label=none
        shape=bar(4,4)
        position=(middle left)
        offset=(5,)
        across=1
        mode=share;
```

Produce the donut chart. SUBGROUP= divides the donut into rings. Each ring represents a value of the subgroup variable, DEPT. The DONUTPCT= option controls the size of the donut hole, which contains the text specified by LABEL=. The NOHEADING option suppresses the default heading that contains the name of the chart variable and the type of statistic. LEGEND= assigns the LEGEND1 statement to the chart COUTLINE= specifies the outline color for the slices and subgroup rings. CTEXT= specifies the color used by the donut label and by the subgroup arrows.

```
proc gchart data=reflib.totals;
  format sales dollar8.;
  donut site / sumvar=sales
        subgroup=dept
        donutpct=30
        label=('All' justify=center 'Quarters')
        noheading
        legend=legend1
        coutline=black
        ctext=black;

run;
quit;
```

Example 10: Ordering and Labeling Slices in a Pie Chart

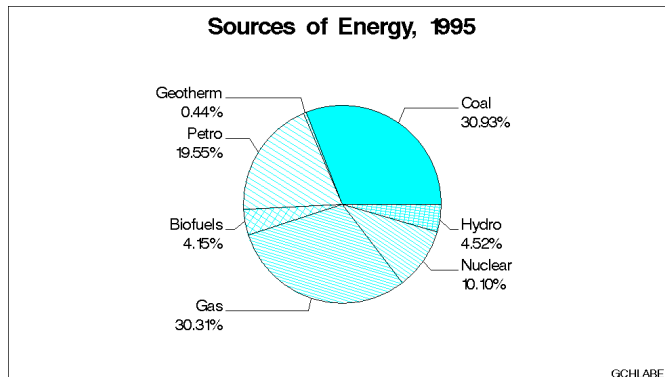
Procedure features:

PIE statement options:

CFILL=

MIDPOINTS=
 PERCENT=ARROW
 SLICE=ARROW
 SUBGROUP=
 VALUE=NONE

Sample library member: GCHLABEL



This example produces a pie chart of sources of energy production for 1995. The labeled slices represent the percent of total production for each source. Instead of the sum statistic, each slice displays the percent each midpoint contributes to the whole pie. Arrows connect the midpoint labels to the slices, which are arranged by the MIDPOINTS= option so that the small slices are not next to each other and their labels do not overlap.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(black blue green red cyan lime gray)
          ftext=swiss ftitle=swissb htitle=6 htext=4;
```

Create the data set ENPROD.

ENPROD contains the amount of energy produced (PRODUCED) for seven sources (ENGYTYPE) for two years (YEAR). The amounts of energy produced are in quadrillion btu.

```
data reflib.enprod;
  input @1 year 4. @6 engytype $8. @16 produced 5.2;
  datalines;
1985 Coal      19.33
1985 Gas       19.22
1985 Petro     18.99
...more data lines...
1995 Hydro     3.21
1995 Geotherm .31
1995 Biofuels  2.95
;
```


Define title and footnote.

```
title 'Sources of Energy, 1995';
footnote h=3 j=r 'GCHLABEL ';
```

Produce the pie chart. The WHERE data set option subsets the data for 1995. OTHER=0 specifies that all midpoints, no matter how small, display a slice. MIDPOINTS= assigns the order of the slices. Each slice displays the percent contribution to total production and the slice name. VALUE=NONE suppresses the chart statistic. Both SLICE= and PERCENT= are assigned the ARROW labeling method to point to the slice, but only one arrow line is displayed. CFILL= specifies a color for the fill used by all slices.

```
proc gchart data=reflib.enprod (where=(year=1995));
  pie engtype / sumvar=produced
    other=0
    midpoints='Coal' 'Geotherm' 'Petro'
              'Biofuels' 'Gas' 'Nuclear'
              'Hydro'
    value=none
    percent=arrow
    slice=arrow
    cfill=cyan
    noheading;
run;
quit;
```

Example 11: Assigning Patterns and Identifying Midpoints with a Legend

Procedure features:

PIE statement options:

```
COUTLINE=
CTEXT=
DESCENDING
LEGEND=
OTHER=
OTHERLABEL=
VALUE=INSIDE
```

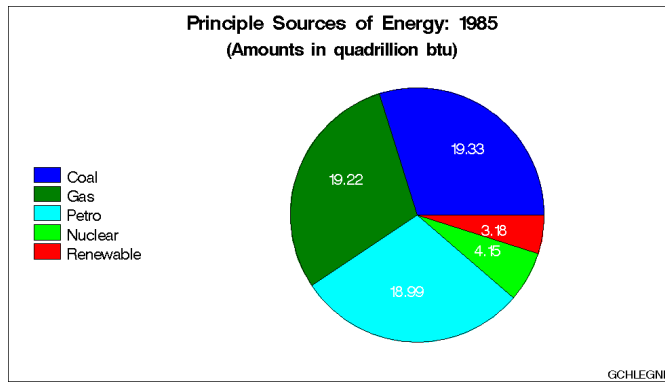
Other features:

LEGEND statement

PATTERN statement

Data set: ENPROD

Sample library member: GCHLEGND



This example shows the actual amount of energy that is produced by each source for 1985. It displays the chart statistic inside each slice and uses a legend instead of slice labels to identify the slices. Pattern colors are assigned explicitly to each energy source.

All of the variables with midpoint values less than or equal to 5 percent of the total (in this case, *biofuels*, *geotherm*, and *hydro*) are grouped into one slice labeled "Other." The slices are ordered from largest to smallest based on the statistic value. Although the "Other" slice is always last, it is in this case also the smallest.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(black blue green red) ftitle=swissb
          ftext=swiss htitle=5 htext=4;
```

Define a title and footnote.

```
title1 'Principle Sources of Energy: 1985';
title2 font=swissb h=4.5 '(Amounts in quadrillion btu)';
footnote h=3 j=r 'GCHLEGND ';
```

Define pattern colors. Each value of the chart variable ENGYTYPE is assigned a pattern whether or not it is displayed as a separate slice. Patterns are assigned to midpoints in the order the values appear in the data set. Because ENGYTYPE is character, the patterns are assigned alphabetically. The eighth pattern is for the "other" slice, which is always last.

```
pattern1 color=black;      /* biofuels */
pattern2 color=blue;      /* coal */
pattern3 color=green;     /* gas */
pattern4 color=gray;      /* geothermal */
pattern5 color=lipk;      /* hydroelectric */
pattern6 color=lime;      /* nuclear */
pattern7 color=cyan;      /* petro */
pattern8 color=red;       /* other */
```

Modify the legend. LABEL= removes the legend label. VALUE= defines the color for the value labels; by default legend value color is determined by the CTEXT= option in the procedure statement. In this case, CTEXT=WHITE, so the legend statement uses the VALUE= option to override that color specification. ORDER= orders the legend values to match the slice order in the pie chart.

```
legend1 label=none
        position=(left middle)
        offset=(4,)
        across=1
        order=('Coal' 'Gas' 'Petro'
              'Nuclear' 'Renewable')
        value=(color=black)
        shape=bar(4,4);
```

Create the pie chart. OTHER= collects all the midpoints with statistic values less than or equal to 5 percent of the total into one slice. OTHERLABEL= specifies the label for the "other" slice. DESCENDING arranges the slices in descending order of the statistic value. LEGEND= displays the customized legend created in the LEGEND statement and suppresses the slice labels. VALUE= places the chart statistics inside the slices. CTEXT= specifies white for the statistic text. Because CTEXT= also affects the color of the legend text, the LEGEND statement specifies black text so that the values can be seen. Because the PATTERN statement is used, the default slice outline matches the fill color; COUTLINE= changes the outline color to black.

```
proc gchart data=reflib.enprod(where=(year=1985));
  pie engtype / sumvar=produced
                other=5
                otherlabel='Renewable'
                descending
                legend=legend1
                value=inside
                ctext=white
                coutline=black
                noheading;
run;
quit;
```

Example 12: Grouping and Arranging Pie Charts

Procedure features:

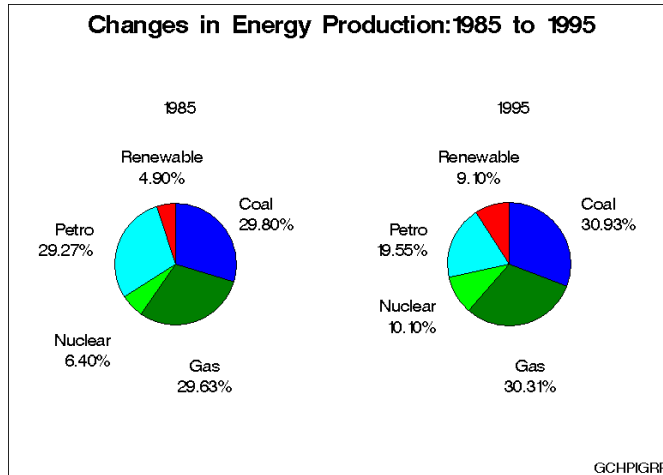
PIE statement options:

```
ACROSS=
CLOCKWISE
GROUP=
OTHER=
PERCENT=OUTSIDE
SLICE=OUTSIDE
```

Other features: PATTERN statement

Data set: ENPROD

Sample library member: GCHPIGRP



This example produces two pie charts that shows energy sources by year. Both charts are displayed on one page and are arranged two across. The program uses the **CLOCKWISE** option to arrange the slices, which begin at the 12 o'clock position and proceed clockwise in alphabetic order of the midpoint.

The chart statistic is suppressed and the midpoint label and the percent of the chart statistic are displayed outside of the slice.

A different color is defined for each energy type. The patterns are assigned in order of midpoint value. Some colors do not appear because the slices they represent are grouped into the **OTHER** slice, which is assigned the eighth color, red.

Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red) ftext=swiss
        ftext=swissb htitle=5 htext=3.5;
```

Define title and footnote.

```
title 'Changes in Energy Production:1985 to 1995';
footnote j=r 'GCHPIGRP ';
```

Define patterns for the pie slices. **PATTERN** statements define a different solid color for each midpoint value.

```
pattern1 color=black;      /* biofuels    */
pattern2 color=blue;      /* coal        */
pattern3 color=green;     /* gas         */
pattern4 color=gray;      /* geothermal  */
pattern5 color=lipk;      /* hydroelectric */
pattern6 color=lime;      /* nuclear     */
pattern7 color=cyan;      /* petro       */
pattern8 color=red;       /* other       */
```

Produce the pie charts. The WHERE= data set option selects the data for only two years. The LABEL statement suppresses the variable name, so only the YEAR value is displayed.

```
proc gchart data=reflib.enprod gout=reflib.excat;
  label year='00'x;
```

GROUP= creates a separate pie for each year. In combination with GROUP=, ACROSS= draws two charts across one page. OTHER= collects all the midpoints with statistic values less than or equal to 5 percent of the total into one slice. CLOCKWISE begins drawing the slices at the 12 o'clock position in alphabetic order of the midpoint. PERCENT=OUTSIDE and SLICE=OUTSIDE display the labels outside the slices.

```
pie engytype / sumvar=produced
  group=year
  across=2
  other=5
  otherlabel='Renewable'
  clockwise
  value=none
  slice=outside
  percent=outside
  coutline=black
  noheading;

run;
quit;
```

Example 13: Specifying the Sum Statistic in a Star Chart

Procedure features:

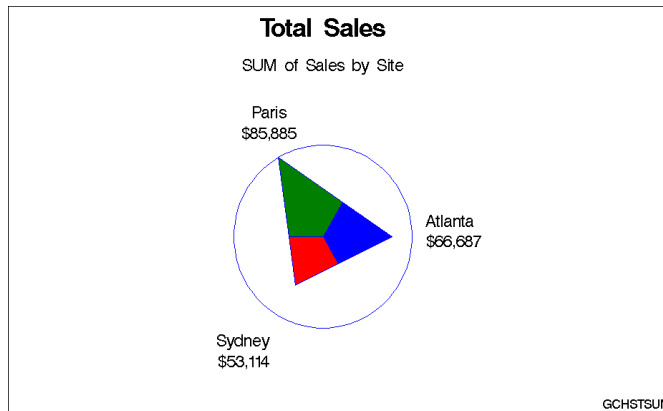
STAR statement options:

SUMVAR=

Other features: FORMAT statement

Data set: TOTALS

Sample library member: GCHSTSUM



This example produces a star chart of total sales for three sites by charting the values of the character variable `SITE` and calculating the sum of the variable `SALES` for each site. It represents the statistics as slices of the star. The center of the circle represents 0 and the edge of the circle represents the largest value, in this case Paris sales. By default, the spines are joined and filled with a solid pattern to form slices, and the midpoint value and the formatted values of the sales statistics are printed beside each slice.

By default, the circle and the slice outlines use the first color in the colors list, in this case, `BLUE`.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
         colors=(blue green red) ctext=black
         ftitle=swissb ftext=swiss htitle=6 htext=4;
```

Define title and footnote.

```
title 'Total Sales';
footnote h=3 j=r 'GCHSTSUM ';
```

Produce the star chart. `SUMVAR=` calculates the sum of `SALES` for each value of the chart variable `SITE`. Because the `TYPE=` option is omitted, the default statistic is sum. The `FORMAT` statement assigns a format to the summary variable `SALES`.

```
proc gchart data=reflib.totals;
  format sales dollar8.;
  star site / sumvar=sales;
run;
quit;
```

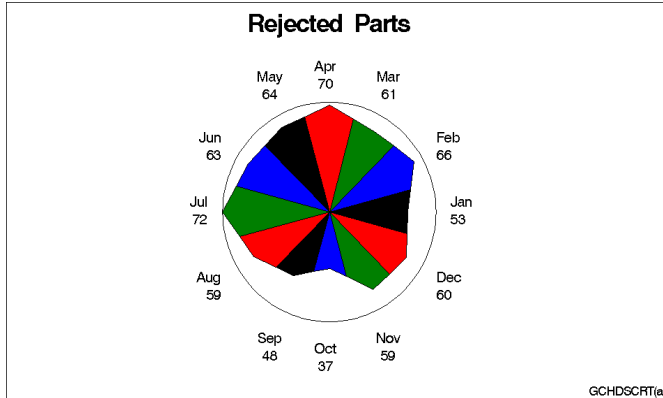
Example 14: Charting a Discrete Numeric Variable in a Star Chart

Procedure features:

STAR statement options:

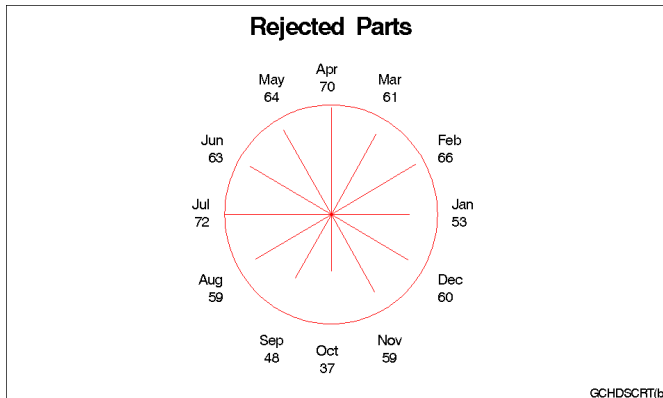
COUTLINE=
 DISCRETE
 FILL=
 NOCONNECT
 NOHEADING
 SUMVAR=

Sample library member: GCHDSCRT



This example produces two star charts that show the total number of parts that were rejected each month for a year. The STAR statement uses the DISCRETE option so that each unique value of the numeric variable DATE is a separate midpoint and has a separate spine. Each slice displays the formatted midpoint value and the chart statistic. Specifying FILL=S rotates the solid pattern through all the colors in the colors list as many times as necessary to provide patterns for all the slices.

The second chart uses the NOCONNECT option so that the chart uses spines instead of slices.



Set the graphics environment. COLORS= specifies the colors list, which is used by the default patterns and outlines.

```

goptions reset=global gunit=pct border cback=white
        colors=(black blue green red) ftext=swiss
        ftitle=swissb htext=3.5 htitle=6;

```

Create the data set REJECTS. REJECTS contains data on the number of defective parts produced at each of three sites for 12 months. BADPARTS is the number of parts that were rejected at each site for each month.

```

data rejects;
    informat date date9.;
    input site $ date badparts;
    datalines;
Sydney  01JAN1997 22
Sydney  01FEB1997 26
...more data lines...
Paris   01NOV1997 12
Paris   01DEC1997 19
;

```

Define title and footnote.

```

title 'Rejected Parts';
footnote j=r 'GCHDSCRT(a) ';

```

Produce the first star chart. DISCRETE must be specified because the numeric chart variable, DATE is assigned the WORDDATE3. Using FILL=S fills all the slices with solid patterns.

```

proc gchart data=rejects;
    format date worddate3.;
    star date / discrete
            sumvar=badparts
            noheading
            fill=s;
run;

```

Define footnote for the second chart.

```

footnote j=r 'GCHDSCRT(b) ';

```

Produce the second star chart with slices and a solid fill. NOHEADING suppresses the default heading for the star chart. NOCONNECT suppresses the lines that by default join the ends of the spines. COUTLINE= colors the spines and the circle.

```

star date / discrete
        sumvar=badparts
        noheading
        noconnect

```



```

                                coutline=red;
run;
quit;

```

Example 15: Creating a Detail Pie Chart

Procedure Features:

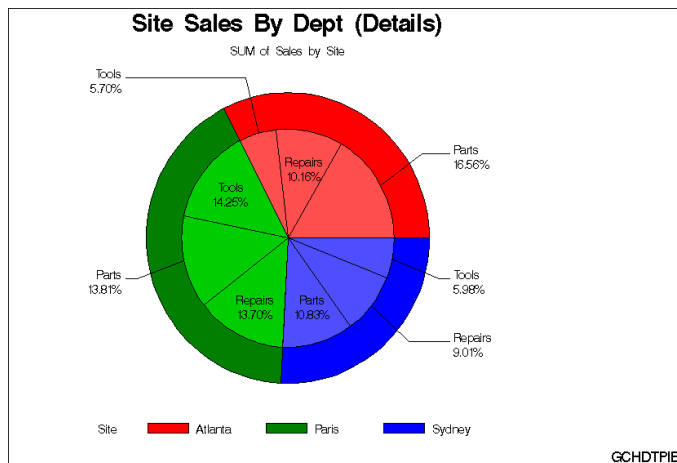
PIE statement options:

```

DETAIL=
DETAIL_PERCENT=
DETAIL_SLICE=
DETAIL_VALUE=
LEGEND
SUMVAR=

```

Sample library member: GCHDTPIE



This example produces a normal pie chart with a detail pie overlay. The pie chart shows percentages of sales for three sites by charting the values of the character variable `SITE` and calculating the percentage of the variable `SALES` for each site. The detail pie overlay shows the percentages of sales for each `DEPT` at each `SITE`. The pie slices use the colors in the default color list and the default fill, which is solid.

Set the graphics environment. The `CBACK=` option sets the background color. The `FTITLE=` and `HTITLE=` options set the font and size of the title text. The `FTEXT=` and `HTEXT=` options set the font and size of other text, such as slice labels.

```

goptions reset=all gunit=pct border cback=white
          ftitle=swissb ftext=swiss htitle=5
          htext=2.5;

```

Create the data set TOTALS. `TOTALS` contains quarterly sales data for three manufacturing sites for one year. Sales figures are broken down by department.

```

data totals;
  length Dept $ 7 Site $ 8;
  input Dept Site Quarter Sales;
  datalines;
Parts   Sydney  1 4043.97
Parts   Atlanta 1 6225.26
Parts   Paris   1 3543.97
...more data lines
Tools   Sydney  4 3817.36
Tools   Atlanta 4 4354.18
Tools   Paris   4 6511.70
;

```

Define the title and footnote.

```

title1 'Site Sales By Dept (Details)';
footnote1 h=3 j=r 'GCHDTPIE ';

```

Produce the detail pie chart. SUMVAR= calculates the sum of SALES for each value of the chart variable SITE. DETAIL= produces a inner pie overlay showing the percentage that each DEPT contributes toward each site's sales. DETAIL_PERCENT= and DETAIL_SLICE= control the positioning of the detail slice labels. DETAIL_VALUE= turns off the display of the sales values for each detail slice.

```

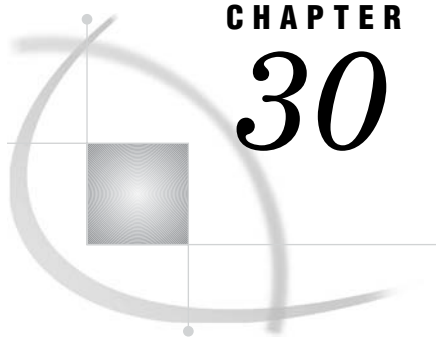
proc gchart data=totals;
  pie site / sumvar=sales
           detail=dept
           detail_percent=best
           detail_value=none
           detail_slice=best
           legend;
run;
quit;

```

References

Nelder, J. A. (1976), "A Simple Algorithm for Scaling Graphs," *Applied Statistics, Volume 25, Number 1*, London: The Royal Statistical Society.

Terrell, G. R. and Scott, D. W. (1985), "Oversmoothed Nonparametric Density Estimates," *Journal of the American Statistical Association*, 80.



CHAPTER 30

The GCONTOUR Procedure

<i>Overview</i>	885
<i>Concepts</i>	885
<i>About Contour Plots</i>	885
<i>Parts of a Contour Plot</i>	886
<i>About the Input Data Set</i>	887
<i>Interpolating Additional Values</i>	887
<i>Procedure Syntax</i>	888
<i>PROC GCONTOUR Statement</i>	888
<i>PLOT Statement</i>	889
<i>Examples</i>	904
<i>Example 1: Generating a Simple Contour Plot</i>	904
<i>Example 2: Labeling Contour Lines</i>	906
<i>Example 3: Specifying Contour Levels</i>	908
<i>Example 4: Using Patterns and Joins</i>	910
<i>References</i>	913

Overview

The GCONTOUR procedure produces plots that represent three-dimensional relationships. The colors, contours, or surface areas of a contour plot represent the values of a contour variable at each point in a plane that is formed by a dependent and an independent variable. The contour variable is applied to the Z axis of the plot, the dependent variable is applied to the X axis of the plot, and the independent variable is applied to the Y axis of the plot.

Concepts

About Contour Plots

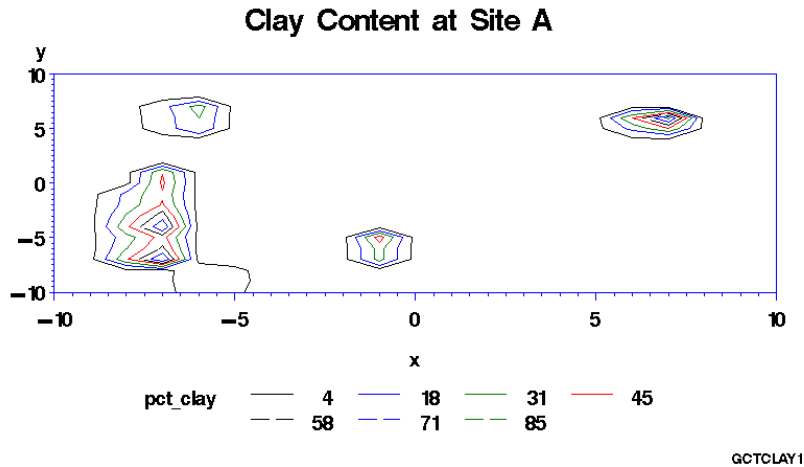
Contour plots represent the levels of magnitude of a variable z , called the *contour variable*, for a position on a plane given by the values of two variables x and y . Contour lines of different colors and line types show different levels of magnitude of z for locations of x and y .

Display 30.1 on page 886 shows a simple contour plot that illustrates the percentage of clay found in soil samples at various locations of a testing site. The x and y axes on the plot represent a graph of surface height at various x - y locations. The contour lines

within the plot represent the locations on the plane that have the clay percentages specified in the legend. The program for this plot is in Example 1 on page 904.

By default, the GCONTOUR procedure automatically scales the axes to include the maximum and minimum data values, labels each axis with the name of its variable or an associated label, and draws a frame around the plot. In addition, it plots values using seven contour levels of the contour variable, representing those levels with default colors and line types. Finally, it generates a legend that is labeled with the contour variable's name.

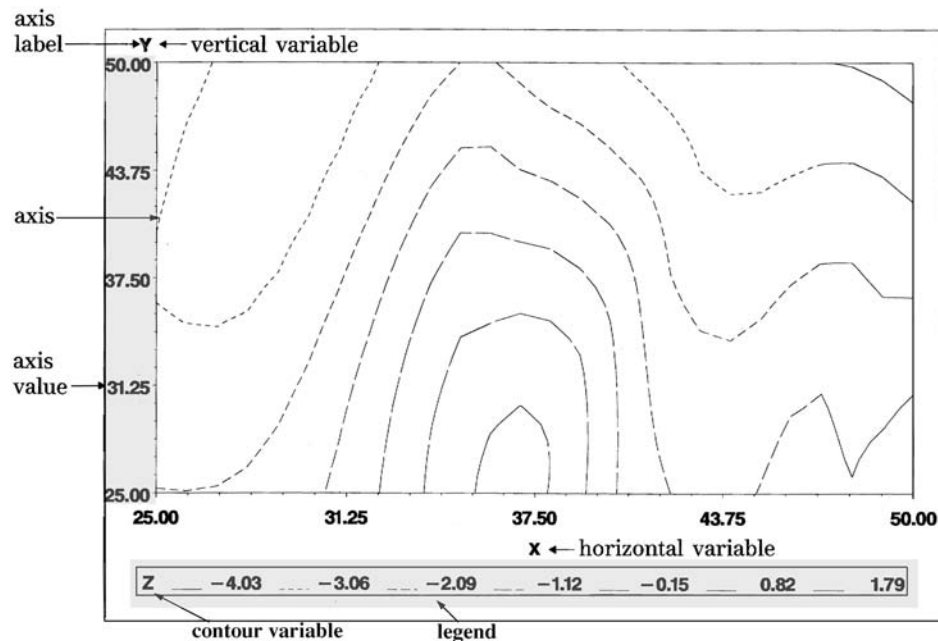
Display 30.1 Sample Contour Plot



Parts of a Contour Plot

Some of the terms used in the discussion of the GCONTOUR procedure are illustrated in Figure 30.1 on page 887.

Figure 30.1 GCONTOUR Procedure Terms



About the Input Data Set

The GCONTOUR procedure requires data sets that include three numeric variables: x , y , and z . The observations in the input data set should form a rectangular grid of x and y values and exactly one z value for each (x, y) combination. For example, data that contain 5 distinct values of x and 10 distinct values for y should be part of a data set that contains 50 observations with values for x , y , and z . If a single (x, y) grid location has more than one associated z value, only the last such observation appears in the plot.

Interpolating Additional Values

By default, the data set must contain values for the z variable for at least 50 percent of the grid in order for the GCONTOUR procedure to produce a satisfactory plot. If your data are clustered in relatively small patches over a larger study area, you can use the PROC GCONTOUR statement's INCOMPLETE option, which allows plotting of data when you have values for the z variable for less than 50 percent of the plot grid.

When the GCONTOUR procedure cannot produce a satisfactory contour plot because of missing values, the SAS/GRAPH software issues an error message, and no graph is produced. To correct this problem, you can use the G3GRID procedure to process data sets to be used by the GCONTOUR procedure. The G3GRID procedure interpolates the necessary values to produce a data set with nonmissing values of the z variable values for every combination of the x and y variables. The G3GRID procedure can also smooth data for use with the GCONTOUR procedure. For details, see Chapter 47, "The G3GRID Procedure," on page 1327.

You can use the output data set from the G3GRID procedure as the input data set for the GCONTOUR procedure. For an example of using PROC G3GRID to interpolate values, see Example 1 on page 904.

Procedure Syntax

Requirements: At least one PLOT statement is required.

Global statements: AXIS, FOOTNOTE, GOPTIONS, LEGEND, PATTERN, SYMBOL, TITLE

Reminder: The procedure can include the BY, FORMAT, LABEL, NOTE, and WHERE statements.

Supports: Output Delivery System (ODS)

```
PROC GCONTOUR <DATA=input-data-set>
    <ANNOTATE=Annotate-data-set>
    <GOUT=<libref.>output-catalog>
    <INCOMPLETE>;
PLOT plot-request </option(s)>;
```

PROC GCONTOUR Statement

Identifies the data set that contains the plot variables. Optionally specifies annotation and an output catalog.

Requirements: An input data set is required.

```
PROC GCONTOUR <DATA=input-data-set>
    <ANNOTATE=Annotate-data-set>
    <GOUT=<libref.>output-catalog>
    <INCOMPLETE>;
```

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate all graphs produced by the GCONTOUR procedure. To annotate individual graphs, use ANNOTATE= in the action statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

DATA=*input-data-set*

identifies the SAS data set that contains the variables to plot. By default, the procedure uses the most recently created SAS data set.

See also: “SAS Data Sets” on page 29 and “About the Input Data Set” on page 887.

GOUT=<*libref.*>*output-catalog*

specifies the SAS catalog in which to save the graphics output produced by the GCONTOUR procedure. If you omit the libref, the SAS/GRAPH software looks for the catalog in the temporary library called WORK and creates the WORK catalog if it does not exist.

See also: “Creating and Specifying Catalogs” on page 54

Not supported by: Java, ActiveX

INCOMPLETE

allows plotting of data when values are missing for more than half of the variables in the data set.

Not supported by: Java, ActiveX

PLOT Statement

Creates contour plots using values of three numeric variables from the input data set as the source of the contour coordinates.

Requirements: A plot request is required.

Global statements: AXIS, FOOTNOTE, LEGEND, PATTERN, SYMBOL, TITLE

Description

The PLOT statement specifies the three variables to plot. Optionally, it controls the contour levels, labels the plot lines, and modifies axes as well as the general appearance of the graph. Only one plot request can be specified in a PLOT statement. To specify multiple plots for a single PROC GCONTOUR statement, use multiple PLOT statements.

The PLOT statement automatically

- plots the values using seven contour levels of the z variable
- scales the axes to include the maximum and minimum data values
- labels the x and y axes and displays the contour levels in the plot's legend
- draws a frame around the plot.

You can use global statements to modify the axes, the legend, the contour lines and contour line labels, and the fill patterns and pattern colors for contour areas. You can also add titles, footnotes, and notes to the chart, and you can use an Annotate data set to enhance the appearance of the chart.

PLOT *plot-request* </option(s)>;

Required Arguments

y*x=z

specifies three numeric variables from the input data set:

- | | |
|-----|--|
| y | is the variable that is plotted on the vertical axis. |
| x | is the variable that is plotted on the horizontal axis. |
| z | is the variable that is plotted as contour lines, on the Z axis, which is perpendicular to the plane formed by the X and Y axes. |

Options by Category

option(s) can be one or more options in the following categories:

- appearance options:
 - ANNOTATE=*Annotate-data-set*
 - CAXIS=*axis-color*
 - CFRAME=*background-color*
 - COUTLINE=*outline-color*
 - CTEXT=*text-color*
 - GRID
 - NOAXIS | NOAXES
 - NOFRAME
- horizontal axis options:
 - AUTOHREF
 - CAUTOHREF=*reference-line-color*
 - LAUTOHREF=*reference-line-type*
 - CHREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*
 - HAXIS=AXIS<1...99>
 - HMINOR=*number-of-minor-ticks*
 - HREF=*value-list*
 - HREVERSE
 - LHREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*
 - XTICKNUM=*number-of-ticks*
- vertical axis options:
 - AUTOVREF
 - CAUTOVREF=*reference-line-color*
 - CVREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*
 - LAUTOVREF=*reference-line-type*
 - LVREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*
 - VAXIS=AXIS<1...99>
 - VMINOR=*number-of-minor-ticks*
 - VREF=*value-list*
 - VREVERSE
 - YTICKNUM=*number-of-ticks*
- contour options:
 - CLEVELS=*color(s)*
 - JOIN
 - LEGEND=LEGEND<1...99>
 - LEVELS=*value-list*
 - LLEVELS=*line-type-list*
 - NLEVELS=*number-of-levels*
 - NOLEGEND
 - PATTERN
- labeling options:
 - AUTOLABEL | AUTOLABEL=(*autolabel-suboptions*)

where *autolabel-suboptions* can be one or more of these:

CHECK=*checking-factor* | NONE

MAXHIDE=*amount*<*units*>

REVEAL

TOLANGLE=*angle*

- catalog entry description options:

DESCRIPTION=*'entry-description'*

NAME=*'entry-name'*

Options

You can specify any of the following options in your PLOT statement, in any order. If you use a BY statement on the procedure, the options in each PLOT statement affect all graphs produced by that BY statement.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies an Annotate data set to enhance the charts produced by the PLOT statement.

See also: Chapter 26, “The GANNO Procedure,” on page 707 and Chapter 25, “Annotate Dictionary,” on page 613.

AUTOHREF

displays reference lines at major tick marks on the horizontal axis. The positions of the major tick marks are determined by the HAXIS= or XTICKNUM= option. To specify colors and line types for these reference lines, use the CAUTOHREF= and LAUTOHREF= options. To specify labels for these reference lines, use the HAXIS= option.

Not supported by: Java

AUTOLABEL | AUTOLABEL=(*autolabel_suboptions*)

automatically labels the contour lines. Autolabel-suboptions “Autolabel Suboptions” on page 898 allow you to control the appearance of these labels.

The label for each contour line is the value of the *z* variable for that contour level. By default, labels are displayed in BEST format. To change the format, use a FORMAT statement.

When AUTOLABEL is used, the LLEVELS= and CLEVELS= options are ignored and the SYMBOL “SYMBOL Statement” on page 183 statement controls label text and contour-line attributes.

Even though AUTOLABEL labels the contour lines, a default legend is still generated. To suppress the legend, use the NOLEGEND option.

Featured in: Example 2 on page 906

Not supported by: Java, ActiveX

AUTOVREF

displays reference lines at major tick marks on the vertical axis. The number of major tick marks is determined by the VAXIS= or YTICKNUM= option. To specify colors and line types for these reference lines, use the CAUTOVREF= and LAUTOVREF= options. To specify labels for these reference lines, use the VAXIS= option.

Not supported by: Java.

CAUTOHREF=*reference-line-color*

specifies a color for reference lines drawn at major tick marks on the horizontal axis, as specified by the AUTOHREF option. The default color for reference lines is

determined by the CAXIS= option or by the first color in the color list. To specify line types for these reference lines, use the LAUTOHREF= option. To specify labels for these reference lines, use the HAXIS= option.

CAUTOVREF=reference-line-color

specifies a color for reference lines drawn at major tick marks on the vertical axis, as specified by the AUTOVREF option. The default color for reference lines is determined by the CAXIS= option or by the first color in the color list. To specify line types for these reference lines, use the LAUTOVREF= option. To specify labels for these reference lines, use the VAXIS= option.

Not supported by: Java

CAXIS=axis-color

specifies a color for axis lines, tick marks, and reference lines. By default, axes are displayed in the second color in the colors list.

If you use the CAXIS= option, it may be overridden by the COLOR= suboption of the MAJOR= or MINOR= option in an AXIS definition.

CFRAME=background-color

CFR=background-color

fills the axis area with the specified color and draws a frame around the axis area.

CHREF=reference-line-color | (reference-line-color) | reference-line-color-list

CH=reference-line-color | (reference-line-color) | reference-line-color-list

specifies the color of reference lines drawn perpendicular to the horizontal axis. Specifying a single color without parentheses applies that color to all reference lines drawn with the AUTOHREF and HREF= options. Note that the CAUTOHREF= option overrides the CHREF= option for lines drawn with the AUTOHREF option. Specifying a single color in parentheses applies that color only to the first reference line drawn with the HREF= option. Specifying a color list applies colors sequentially to successive reference lines drawn with the HREF= option. The syntax of the color list is of the form (*color1 color2... colorN*) or (*color1, color2..., colorN*). Default colors for reference lines are determined by the CAXIS= option or by the first color in the color list. To specify line types for these reference lines, use the LHREF= option. To specify labels for these reference lines, use the HAXIS= option.

Not supported by: Java

CLEVELS=color(s)

specifies a list of colors for plot contour levels. The number of specified colors should correspond to the number of contour levels since one color represents each level of contour. If fewer colors are specified than the number of levels in the plot, the procedure provides default colors from the current colors list. The procedure default is to rotate through the current colors list for each line type.

This option is ignored if AUTOLABEL is used.

COUTLINE=outline-color

specifies a color for outlining filled areas. This option is ignored unless the PATTERN option is also used. By default, the outline color is the same as the color of the filled area.

The default outline color depends on the PATTERN statement. If you do not specify a PATTERN statement, the default outline color is black for the Java or ActiveX devices. Otherwise, the default outline color is the foreground color (the first color in the colors list). If you specify the PATTERN statement or the V6COMP graphics option, the default is COUTLINE=SAME.

Note: The outline color is the only distinction between empty patterns. Use of this option makes the patterns look the same when VALUE=EMPTY in PATTERN definitions. \triangle

Featured in: Example 4 on page 910

Not supported by: ActiveX (partial)

CTEXT=*text-color*

specifies a color for all text on the axes and legend, including axis labels, tick mark values, legend labels, and legend value descriptions.

For the Java and ActiveX device drivers, the default text color is black. Otherwise, the default text color is determined in the following order:

- 1 specified colors on assigned AXIS and LEGEND statements
- 2 the CTEXT= option in a GOPTIONS statement
- 3 the first color in the colors list.

For legend text, colors that you specify on an assigned LEGEND statement override CTEXT=. Thus, a LEGEND statement's VALUE= color is used for legend values, and its LABEL= color is used for legend labels.

For axes text, colors that you explicitly specify for values and labels on an assigned AXIS statement override CTEXT=. Thus, an AXIS statement's VALUE= color is used for axis values, and its LABEL= color is used for axis labels. However, if the AXIS statement specifies only general axis colors with its COLOR= option, the CTEXT= color overrides that general specification and is used for axis labels and values; the COLOR= color is still used for all other axis colors, such as tick marks.

If you use a BY statement in the procedure, the color of the BY variable labels is controlled by the CBY= option in the GOPTIONS statement.

Featured in: Example 4 on page 910

CVREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*

CV=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*

specifies the color of reference lines drawn perpendicular to the vertical axis.

Specifying a single color without parentheses applies that color to all reference lines drawn with the AUTOVREF and VREF= options. Note that the CAUTOVREF= option overrides CVREF= option for lines drawn with the AUTOVREF option.

Specifying a single color in parentheses applies that color only to the first reference line. Specifying a color list applies colors sequentially to successive reference lines.

The syntax of the color list is of the form (*color1 color2... colorN*) or (*color1, color2..., colorN*). Default colors for reference lines are determined by the CAXIS= option or by the first color in the color list. To specify line types for these reference lines, use the LVREF= option. To specify labels for these reference lines, use the VAXIS= option.

Not supported by: Java

DESCRIPTION=*'entry-description'*

DES=*'entry-description'*

specifies the description of the catalog entry for the chart. The maximum length for *entry-description* is 256 characters. The description does not appear on the chart. By default, the GCONTOUR procedure assigns a description of the form PLOT OF $y^*x=z$, where $y^*x=z$ is the request specified in the PLOT statement.

GRID

draws reference lines at all major tick marks on both axes and outlines the axis area. This is the same as specifying both AUTOHREF and AUTOVREF.

Not supported by: ActiveX

HAXIS=AXIS<1...99>

assigns axis characteristics from the corresponding axis definition to the horizontal (X) axis. If the AXIS statement specifies the REFLABEL= option, labels will be applied in sequence to all reference lines generated with the AUTOHREF and HREF= options.

For Web output that is generated with a Java or ActiveX device driver, certain options of the AXIS statement are not supported. For details, see “AXIS Statement” on page 124.

See also: “AXIS Statement” on page 124

Featured in: Example 2 on page 906

Not supported by: Java (partial), ActiveX (partial)

HMINOR=*number-of-minor-ticks*

HM=*number-of-minor-ticks*

specifies the number of minor tick marks to draw between each major tick mark on the horizontal (X) axis. The HMINOR= option overrides the MINOR= option in an AXIS definition assigned to the horizontal axis.

HREF=*value-list*

draws one or more reference lines perpendicular to the horizontal axis at points specified by *value-list*. See the LEVELS= option for a description of *value-list*. To specify colors for these reference lines, use the CHREF= option. To specify line types for these reference lines, use the LHREF= option. To specify labels for these reference lines, use the HAXIS= option.

Not supported by: Java

HREVERSE

specifies that the order of the values on the horizontal (X) axis be reversed.

Not supported by: Java

JOIN

combines adjacent grid cells with the same pattern to form a single pattern area. This option is ignored unless the PATTERN option is also used.

Featured in: Example 4 on page 910

LAUTOHREF=*reference-line-type*

specifies a line type for reference lines drawn at major tick marks on the horizontal axis, as specified by the AUTOHREF option. The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. The default value 1 draws a solid line. To specify colors for these reference lines, use the CAUTOHREF= option. To specify labels for these reference lines, use the HAXIS= option.

Not supported by: Java

LAUTOVREF=*reference-line-type*

specifies a line type for reference lines drawn at major tick marks on the vertical axis, as specified by the AUTOVREF option. The *reference-line-type* value can be a whole number in the range of 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. The default value 1 draws a solid line. To specify colors for these reference lines, use the CAUTOVREF= option. To specify labels for these reference lines, use the VAXIS= option.

Not supported by: Java

LEGEND=LEGEND<1...99>

assigns a legend definition that specifies the location, text, and appearance of axes on the plots. To suppress the legend, use the NOLEGEND option. The LEGEND= option is ignored if the specified LEGEND definition is not currently in effect.

If you use the SHAPE= option in a LEGEND statement, the value LINE is valid. If you use the PATTERN option, SHAPE=BAR is also valid.

In Web output that uses a Java or ActiveX device driver, the legend will always appear on the right side of the plot.

See also: “LEGEND Statement” on page 151

Featured in: Example 2 on page 906

Not supported by: Java (partial), ActiveX (partial)

LEVELS=*value-list*

specifies values for the z variable for plot contour levels and therefore changes the number of contour levels. You can specify up to 100 values. This option is ignored if you use the AUTOLABEL option.

By default, the GCONTOUR procedure plots seven contour levels for the z variable. These levels occur at every 15th percent of the range between the 5th and 95th percentiles.

For numeric variables, *value-list* can be an explicit list of values, a starting and an ending value with an interval increment, or a combination of both forms:

- n <... n >
- n TO n <BY *increment*>
- n <... n > TO n <BY *increment*> n <... n >

If a numeric variable has an associated format, the specified values must be the *unformatted* values.

By default, the GCONTOUR procedure selects colors and line types for the contour levels by rotating through the colors list for each line type (1 through 46) until all the levels have been represented. The level lines on the plot represent the intersection of a plane, parallel to the X/Y plane, and the surface that is formed by the values of the z variable. See “Selecting Contour Levels” on page 899 for more information.

You can specify the colors and line types for contour levels. The way to do this depends on whether AUTOLABEL is used:

- If AUTOLABEL is used, the “SYMBOL Statement” on page 183 controls colors and line types for contour levels.
- If AUTOLABEL is not used, the CLEVELS= and LLEVELS= options control colors and line types for contour levels.

As an alternative to representing contour levels with contour lines, you can use the PATTERN option to fill each level with a solid pattern or with the colors and patterns specified in PATTERN statements.

Featured in: Example 2 on page 906 and Example 3 on page 908

LHREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*

LH=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*

specifies line types for reference lines drawn perpendicular to the horizontal axis.

The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. Specifying a single line type without parentheses applies that line type to all reference lines drawn with the AUTOHREF and HREF= options. Note that the CAUTOHREF= option overrides LHREF=*reference-line-type* for lines drawn with the AUTOHREF option. Specifying a single line type in parentheses applies that line type only to the first reference line drawn by the HREF= option. Specifying a line-type list applies line types in sequence to successive reference lines drawn with the HREF= option. The syntax of the line type list is of the form (*type1 type2... type3*). The default value of 1 draws a solid line. To specify colors for these reference lines, use the CHREF= option. To specify labels for these reference lines, use the HAXIS= option.

Not supported by: Java (partial), ActiveX (partial)

LLEVELS=*line-type-list*

lists line type numbers for plot contour lines. Each line type represents one contour level, so the number of line types listed should correspond to the number of contour

levels. Thus, for a contour plot that uses the default seven levels, specify seven line types.

If fewer line types are specified than the number of levels in the plot, the procedure provides default line types. With the default, contour levels rotate through line types 1 through 46, displaying each line type in all of the colors in the colors list before moving to the next line type. Line type 1 is a solid line and the others are dashed line. See “Specifying Line Types” on page 207 for available line types.

For colors and lines specified with both the CLEVELS= and LLEVELS= options, the first contour level is displayed in the first color in the CLEVELS= color list and in the first line type specified with the LLEVELS= option. The second level is displayed in the second color and the second line type, and so on.

This option is ignored if AUTOLABEL is used.

Featured in: Example 3 on page 908

Not supported by: Java

LVREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*

LV=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*

specifies line types for reference lines drawn perpendicular to the vertical axis. The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. Specifying a single line type without parentheses applies that line type to all reference lines drawn with the AUTOVREF and VREF= options. Note that the CAUTOVREF= option overrides LVREF=*reference-line-type* for lines drawn with the AUTOVREF option. Specifying a single line type in parentheses applies that line type only to the first reference line drawn by the VREF= option. Specifying a line-type list applies line types in sequence to successive reference lines drawn with the VREF= option. The syntax of the line type list is of the form (*type1 type2... type3*). The default value 1 draws a solid line. To specify colors for these reference lines, use the CVREF= option. To specify labels for these reference lines, use the VAXIS= option.

Not supported by: Java (partial), ActiveX (partial)

NAME='*entry-name*'

specifies the name of the catalog entry for the graph. The maximum length for *entry-name* is 8 characters. The default name is GCONTOUR. If you specify DEVICE=ACTIXIMG or DEVICE=JAVAIMG, then the name that you specify will be used for the client image output even in the file exists. For all other devices, if the name duplicates an existing entry name, SAS/GRAPH adds a number to the duplicate name to create a unique entry, for example, GCONTOU1.

NLEVELS=*number-of-levels*

specifies the number of contour levels to plot. Values can be integers from 1 to 100. The default is NLEVELS=7. The contour levels are computed as follows, where **L** represents an array of levels:

- If the value of NLEVELS= is less than 7, then

$$\begin{aligned} D &= (Z_{\max} - Z_{\min}) / NLEVELS \\ d &= 0.5 * D \\ L[0] &= Z_{\min} + d, L[i] = L[i-1] + D \end{aligned}$$

In this case, each level is the midpoint of a number of ranges equal to the value of the NLEVELS= option. These ranges exactly cover the range of the *z* variable.

- If the value of NLEVELS= is greater than or equal to 7, then

$$\begin{aligned} e &= 0.05 * (100 - NLEVELS) / 93 \\ d &= (Z_{\max} - Z_{\min}) * e \\ D &= ((Z_{\max} - Z_{\min} - 2*d) / (NLEVELS - 1)) \end{aligned}$$

$$L[0] = Z_{\min} + d, L[i] = L[i-1] + D$$

In this case, the first and last midpoints are set closer to the minimum and maximum values of the z variable as the values of NLEVELS= gets closer to 100, and the remaining midpoints are equally spaced between them.

NOAXIS**NOAXES**

specifies that a plot have no axes, axis values, or axis labels. The frame is displayed around the plot unless you use the NOFRAME option.

NOFRAME

suppresses the frame that is drawn by default around the plot area.

NOLEGEND

suppresses the plot legend that describes contour levels and their line types or fill patterns and colors.

PATTERN

specifies the fill pattern and pattern colors for contour areas. The plot contour levels are represented by rectangles filled with patterns. The pattern for each rectangle is determined by calculating the mean of the values of the z variable for the four corners of the rectangle and assigning the pattern for the level closest to the mean.

By default, the procedure uses a solid pattern for the levels and rotates the pattern through the colors list. If the V6COMP option is in effect for the GOPTIONS statement, cross-hatch patterns are used instead of solid patterns. To explicitly define patterns, use PATTERN definitions for map/plot patterns.

For information on PATTERN statement that are enabled for Web output, see “PATTERN Statement” on page 169.

See also: “Selecting Contour Levels” on page 899

Featured in: Example 4 on page 910

Featured in: Example 3 on page 908

Not supported by: Java (partial), ActiveX (partial)

VAXIS=AXIS<1...99>

assigns axis characteristics from the corresponding axis definition to the vertical (Y) axis. If the AXIS statement specifies the REFLABEL= option, labels will be applied in sequence to all reference lines generated with the AUTOVREF and VREF= options.

For Web output that is generated with a Java or ActiveX device driver, certain options of the AXIS statement are not supported. For details, see “AXIS Statement” on page 124.

See also: “AXIS Statement” on page 124

Featured in: Example 2 on page 906

Not supported by: Java (partial), ActiveX (partial)

VMINOR=number-of-minor-ticks**VM=number-of-minor-ticks**

specifies the number of minor tick marks located between each major tick mark on the vertical (Y) axis. No values are displayed for minor tick marks. The VMINOR= option overrides the MINOR= option in an AXIS definition that is assigned to the vertical axis.

VREF=value-list

draws one or more reference lines perpendicular to the vertical axis at points specified by *value-list*. See the LEVELS= option for a description of *value-list*. To specify colors for these reference lines, use the CVREF= option. To specify line types for these reference lines, use the LVREF= option. To specify labels for these reference lines, use the VAXIS= option.

Not supported by: Java

VREVERSE

specifies that the order of the values on the vertical axis be reversed.

Not supported by: Java

XTICKNUM=*number-of-ticks*

YTICKNUM=*number-of-ticks*

specify the number of major tick marks located on a plot's X or Y axis, respectively. The value of *number-of-ticks* must be 2 or greater. The defaults are XTICKNUM=5 and YTICKNUM=5.

The MAJOR= or ORDER= option in an AXIS definition that is assigned to the X axis overrides the XTICKNUM= option. The MAJOR= or ORDER= option in an AXIS definition that is assigned to the Y axis overrides the YTICKNUM= option.

Autolabel Suboptions

The AUTOLABEL= option accepts the following autolabel suboptions.

CHECK=*checking-factor* | NONE

specifies a collision checking factor that controls collisions between contour label text and other contour lines or other labels. Values can be integers from 0 to 100, inclusive, where 0 provides minimal collision checking and 100 provides maximal collision checking. Fractional values are permitted. The default is CHECK=75.

CHECK=NONE suppresses contour label collision checking and may lessen the time needed to compute the contour graph.

MAXHIDE=*amount* <*units*>

specifies the maximum amount of contour line that can be hidden by contour labels. The value of *amount* must be greater than zero.

Valid *units* are CELLS (horizontal character cell positions), CM (centimeters), IN (inches), or PCT (percentage of the width of the graphics output area). The default is MAXHIDE=100PCT. If you omit *units*, a unit specification is searched for in this order:

- 1 the GUNIT= option in a GOPTIONS statement
- 2 the default unit, CELLS.

If you specify units of PCT or CELLS, the MAXHIDE= suboption calculates the amount of contour line that can be hidden based on the width of the graphics output area. For example, if you specify MAXHIDE=50PCT and if the graphics output area is 9 inches wide, the maximum amount of the contour line that can be hidden by labels is 4.5 inches.

This option maintains data integrity. It provides a check for overly small increments in the STEP= option in the SYMBOL statement. Additionally, it can prevent small contours from being significantly hidden even when the value of STEP= is sufficiently large.

REVEAL

specifies that the contour lines are visible through the label text as dashed lines. Line style 33 is used. This option provides a simple way to see all portions of labeled contours and can be used to inspect the label positions with respect to the contour lines. It is primarily used for debugging. Occasionally, single-character contour labels can be placed off center from the clipped portion of the contour line when the contour line is irregular or jagged.

TOLANGLE=*angle*

specifies the maximum angle (the tolerance angle) between any two adjacent characters of a contour label. The value of *angle* must be between 1 and 85 degrees. The default is TOLANGLE=30. To force contour labels to fall on very smooth sections, specify a small tolerance angle.

Selecting Contour Levels

You can use the LEVELS= option to select the contour levels for your plot. You use LEVELS= values differently, depending on whether you specify the PATTERN option in the PLOT statement.

When you do not use the PATTERN option, the levels represent the intersection of a plane (parallel to the X/Y plane at the value of the *z* variable) and the surface formed by the data. That is, if you use the data to create a surface plot with the G3D procedure, the contour lines in a GCONTOUR procedure plot represent the intersection of the plane and the surface.

For example, suppose that you use the G3D procedure, and your data produces a surface plot like the one shown in Figure 30.2 on page 899. The same data used with the following PLOT statement in the GCONTOUR procedure produces a similar contour plot:

```
plot y*x=z / levels=-7.5 to 7.5 by 2.5;
```

The contour lines in Figure 30.3 on page 900 represent the intersection of the surface in Figure 30.2 on page 899 with planes parallel to the plane formed by the variables *x* and *y* and located at *z* values of -7.5 , -5.0 , -2.5 , and so on.

Figure 30.2 Surface Plot

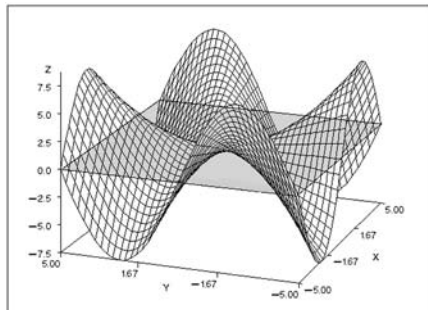
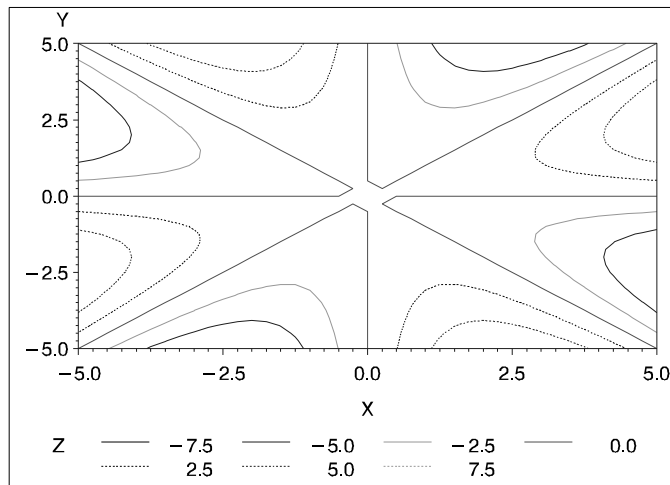


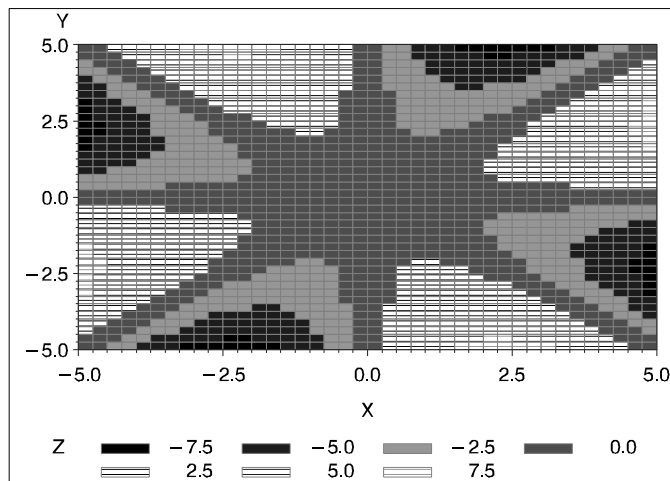
Figure 30.3 Line Contour Levels



When you use the PATTERN option, contour levels are represented by rectangles filled with patterns. The rectangles are formed by points in the x - y grid. The contour pattern of a rectangle, or grid cell, is determined by the mean or average value of the z variable for the four corners of the rectangle. The grid cell is assigned the pattern for the level closest to the calculated mean. For example, if you have specified contour levels of 0, 5, and 10, and the plot contains a grid cell with a mean of 100, it is assigned the pattern for the nearest level: 10. A grid cell with a mean of 7.6 will also be assigned the pattern for the 10 level.

Figure 30.4 on page 900 shows a contour plot with the PATTERN option that uses the same data and contour levels as Figure 30.3 on page 900. The pattern for the rectangle is assigned depending on the mean of the grid values at the four corners. As a result, two contour plots using the same contour levels can present your data differently if one plot uses a pattern and the other does not. The contour pattern boundaries do not correspond to the contour lines shown in Figure 30.3 on page 900.

Figure 30.4 Pattern Contour Levels



Specifying Axis Order

You can use AXIS statements to modify the text and appearance of plot axes, and then you can assign the axes to the contour plot with the PLOT statement's HAXIS= and VAXIS= options. If the AXIS statement uses an ORDER= option, there are special considerations for using that AXIS definition with the GCONTOUR procedure.

A list of variable values that are specified with the AXIS statement's ORDER= option must contain numbers listed in ascending or descending order; these numbers are treated as a continuous data range for an axis. Thus, for a contour line or pattern to span the entire specified range, it is not necessary for the maximum and minimum values of the list to match exactly with the maximum and minimum data values of the corresponding x or y variable. For example, suppose that you assign this AXIS definition to the horizontal (x) axis:

```
axis1 order=-2.5 to 2.5 by .5
```

Suppose also that the horizontal axis variable has these values: $-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5$. Depending on the data, contours could extend through the full range of the ORDER= list rather than from -2 to 2 , which are the actual values of the variable assigned to the horizontal (x) axis. In this case, values are interpolated for the x variable at any point where the y variable intersects the minimum axis value (-2.5) or the maximum axis value (2.5). Data values that are outside of the axis range (in this case, $-5, -4, -3, 3, 4,$ and 5) are clipped from the plot.

When ORDER= lists cause data clipping, internal plotting grids are modified according to these rules:

- If an ORDER= list causes data clipping on a single axis, linear interpolation generates the z values of the starting and/or ending column of the plotting grid. For example, in the previous example, the value of z is interpolated for -2.5 and 2.5 on the horizontal (x) axis.
- If ORDER= lists cause data clipping on both axes, the response variable values of the new corners are derived by fitting the new x, y location on a plane formed by three of the original four points of the corresponding grid square.

In addition, if you assign the following AXIS definition to a plot of the same data, the contour levels on the plot will not extend beyond the range of the data:

```
axis1 order=-10 to 10 by 1;
```

To see the effects of the ORDER= option:

- Figure 30.5 on page 902 shows the effects when the range of ORDER= values matches the range of values for the variables assigned to the horizontal (x) and vertical (y) axes.
- Figure 30.6 on page 902 shows the effects when the range of ORDER= values is smaller than the range of data values.
- Figure 30.7 on page 903 shows the effects when the range of ORDER= values is larger than the range of data values.

Figure 30.5 AXIS Statement's ORDER= Option, where Option Values Match Variable Values

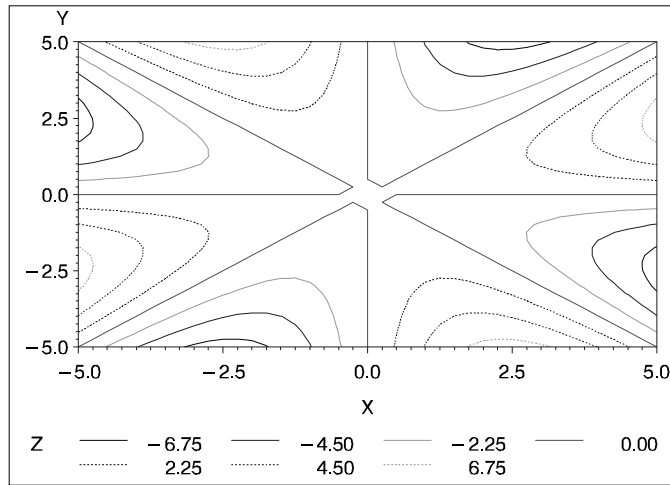


Figure 30.6 ORDER= Option, where the Option Range is Smaller than the Variable Range

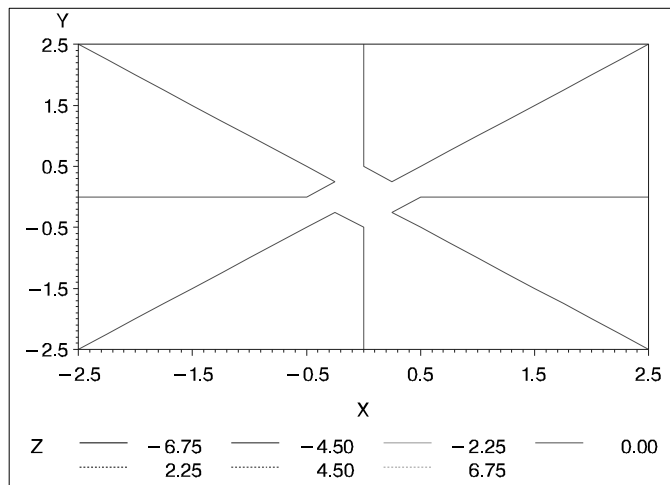
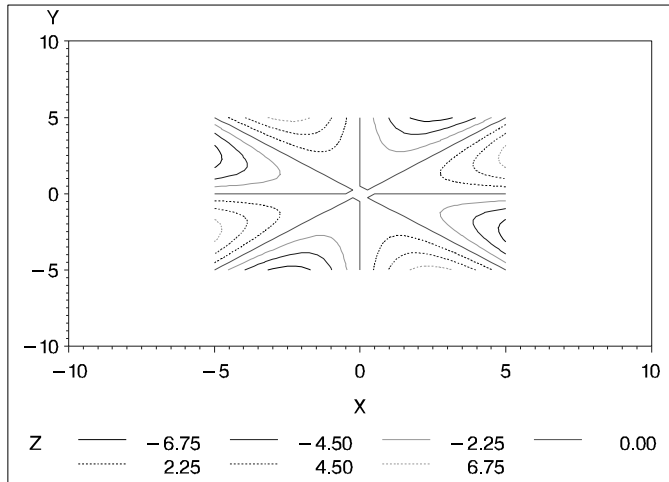


Figure 30.7 ORDER= Option, where the Option Range is Larger than Variable Range



Modifying Contour Lines and Labels with the SYMBOL Statement

When you use the AUTOLABEL option, the LLEVELS= and CLEVELS= options are ignored, and contour-line and label attributes are controlled by the SYMBOL statement. Defaults are used if not enough SYMBOL statements are specified to match the number of contour levels.

If a SYMBOL statement does not include a color option, that statement may be applied to more than one contour level. In this case, the SYMBOL statement is used once with every color in the colors list and generates more than one SYMBOL definition. See “SYMBOL Statement” on page 183 for details.

Table 30.1 on page 903 describes how SYMBOL statement options affect contour plot lines and labels.

Table 30.1 The Effect of SYMBOL Statement Options on Contour Lines and Labels

SYMBOL Statement Option	Contour Line or Label Element Affected
LINE= <i>line-type</i>	Contour line style
WIDTH= <i>n</i>	Contour line thickness
CI= <i>line-color</i> or COLOR= <i>color</i>	Contour line color
FONT= <i>font</i>	Contour label font
HEIGHT= <i>height</i>	Contour label height
CV= <i>color</i> or COLOR= <i>color</i>	Contour label color
STEP= <i>distance<units></i>	Minimum distance between labels on the same contour line
VALUE= <i>'text'</i>	Contour label text
VALUE=NONE	Suppresses the contour label text

The SYMBOL statement option INTERPOL= is not supported by the GCONTOUR procedure.

The `STEP=` option specifies the minimum distance between contour labels. The lower the value, the more labels the procedure uses. A `STEP=` value of less than 10 percent is ignored by the `GCONTOUR` procedure and a value of 10 percent is substituted.

For more information, see “`SYMBOL Statement`” on page 183.

Specifying Text for Contour Labels

To override the default labels that are displayed by the `AUTOLABEL` option, you can specify label text for one or more contour lines. To do so, use both the `FONT=` and `VALUE=` options on the `SYMBOL` statement that is assigned to the contour level. Default labels are used for contour levels that you do not label.

For example, this `SYMBOL1` statement displays the text string **Highest** in the Swiss font on the contour line that it modifies:

```
symbol1 font=swiss value='Highest';
```

You must specify both `FONT=` and `VALUE=` or the text is not used. For an example, see Example 2 on page 906.

Examples

Example 1: Generating a Simple Contour Plot

Procedure features:

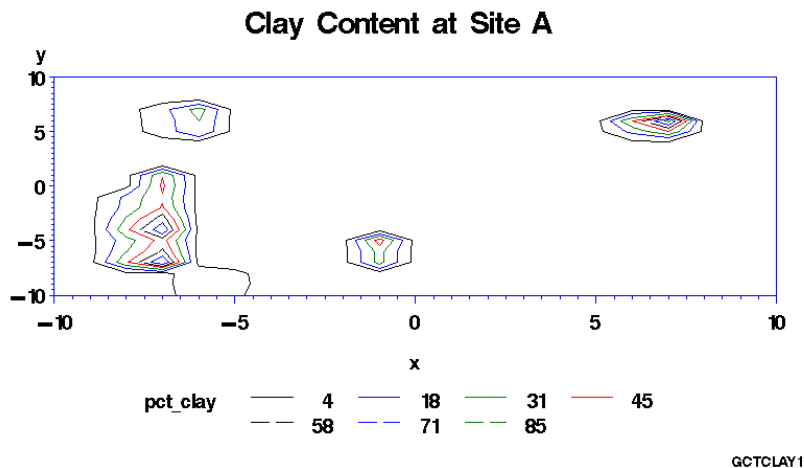
`PLOT` statement

Other features:

`FORMAT` statement

`G3GRID` procedure

Sample library member: `GCTCLAY1`



This example shows a simple contour plot that describes the percentage of clay found in soil samples at various locations of a testing site. By default, the axes are scaled to

include all data values and are labeled with the names of the axes variables. Values are plotted with seven contour levels, which are represented by contour lines with default colors and line types. The default contour levels occur at every 15th percent of the range between the contour variable's 5th and 95th percentile. The legend is labeled with the contour variable's name and identifies the contour levels that are included in the plot.

This example uses the G3GRID procedure to interpolate clay percentages for grid cells that do not have percentages in the data. Without the G3GRID procedure, there are too many missing values for the percentages, and the GCONTOUR procedure cannot produce a satisfactory contour plot.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(black blue green red)
          ftext=swiss ftitle=swissb htitle=6 htext=4;
```

Create the data set. REFLIB.CLAY contains the percent of clay at various locations of a test site.

```
data reflib.clay;
  input x y pct_clay;
  datalines;
-10   -10      2.316
-10   -9       1.816
-10   -8       2.427
...more data lines...
  10    8       .
  10    9       .
  10   10       .
;
```

Interpolate values for the contour plot. The interpolated data set is stored in REFLIB.CLAY2.

```
proc g3grid data=reflib.clay out=reflib.clay2;
  grid y*x=pct_clay / naxis1=21
                    naxis2=21
                    join;
run;
```

Define title and footnote.

```
title1 'Clay Content at Site A';
footnote1 j=r ' GCTCLAY1 ';
```

Generate a simple contour plot. The procedure uses REFLIB.CLAY2, the output data set from PROC G3GRID. To simplify the legend labels, clay percentages are formatted with no decimal positions.

```

proc gcontour data=reflib.clay2;
  format pct_clay 2.0;
  plot y*x=pct_clay;
run;
quit;

```

Example 2: Labeling Contour Lines

Procedure features:

PLOT statement options:

```

AUTOLABEL=
HAXIS=
LEGEND=
LEVELS=
VAXIS=

```

Other features:

```

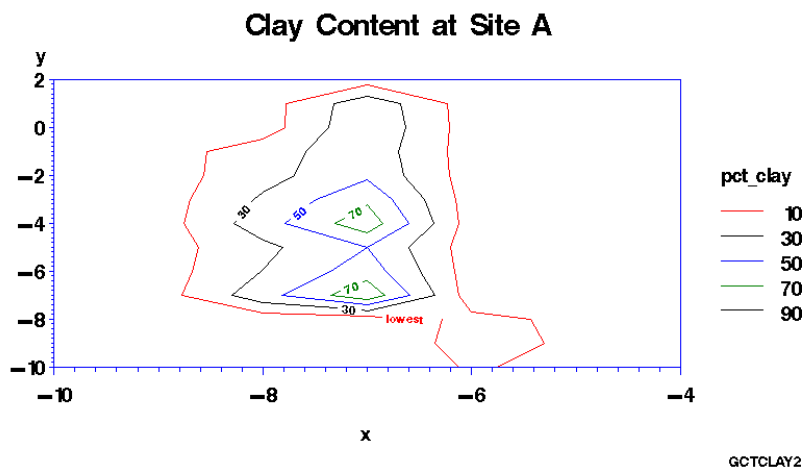
AXIS statement
LEGEND statement
SYMBOL statement

```

Data set: REFLIB.CLAY2

Sample library member: GCTCLAY2

Display 30.2 A Simple Contour Plot with Labelled Contour Levels



This example modifies Example 1 on page 904 to label contour levels with the `AUTOLABEL` option. When `AUTOLABEL` is used, the `SYMBOL` statement controls the labels and attributes of contour lines. In this example, `SYMBOL1` defines a text label for the lowest contour level. Each remaining contour line gets the default label, which is the contour variable's value at that contour level. All the contour lines are solid, which is the default line type for the `SYMBOL` statement.

This example also uses AXIS statements to limit the plot to one of the contour areas from the output of Example 1 on page 904, and it uses a LEGEND statement to move the legend so the procedure has more room for displaying the y axis.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(black blue green red)
          ftext=swiss ftitle=swissb htitle=6 htext=4;
```

Define title and footnote.

```
title1 'Clay Content at Site A';
footnote1 j=r 'GCTCLAY2';
```

Define axes characteristics. AXIS1 uses ORDER= to set major tick marks at 2-unit intervals from -10 to -4. AXIS2 uses ORDER= to specify 2-unit intervals from -10 to 2. These axes ranges effectively zoom in on one of the contour areas from Example 1 on page 904.

```
axis1 order=(-10 to -4 by 2);
axis2 order=(-10 to 2 by 2);
```

Define legend characteristics. POSITION= centers the legend to the right of the graphics area, and LABEL= positions the legend label above the legend entries. ACROSS= places legend entries in rows 1 entry wide.

```
legend1 position=(right middle)
        label=(position=top)
        across=1;
```

Define symbol characteristics. SYMBOL1 specifies a font and text string to label the lowest-level contour lines. COLOR= ensures that each SYMBOL definition is used only once. In SYMBOL2, STEP= increases the number of contour labels by placing the labels closer together than the default distance of 65 percent.

```
symbol1 height=2.5
        font=swissb
        value='lowest'
        color=red;
symbol2 height=2.5
        step=25pct
        color=black;
symbol3 height=2.5
        color=blue;
symbol4 height=2.5
        color=green;
```

Generate the contour plot. LEVELS= specifies contour levels from 10 to 90 at 20-unit intervals. AUTOLABEL= turns on labeling, and CHECK=NONE turns off collision checking so the maximum number of contour labels can be displayed. HAXIS= and VAXIS= assign AXIS definitions to the plot. LEGEND= assigns the LEGEND1 definition to the plot.

```
proc gcontour data=reflib.clay2;
  plot y*x=pct_clay / levels=10 to 90 by 20
        autolabel=(check=none)
        haxis=axis1
        vaxis=axis2
        legend=legend1;

run;
quit;
```

Example 3: Specifying Contour Levels

Procedure features:

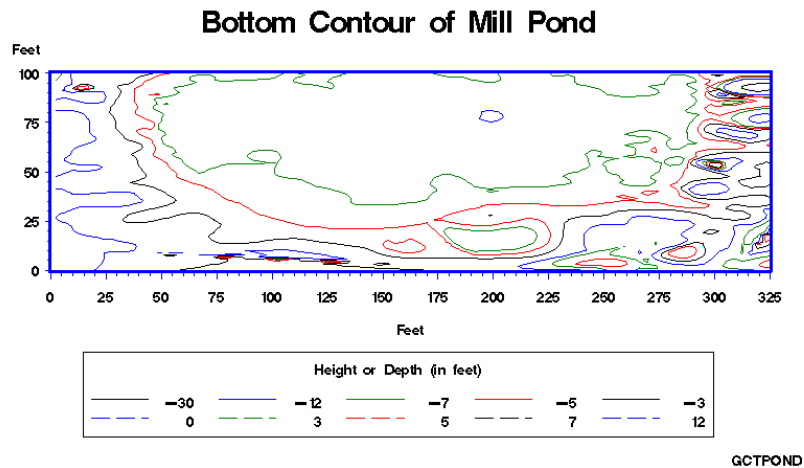
PLOT statement options:

LEVELS=

LLEVELS=

Sample library member: GCTPOND

Display 30.3 A Contour Plot Generated with Specific Contour Levels



This example generates a contour plot that shows the height or depth of a pond and its surrounding land. In the example, the PLOT statement uses the LEVELS= and LLEVELS= options to specify explicit contour levels and line types for the contour plot. It also uses a LEGEND statement to modify the plot's default legend.

This example uses the G3GRID procedure to interpolate points for grid cells that do not have a needed dimension in the data. Without the G3GRID procedure, there are too many missing values for the point locations, and the GCONTOUR procedure cannot produce a satisfactory contour plot.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss ftitle=swissb htitle=6 htext=3;
```

Create the data set. REFLIB.POND contains the raw data for a pond floor and surrounding land.

```
data reflib.pond;
  input vdist hdist height;
  datalines;
10    88      0
18    55     -1
24    22.5   -1.67
...more data lines...
64    272.5  -6.25
60    277.5  -6.5
62    277.5  -6.5
;
```

Define title and footnote.

```
title 'Bottom Contour of Mill Pond';
footnote j=r ' GCTPOND ';
```

Define axis characteristics.

```
axis1 order=(0 to 325 by 25) width=3 minor=(n=4)
      label=('Feet');
axis2 order=(0 to 100 by 25) width=3 minor=(n=4)
      label=(' Feet');
```

Define legend characteristics.

```
legend1 frame shape=line(7)
        label=(position=top j=c 'Height or Depth (in feet)');
```

Interpolate points for the contour plot.

```
proc g3grid data=reflib.pond out=reflib.pondgrid;
  grid vdist*hdist=height / naxis1=100 naxis2=100;
run;
```

Generate the contour plot. LEVELS= specifies the values of the contour levels. LLEVELS= sets the line types for the contour lines. Solid lines identify negative contour levels, and dashed lines identify positive contour levels.

```

proc gcontour data=reflib.pondgrid;
  plot vdist*hdist=height /levels= -30 -12 -7  -3 0 3 5 7 12
                                llevels= 1  1  1  1  1 2 2 2 2 2
                                legend=legend1
                                haxis=axis1
                                vaxis=axis2;

run;
quit;

```

Example 4: Using Patterns and Joins

Procedure features:

PLOT statement options:

COUTLINE=

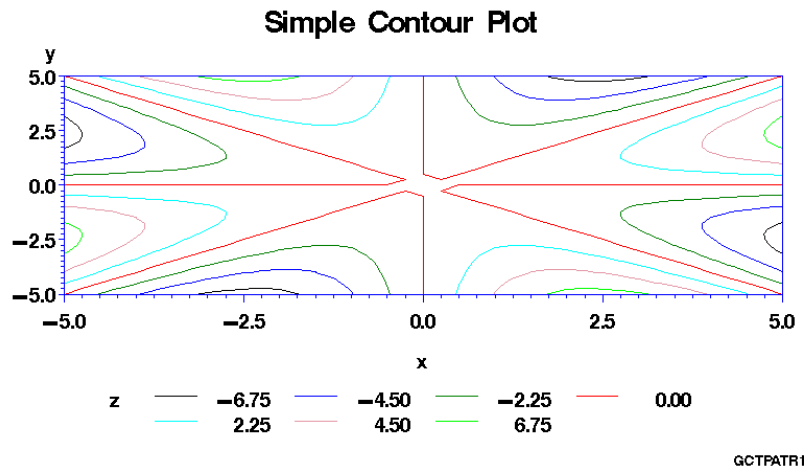
CTEXT=

JOIN

PATTERN

Sample library member: GCTPATRN

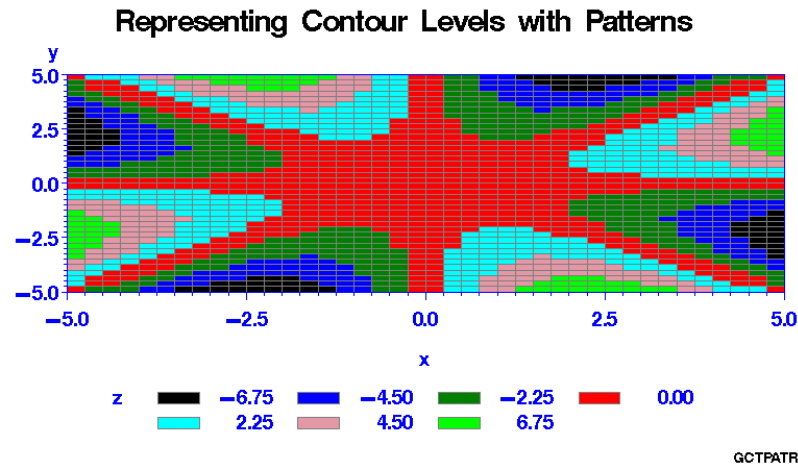
Display 30.4 A Contour Plot that uses Contour Lines



This example demonstrates the differences between using lines and patterns to represent contour levels. It first uses a simple PLOT statement to generate the default output, which uses lines to represent contour levels.

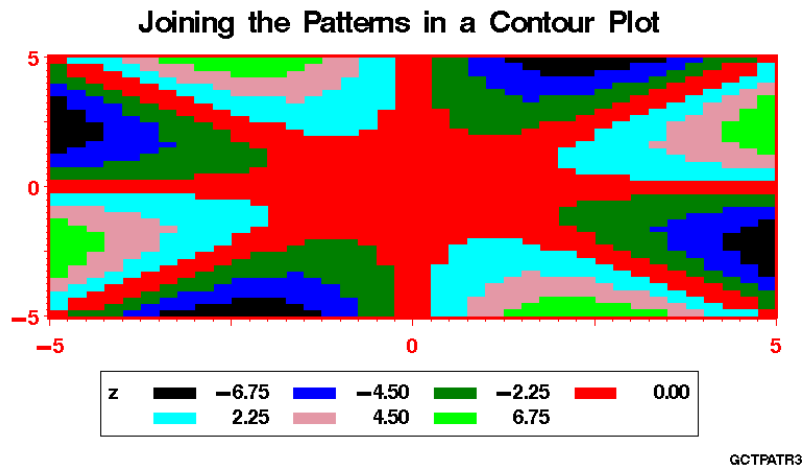
As shown in the following output, the example then modifies the PLOT statement by specifying the PATTERN option, which uses patterns to distinguish between contour levels. Additional PLOT statement options outline filled areas in gray and specify green text for all text on the axes and in the legend.

Display 30.5 A Contour Plot that uses Patterns



Finally, as shown by the following output, the example uses the JOIN option to combine the patterns in grid cells for the same contour level. Additional options enhance the plot by modifying the axes and framing the legend.

Display 30.6 A Contour Plot with Joined Cells



Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss ftitle=swissb htitle=6 htext=4;
```

Create the data set. REFLIB.SWIRL is generated data that produces a symmetric contour pattern, which is useful for illustrating the PATTERN option.

```

data reflib.swirl;
  do x= -5 to 5 by 0.25;
    do y= -5 to 5 by 0.25;
      if x+y=0 then z=0;
      else z=(x*y)*((x*x-y*y)/(x*x+y*y));
      output;
    end;
  end;
run;

```

Define title and footnote for the default output.

```

title 'Simple Contour Plot';
footnote j=r 'GCTPATR1 ';

```

Generate a simple contour plot.

```

proc gcontour data=reflib.swirl;
  plot y*x=z;
run;

```

Define title and footnote for second plot.

```

title 'Representing Contour Levels with Patterns';
footnote j=r 'GCTPATR2 ';

```

Generate the contour plot. PATTERN fills the contour levels with solid patterns. COUTLINE= names the color that outlines the grid cells. CTEXT= names a color for axes and legend text.

```

proc gcontour data=reflib.swirl;
  plot y*x=z / pattern
              coutline=gray
              ctext=green;
run;

```

Define title and footnote for last plot.

```

title 'Joining the Patterns in a Contour Plot';
footnote j=r 'GCTPATR3 ';

```

Define axis and legend characteristics for last plot. Blanks are used to suppress tick labels at positions -2.5 and 2.5.

```

axis1 label=none value=(' ' ' ' '0' ' ' '5')
      color=red width=3;
axis2 label=none value=(' ' ' ' '0' ' ' '5')

```

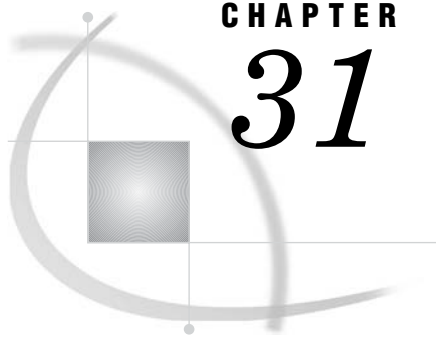
```
color=red width=3;  
  
legend frame;
```

Generate the last contour plot. JOIN combines grid cells for the same contour levels.

```
proc gcontour data=reflib.swirl;  
  plot y*x=z / pattern  
    join  
    haxis=axis1  
    vaxis=axis2  
    legend=legend1;  
  
run;  
quit;
```

References

Snyder, W.V. (1978), "Contour Plotting [J6] ," *ACM Transactions on Mathematical Software*, 4, 290–294.



CHAPTER

31

The GDEVICE Procedure

<i>Overview</i>	916
<i>Concepts</i>	916
<i>About Device Catalogs</i>	916
<i>About the Current Catalog</i>	916
<i>Search Order of Device Catalogs</i>	917
<i>Ways to Use the GDEVICE Procedure</i>	917
<i>Windowing Mode</i>	918
<i>Program Mode</i>	918
<i>Procedure Syntax</i>	920
<i>PROC GDEVICE Statement</i>	920
<i>ADD Statement</i>	921
<i>COPY Statement</i>	924
<i>DELETE Statement</i>	925
<i>FS Statement</i>	925
<i>LIST Statement</i>	925
<i>MODIFY Statement</i>	926
<i>QUIT Statement</i>	927
<i>RENAME Statement</i>	927
<i>Using the GDEVICE Procedure</i>	928
<i>Using the GDEVICE Windows</i>	928
<i>GDEVICE Window Commands</i>	929
<i>GDEVICE Window Descriptions</i>	929
<i>DIRECTORY Window</i>	929
<i>Detail window</i>	930
<i>Parameters window</i>	930
<i>Gcolors window</i>	931
<i>Chartype window</i>	931
<i>Colormap window</i>	931
<i>Metagraphics window</i>	932
<i>Gprolog window</i>	932
<i>Gepilog window</i>	933
<i>Gstart window</i>	933
<i>Gend window</i>	933
<i>Host File Options window</i>	933
<i>Host Commands window</i>	934
<i>Creating or Modifying Device Entries</i>	934
<i>Creating a New Device Entry</i>	935
<i>Modifying an Existing Device Entry</i>	935
<i>Changing Device Parameters Temporarily</i>	935
<i>Examples</i>	936
<i>Example 1: Creating a Custom Device Entry with Program Statements</i>	936

Overview

The GDEVICE procedure is a tool for examining and changing the parameters of the graphics device driver catalog entries used with SAS/GRAPH software. With the GDEVICE procedure, you can use either the GDEVICE windows or GDEVICE procedure statements to

- list the device entries stored in any DEVICES catalog
- view the parameters for any device entry
- create and modify new device entries
- copy, modify, rename, or delete existing device entries.

See Chapter 3, “Device Drivers,” on page 41 for a discussion of device drivers and device entries, as well directions for selecting device drivers and changing the settings of device parameters.

For a complete list of Institute-supplied device entries supported by your operating environment, see the SASHELP.DEVICES catalog that is supplied with SAS/GRAPH software.

Concepts

About Device Catalogs

Device entries are stored in SAS catalogs that are named *libref*.DEVICES. Device entries for your operating environment that are supplied with SAS/GRAPH software are stored in the Institute-supplied catalog, SASHELP.DEVICES.

Custom device entries are typically stored in a catalog named GDEVICE n .DEVICES (where n can be any number from 0 to 9). However, device entries that have been created or modified by a system administrator specifically for your site also may be stored in SASHELP.DEVICES. (On multi-user systems, the SAS Support Consultant is usually the person who has write access to the SASHELP.DEVICES catalog and makes any changes.)

About the Current Catalog

When the GDEVICE procedure determines which catalog it should use, it searches for the catalog in the following order:

- 1 the catalog name specified in the CATALOG= option in the PROC GDEVICE statement
- 2 the catalog associated with the GDEVICE0 libref, if the libref has been assigned
- 3 the Institute-supplied catalog, SASHELP.DEVICES. (SASHELP.DEVICES is usually write-protected and is opened in browse mode.)

The first catalog it finds becomes the current catalog.

You can specify the current catalog by

- using the CATALOG= option in the PROC GDEVICE statement (this is required to open a driver entry in update mode)
- assigning the GDEVICE0 libref to the appropriate catalog.

Search Order of Device Catalogs

When you specify a device driver, SAS/GRAPH software looks only into catalogs with certain librefs and names to find a device entry for that driver. It searches these catalogs sequentially in the following order:

- 1 If the libref GDEVICE0 has been assigned to a SAS library, SAS/GRAPH software looks in that library for a catalog named DEVICES. If the GDEVICE0.DEVICES catalog exists, it is checked for the specified device entry. If the device entry is not there, SAS/GRAPH software looks next for a library with the libref GDEVICE1 and for a catalog named DEVICES in that library. The search is repeated for the sequence of librefs through GDEVICE9.
- 2 If SAS/GRAPH fails to find the specified device entry in any DEVICES catalog in the libraries GDEVICE0 through GDEVICE9, or if before locating the specified device entry it encounters in that sequence an undefined libref or a library that does not contain a DEVICES catalog, it jumps to SASHELP.DEVICES to search for the device entry. For example, if a GDEVICE0 libref is allocated but this library does not contain a DEVICES catalog, SAS/GRAPH software jumps to the SASHELP.DEVICES catalog, without searching for a GDEVICE1.DEVICES catalog, even if it exists. (SASHELP.DEVICES is the device catalog supplied with SAS/GRAPH software. SASHELP is one of the standard librefs defined automatically whenever you start your SAS session; you do not need to issue a LIBNAME statement to define it.)
- 3 If the specified device entry is not found in the SASHELP.DEVICES catalog, you receive an error message.

Since the GDEVICE0.DEVICES catalog is the first place that SAS/GRAPH software looks, you always should assign that libref to the library containing your personal catalog of device entries, if you have one. If for some reason you have personal device catalogs in more than one SAS data library, assign them librefs in the sequence GDEVICE0, GDEVICE1, GDEVICE2, and so on.

Note: As stated above, the search for entries terminates if there is a break in the sequence; the catalog GDEVICE1.DEVICES is not checked if the libref GDEVICE0 is undefined, or if GDEVICE0 does not contain a catalog named DEVICES. △

To cancel or redefine the libref GDEVICE n , first clear the current graphics options:

```
goptions reset=all;
```

You can then redefine the libref with another LIBNAME statement. To cancel a libref, use a null LIBNAME statement.

Ways to Use the GDEVICE Procedure

There are two ways to use the GDEVICE procedure:

- browse or edit the fields in the GDEVICE procedure windows (windowing mode)
- submit GDEVICE procedure statements in a SAS program (program mode).

If you run SAS software in a windowing environment (the SAS Display Manager System, for example), you can use either the GDEVICE procedure windows or the GDEVICE procedure statements. In a windowing environment, the GDEVICE procedure automatically opens the GDEVICE procedure windows.

If you run SAS software in a non-windowing environment (such as line-mode or batch), you can use only GDEVICE procedure statements. In a non-windowing environment, the GDEVICE procedure automatically uses program mode.

Both methods provide identical functionality and allow you to display or modify device parameters, or create new device entries.

Windowing Mode

In a windowing environment, open the GDEVICE windows by submitting the PROC GDEVICE statement without the NOFS option:

```
proc gdevice;
```

This opens the DIRECTORY window in browse mode. This window lists all of the device entries in the current catalog. (See “About the Current Catalog” on page 916.)

To open the DIRECTORY window in edit mode, or to specify a different catalog, include the CATALOG= option in the PROC GDEVICE statement.

From the DIRECTORY window you can select the device entry you want to work with and open other GDEVICE windows in which you can view or modify device parameters. For more information, see “Using the GDEVICE Windows” on page 928.

In a windowing environment, you can switch between the GDEVICE windows and program statements while you are running the procedure. See the “FS Statement” on page 925 and the NOFS window command in the SAS Help facility for SAS/GRAPH.

To exit the GDEVICE windows, submit the End command or close the window.

Program Mode

If you are in a non-windowing or batch environment, the GDEVICE procedure automatically starts in program mode. If you are in a windowing environment, specify the NOFS option to start the GDEVICE procedure in program mode:

```
proc gdevice nofs;
```

By default, the GDEVICE procedure accesses the current catalog in browse mode and prompts you in the LOG to enter additional program statements. (See “About the Current Catalog” on page 916.) To specify the current catalog, include the CATALOG= option in the PROC GDEVICE statement.

Once you start the GDEVICE procedure, you can enter and run additional statements without re-entering the PROC GDEVICE statement. For example, the following statement generates a listing of the device parameters for the PSCOLOR device entry that is stored in the Institute-supplied catalog, SASHELP.DEVICES:

```
list pscolor;
```

PROC GDEVICE procedure output is displayed in the Output window. Output 31.1 shows the listing generated by the LIST statement.

Output 31.1 Sample Device Entry Listing Generated in Program Mode

```

                                GDEVICE procedure
                                Listing from SASHELP.DEVICES - Entry PSCOLOR

Orig Driver: PSCOLOR           Module: SASGDPSL   Model: 1251
Description: PostScript color--RGB color defs      Type: PRINTER
*** Institute-supplied ***
Lrows: 0 Xmax: 8.500 IN      Hsize: 8.000 IN  Xpixels: 2550
Lcols: 0 Ymax: 11.000 IN    Vsize: 8.500 IN  Ypixels: 3300
Prows: 68                   Horigin: 0.218 IN
Pcols: 80                   Vorigin: 1.496 IN
Aspect: 0.000               Rotate:
Driver query: Y             Queued messages: N
                                Paperfeed: 0.000 IN

OPTIONS

Erase:                       Autofeed: Y           Chartype: 1
Swap:                         Cell:                Maxcolors: 256
Autocopy:                     Characters:          Repaint: 0
Handshake: XONXOFF           Circlearc:          Gcopies: 0
                                Dash:                Gsize: 0
Prompt - startup:            Fill:                Speed: 0
                                end graph:          Piefill:            Fillinc: 0
                                mount pen:          Polyfill:           Maxpoly: 1450
                                chg paper:         Symbol:             Lfactor: 0
                                Pensort: N

Promptchars: '000A010D05000000'X
Devopts: 'FD9230402C130000'X
UCC: '0001'X

Cback: WHITE
Color list:

    BLACK    RED    GREEN    BLUE    CYAN
    MAGENTA  YELLOW  GRAY

CHARTYPE RECORDS

Chartype Rows  Cols          Font Name                Scalable
   1      89    85    Courier                Y
   2      89    85    Courier-Oblique        Y
   .
   {ob ...more hardware fonts...}
   .
  34      89    85    Bookman-LightItalic    Y
  35      89    85    Bookman-DemiItalic     Y
Gend: '0A'X

FILE INFORMATION

Gaccess: sasgastd>sasgraph.ps
Gsfname:
Trantab: Gsfmode: PORT      Gsflen: 0
Devmap:
Devtype: DISK
Gprotocol:
Fileclose: DRIVERTERM
Hostspec:

HOST INFORMATION

```

You can exit the GDEVICE procedure in these three ways:

- Submit the END, QUIT, or STOP statement.
- Submit another PROC statement or DATA step.
- Exit your SAS session.

Procedure Syntax

Requirements: Statements other than the PROC GDEVICE statement can be used only in a non-windowing or batch environment. In these environments, at least one statement is required to give GDEVICE an action to perform. In a windowing environment, only the PROC GDEVICE statement is required. In program mode, at least one additional statement is required, and you can submit as many of each statement as you want.

Note: You must have write access to the device catalog in order to modify, add, or delete entries.

Supports: Output Delivery System (ODS LISTING).

```

PROC GDEVICE <CATALOG=<libref.>SAS-catalog>
  <BROWSE>
  <NOFS>;

ADDnew-device-entry
  required-parameters
  <optional-parameters>;

COPYdevice-entry
  <FROM=<libref.>SAS-catalog>
  <NEWNAME=new-device-entry>;

DELETE device-entry;

FS;

LIST device-entry | _ALL_ | _NEXT_ | _PREV_ | DUMP>;

MODIFY device-entry
  parameter(s);

QUIT | END | STOP;

RENAME device-entry NEWNAME=new-entry-name;

```

PROC GDEVICE Statement

Starts the procedure and determines whether it runs in windowing mode or program mode. Optionally identifies a device catalog and determines how that catalog is opened.

```

PROC GDEVICE <CATALOG=<libref.>SAS-catalog>
  <BROWSE>
  <NOFS>;

```

Options

Options used in the PROC GDEVICE statement affect the way you use the procedure.

BROWSE

opens a catalog in browse mode. You cannot modify a catalog when you open it with the BROWSE option. If you are running in program mode when you use BROWSE, you can use only the FS, LIST, QUIT, END, or STOP statements.

CATALOG=<libref.>*SAS-catalog*

CAT=<libref.>*SAS-catalog*

C=<libref.>*SAS-catalog*

specifies the catalog containing device information. If you do not specify a catalog, the procedure opens the first catalog found in the search order of catalogs in browse mode. (See “About the Current Catalog” on page 916. for information on how the GDEVICE procedure determines which catalog to use.)

To edit the device entries in a catalog, you must use the CATALOG= option.

NOFS

specifies that you are using program mode. In windowing environments, the GDEVICE windows are the default and you must specify NOFS to start GDEVICE in program mode.

ADD Statement

Adds a new device entry to the catalog selected by the CATALOG= option in the PROC GDEVICE statement. The device entry is initialized with NULL values for most parameters.

Requirements: You must have write access to the device catalog in order to add entries, and use CATALOG= in the PROC GDEVICE statement.

Restriction: Not valid in browse mode.

ADD *new-device-entry*
required-parameters
 <*optional-parameters*>;

required-parameters are all of the following:

MODULE=*driver-module*
 XMAX=*width* <IN | CM>
 YMAX=*height* <IN | CM>
 XPIXELS=*width-in-pixels*
 YPIXELS=*height-in-pixels*

plus one or both of the following parameter pairs:

LCOLS=*landscape-columns*
 LROWS=*landscape-rows*

or

PCOLS=*portrait-columns*
 PROWS=*portrait-rows*

optional-parameters can be one or more of the following:

ASPECT=*scaling-factor*
 AUTOCOPY=Y | N
 AUTOFEED=Y | N
 CBACK=*background-color*
 CELL=Y | N

CHARACTERS=Y | N
 CHARREC=(*charrec-list(s)*)
 CHARTYPE=*hardware-font-chartype*
 CIRCLEARC=Y | N
 CMAP=(*'from-color : to-color' <...,'from-color-n : to-color-n'>*)
 COLORS=(*<colors-list>*)
 COLORTYPE=NAME | RGB | HLS | GRAY | CMY | CMYK | HSV | HSB
 DASH=Y | N
 DASHLINE=*'dashed-line-hex-string'*X
 DESCRIPTION=*'text-string'*
 DEVMAP=*device-map-name* | NONE
 DEVOPTS=*'hardware-capabilities-hex-string'*X
 DEVTYPE=*device-type*
 DRVINIT1=*'system-command(s)'*
 DRVINIT2=*'system-command(s)'*
 DRVQRY | NODRVQRY
 DRVTERM1=*'system-command(s)'*
 DRVTERM2=*'system-command(s)'*
 ERASE=Y | N
 FILECLOSE=DRIVERTERM | GRAPHEND
 FILL=Y | N
 FILLINC=0...9999
 FORMAT=CHARACTER | BINARY
 GACCESS=*output-format* | *'output-format > destination'*
 GCOPIES=*current-copies*
 GEND=*'string' <...'string-n'>*
 GEPILOG=*'string' <...'string-n'>*
 GPROLOG=*'string' <...'string-n'>*
 GPROTOCOL=*module-name*
 GSFLLEN=*record-length*
 GSFMODE=APPEND | REPLACE | PORT
 SFNAME=*fileref*
 GSIZE=*lines*
 GSTART=*'string' <...'string-n'>*
 HANDSHAKE=HARDWARE | NONE | SOFTWARE | XONXOFF
 HEADER=*command'*
 HEADERFILE=*fileref*
 ORIGIN=*horizontal-offset* <IN | CM>
 HOSTSPEC=*'text string'*
 HSIZE=*horizontal-size* <IN | CM>
 ID=*'description'*
 INTERACTIVE=USER | GRAPH | PROC
 LFACTOR=*line-thickness-factor*
 MAXCOLORS=*number-of-colors*
 MAXPOLY=*number-of-vertices*

MODEL=*model-number*
 NAK=*'negative-handshake-response'*X
 PAPERFEED=*feed-increment* <IN | CM>
 PATH=*angle-increment*
 PENSORT=Y | N
 PIEFILL=Y | N
 POLYGONFILL=Y | N
 POSTGRAPH1=*'system-command(s)'*
 POSTGRAPH2=*'system-command(s)'*
 PREGRAPH1=*'system-command(s)'*
 PREGRAPH2=*'system-command(s)'*
 PROCESS=*'command'*
 PROCESSINPUT=*fileref*
 PROCESSOUTPUT=*fileref*
 PROMPT=0...7
 PROMPTCHARS=*'prompt-chars-hex-string'*X
 QMSG | NOQMSG
 RECTFILL=*'rectangle-fill-hex-string'*X
 REPAINT=*redraw-factor*
 ROTATE=LANDSCAPE | PORTRAIT
 ROTATION=*angle-increment*
 SPEED=*pen-speed*
 SWAP=Y | N
 SYMBOL=Y | N
 SYMBOLS=*'hardware-symbols-hex-string'*X
 TRAILER=*'command'*
 TRAILERFILE=*fileref*
 TRANTAB=*table* | *user-defined-table*
 TYPE= CAMERA | CRT | EXPORT | PLOTTER | PRINTER
 UCC=*'control-characters-hex-string'*X
 VORIGIN=*vertical-offset* <IN | CM>
 VSIZE=*vertical-size* <IN | CM>

Required Arguments

new-device-entry

specifies the one-level name of the new device entry. *New-device-entry* must be a valid name for a SAS catalog entry for your operating environment and cannot already exist in the current catalog.

required-parameters

all required parameters for the ADD statement correspond to device parameters of the same name. Refer to Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for a description of each parameter.

Options

All optional parameters for the ADD statement correspond to device parameters of the same name. Refer to Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for a description of each parameter.

Note: The COLORS= device parameter is not required; the device entry will be created if you do not use it. However, the GDEVICE procedure issues an error message if you do not specify at least one color for COLORS=. △

Details

The ADD statement is rarely used because it initializes parameter values to NULL and you have to set values for all the parameters. The best way to add a new driver is to copy an existing driver and modify it.

COPY Statement

Copies a device entry and places the copy in the current catalog. The original device entry can be either in the current catalog or in a different catalog.

Requirements: You must have write access to the catalog to which the device entry is being copied.

Restriction: Not valid in browse mode.

See also: “Creating or Modifying Device Entries” on page 934

Featured in: Example 1 on page 936

COPY *device-entry where*;

Where *where* must be one or both of the following:

FROM=<*libref.*>SAS-catalog

NEWNAME=*new-device-entry*

Required Arguments

device-entry

specifies the one-level name of the device entry to copy. The entry must exist in either the current catalog (the default) or the catalog specified by FROM=.

FROM=<*libref.*>SAS-catalog

names the catalog from which to copy *device-entry*.

NEWNAME=*new-device-entry*

specifies a name for the copy of the device entry that is placed in the current catalog. *New-device-entry* must be a valid name for a SAS catalog entry and cannot already exist in the current catalog.

If you copy device entries across catalogs and you do not specify a new name, the GDEVICE procedure uses the original name for the new device entry.

DELETE Statement

Deletes the device entry from the current catalog.

Requirements: You must have write access to the current catalog to delete a device entry from it, and use CATALOG= in the PROC GDEVICE statement.

Restriction: Not valid in browse mode.

Caution: A device entry cannot be restored once it has been deleted. Depending on the environment in which you are using the GDEVICE procedure, you may be asked to verify that you really want to delete the entry.

```
DELETE device-entry;
```

Required Arguments

device-entry

specifies the one-level name of device entry to delete. The entry must exist in the current catalog.

FS Statement

Switches from program mode to the GDEVICE windows.

Requirements: You must be running SAS software in a windowing environment.

```
FS;
```

Options

No options.

LIST Statement

Lists all of the parameters of the specified device entry in the Output window.

Default: `_ALL_`

See also: “Program Mode” on page 918

```
LIST <device-entry>
    <_ALL_>
    <_NEXT_>
```

```
<_PREV_>
<DUMP>;
```

Options

device-entry

specifies the one-level name of the device entry whose contents you want to list. The entry must exist in the current catalog.

ALL

lists only the name, description, and creation date of all device entries in the current catalog. This is the default. If no entries exist in the catalog, the GDEVICE procedure issues a message.

NEXT

lists the contents of the next device entry. The GDEVICE procedure lists the first entry in the catalog if no entries have been previously listed.

PREV

lists the contents of the previous device entry. If you have not previously listed the contents of a device entry, the GDEVICE procedure issues the following message:

```
No objects preceding current object.
```

DUMP

lists detailed information on *all* device entries in the current catalog. Depending on the number of device entries in the catalog, the DUMP option can create a *large* amount of output.

MODIFY Statement

Changes the values in a device entry.

Requirements: You must have write access to the current catalog to modify a device entry, and use CATALOG= in the PROC GDEVICE statement.

Restriction: Not valid in browse mode.

See also: “Creating or Modifying Device Entries” on page 934

Featured in: Example 1 on page 936

```
MODIFY device-entry
      parameter(s);
```

Required Arguments

device-entry

specifies the one-level name of the device entry that you want to modify. The entry must exist in the current catalog.

parameter(s)

are the parameters you want to modify. These can be any of the parameters listed in the ADD statement, whether listed as required or optional for ADD. See “ADD Statement” on page 921 for a complete list. Refer to Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for a description of each parameter.

Details

To modify a device entry, create your own catalog and then copy the device entries you need into it. You can then change your personal copies of the device entries without affecting the original drivers in SASHELP.DEVICES. (To copy device entries, use the COPY statement, the COPY command available after you choose Import Device Entry from the DIRECTORY window’s File menu, or the CATALOG procedure, which is part of base SAS.

CAUTION:

Be careful when modifying device entries in program mode. In program mode, you cannot cancel any modifications you have just made. To change a value you have modified, you must use another MODIFY statement to replace the original value or reset it to its default. (In the GDEVICE windows, you can type the CANCEL command in the command line to cancel changes you have made to the fields.) △

QUIT Statement

Saves all modifications made to device entries during the procedure and exits the GDEVICE procedure.

QUIT | *END* | *STOP*;

Options

No options.

RENAME Statement

Changes the name of the device entry to the name specified in the statement.

Requirements: You must have write access to the current catalog to rename a device entry, and use CATALOG= in the PROC GDEVICE statement.

Restriction: Not valid in browse mode.

RENAME *device-entry*
NEWNAME=*new-entry-name*;

Required Arguments

device-entry

specifies the one-level name of the device entry that you want to rename. The entry must exist in the current catalog.

NEWNAME=*new-entry-name*

specifies the new entry name. *New-entry-name* must be a valid name for a SAS catalog entry and cannot already exist in the current catalog. If the name already exists, the GDEVICE procedure issues an error message.

Using the GDEVICE Procedure

Using the GDEVICE Windows

You can use the GDEVICE windows instead of program mode to view, modify, copy, create, or delete device entries. You perform tasks in the GDEVICE windows by entering values in the fields, by using the pulldown menus, and by issuing commands from the command line.

These are the thirteen GDEVICE windows in order of appearance:

- Directory Window
- Detail Window
- Parameters Window
- Gcolors Window
- Chartype Window
- Colormap Window
- Metagraphics Window
- Gprolog Window
- Gepilog Window
- Gstart Window
- Gend Window
- Host File Options Window
- Host Commands Window

The fields in these windows represent device entry parameters. The GDEVICE windows group the device parameters by topic, to make it easy for you to review or modify the entry. If you open the device entry in edit mode, you can modify the fields directly. For a description of each field, see the corresponding parameter in Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 or refer to the SAS Help facility. For a complete list of device parameters, see “ADD Statement” on page 921.

Note: The parameters are sometimes an abbreviation of the field names, but the correspondence should be clear. For example, in the Detail window, the "Driver query" field corresponds to the DRVQRY parameter, and the "Queued messages" field corresponds to the QMSG parameter. \triangle

This section briefly describes the GDEVICE windows; for a complete description of each window and its fields, refer to the SAS Help facility.

GDEVICE Window Commands

You can navigate and manipulate the GDEVICE windows by entering commands on the command line or selecting them from the menus. For a complete description of all the GDEVICE window commands, refer to the SAS Help facility.

Note: In a Windows environment, the GDEVICE commands are presented on pop-up menus rather than on the menubar. Right-click a GDEVICE window to access a pop-up menu. \triangle

GDEVICE Window Descriptions

DIRECTORY Window

This window appears when you start the GDEVICE procedure in window mode. It lists all the device entries in the default catalog or the catalog you specified in the PROC GDEVICE statement. You can use it to

- copy, rename, or delete device entries in the catalog
- select a device entry whose parameters you want to browse or edit.

You can enter these commands in the Directory window selection field:

B | S

open the Detail window and browse (B) or, if you are in edit mode, edit (S) the selected device entry.

D

delete the selected device entry. You cannot restore a device entry once it has been deleted.

E

open the Detail window and edit the selected device entry.

R

rename the device entry and/or description.

You cannot edit the TYPE and UPDATED fields in the Directory Window.

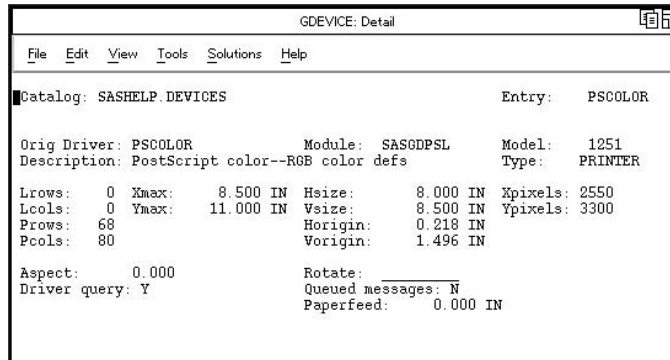
Figure 31.1 The DIRECTORY Window

GDEVICE: DIRECTORY SASHELP.DEVICES (B)			
File Edit View Tools Solutions Help			
Name	Type	Description	Updated
- PHASERM	DEV	Tektronix Phaser II Pxi - Special A4	04/07/98
- PHASR340	DEV	Tektronix Phaser 340 Color Printer	04/07/98
- PHASR540	DEV	Tektronix Phaser 540 Color Printer	04/07/98
- PHASRIII	DEV	Phaser III Pxi PostScript Printer	04/07/98
- PHSR340M	DEV	Tektronix Phaser 340 Color Printer -- A4	04/07/98
- PHSR540M	DEV	Tektronix Phaser 540 Color Printer -- A4	04/07/98
- PNG	DEV	PNG (Portable Network Graphics) Format	04/07/98
- PROPRINT	DEV	IBM PROPRINTER	04/07/98
- PROPRNXL	DEV	IBM PROPRINTER XL	04/07/98
- PS	DEV	PostScript devices	04/07/98
- PS1200	DEV	PostScript devices--thin lines, 1200 DPI	04/07/98
- PS1200A4	DEV	PostScript devices--1200 DPI--A4	04/07/98
- PS300	DEV	PostScript devices--thin lines, 300 DPI	04/07/98
- PS300A4	DEV	PostScript--thin lines--A4 size paper	04/07/98
- PS5232	DEV	Schlumberger Color Postscript Printer	04/07/98
- PS600	DEV	PostScript devices--thin lines, 600 DPI	04/07/98
- PS720	DEV	PostScript devices--thin lines, 720 DPI	04/07/98
- PS720A4	DEV	PostScript devices--720 DPI--A4	04/07/98
- PSCAL	DEV	Calcomp Colormaster Plus Printer	04/07/98
- PSCLRM4	DEV	PostScript color--RGB color defs--A4	04/07/98
- PSCLRSEP	DEV	PostScript experimental color separator	04/07/98
█ PSCOLOR	DEV	PostScript color--RGB color defs	04/07/98
- PSCSEPL	DEV	PostScript experimental color separation	04/07/98

Detail window

This window contains device parameters that control basic characteristics of the device, for example, the size of the graphics output area.

Figure 31.2 The Detail Window



From this window you can access any of the subsidiary GDEVICE windows by

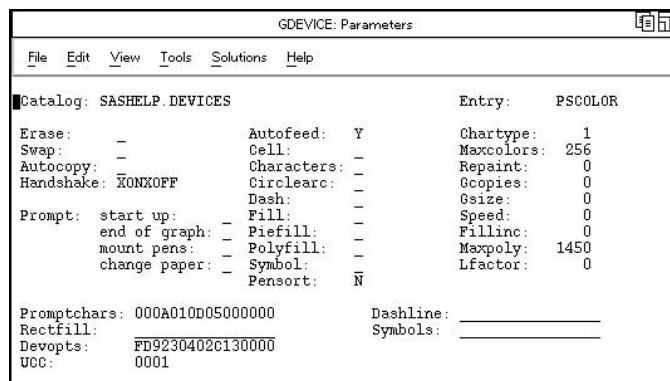
- entering the name of the window on the command line
- selecting the window from the Tools pulldown
- opening the subsidiary windows in order of appearance by using the View pulldown and choosing Next Screen, or using the NEXTSCR command on the command line.

Parameters window

This window includes additional device parameters that affect the way graphs are drawn. For example, you choose whether certain graphics primitives are drawn by your hardware or by SAS/GRAPH software, whether to feed paper to printers or plotters automatically, and whether to have SAS/GRAPH software prompt you with messages under certain conditions.

Note: If the device does not support a hardware characteristic, the catalog entry cannot enable the support. \triangle

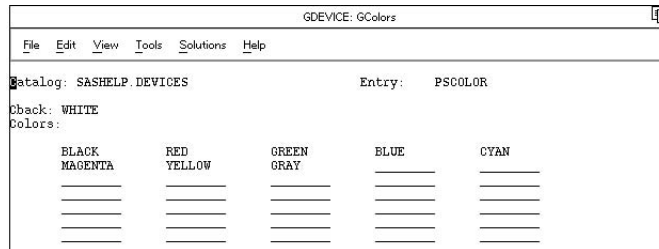
Figure 31.3 The Parameters Window



Gcolors window

This window lists the colors that the device driver uses by default. When you do not explicitly specify the color of a graphics feature in your program or in a GOPTIONS statement, SAS/GRAPH software uses this list to determine what color to use.

Figure 31.4 The Gcolors Window (partial view)



Chartype window

This window lists the hardware fonts that the device can use, along with information about the size of the characters. The Chartype value is the value you can use to reference a font in another window. For example, you would enter a Chartype number in the Parameters window's Chartype field.

Figure 31.5 The Chartype Window (partial view)

The screenshot shows a window titled "GDEVICE: Chartype". It has a menu bar with "File", "Edit", "View", "Tools", "Solutions", and "Help". Below the menu bar, it displays "Catalog: SASHELP.DEVICES" and "Entry: PSCOLOR". The main content is a table with the following columns: Chartype, Rows, Cols, Font Name, and Scalable.

Chartype	Rows	Cols	Font Name	Scalable
1	89	85	Courier	Y
2	89	85	Courier-Oblique	Y
3	89	85	Courier-Bold	Y
4	89	85	Courier-BoldOblique	Y
5	89	85	Times-Roman	Y
6	89	85	Times-Italic	Y
7	89	85	Times-Bold	Y
8	89	85	Times-BoldItalic	Y
9	89	85	Helvetica	Y
10	89	85	Helvetica-Oblique	Y
11	89	85	Helvetica-Bold	Y
12	89	85	Helvetica-BoldOblique	Y

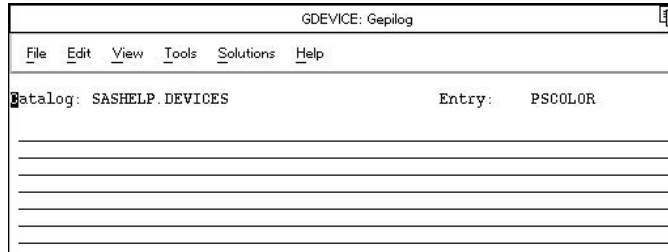
Colormap window

This window allows you to specify a color map for the device. The FROM field specifies the name to assign to the color designated by the *color* value, and the TO field specifies a SAS/GRAPH color name up to eight characters long. Once you have defined the color mapping, you can use the new color name in any color option. For example, if your device entry maps the color name DAFFODIL to the SAS color value PAOY, you can specify COLOR=DAFFODIL on any statement that supports a COLOR= option, and the driver will map this to the color value PAOY.

Gepilog window

This window enables you to specify one or more hexadecimal strings that are sent to the device just after graphics commands are sent. Additional commands can be sent with the PREGEPILOG= and POSTGEPILOG= graphics options. See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for details.

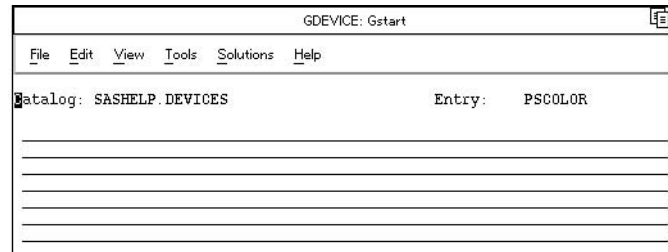
Figure 31.9 The Gepilog Window (partial view)



Gstart window

This window enables you to specify one or more hexadecimal strings that are placed at the beginning of each record of graphics data.

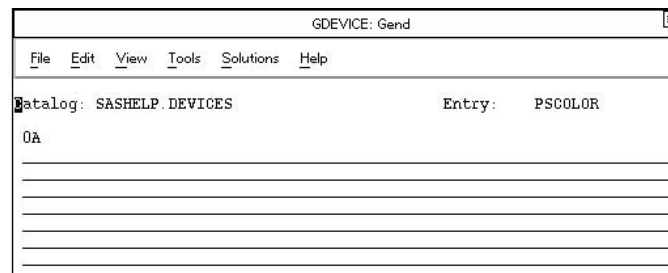
Figure 31.10 The Gstart Window (partial view)



Gend window

This window enables you to specify one or more hexadecimal strings that are placed at the end of each record of graphics data.

Figure 31.11 The Gend Window (partial view)

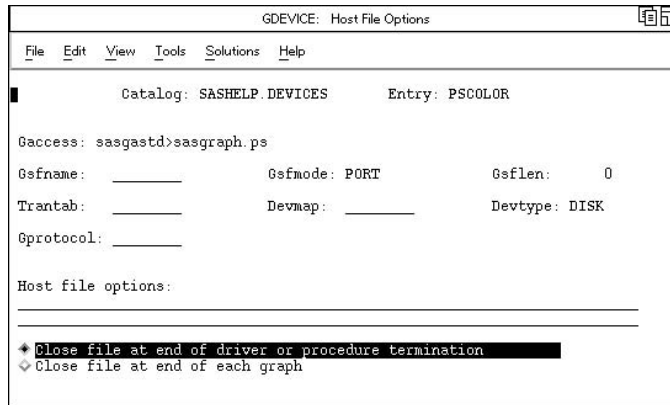


Host File Options window

This window controls the output destination and formatting of the data stream produced by the driver. (Most of these values can also be specified with the GOPTIONS

statement and with the FILENAME statement. See also “Exporting SAS/GRAPH Output with Program Statements” on page 62.)

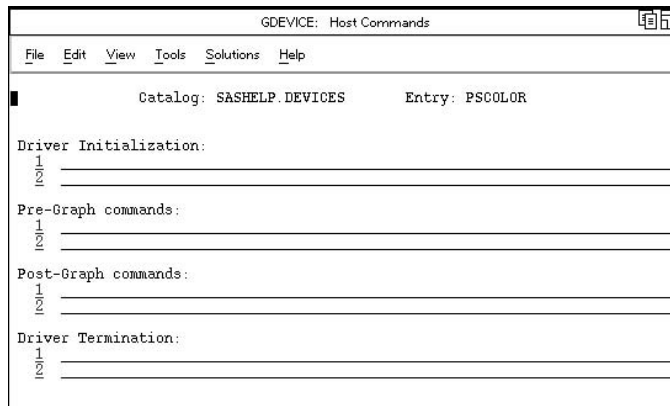
Figure 31.12 The Host File Options Window



Host Commands window

This window stores the host commands issued at driver initialization, before and after each graph is produced, and at driver termination. These commands are typically used to send graphics output to a hardcopy device such as a printer or a plotter.

Figure 31.13 The Host Commands Window



Creating or Modifying Device Entries

In order to add, modify, or delete device entries, you must have write access to the catalog. On multi-user systems, the SAS support consultant is usually the only person who has write access to the SASHELP.DEVICES catalog and can make any changes. Therefore, when creating new entries or modifying existing ones, individual users usually work in a personal catalog. Be sure the catalog in which you store new or modified device entries is named DEVICES.

To use a device entry stored in a personal catalog, you must assign the GDEVICE n libref to the library that contains the device catalog. See “About Device Catalogs” on page 916.

It is a good idea to give a new or modified device entry a name that is different from the original. Then, if you want to use the original device, SAS/GRAPH can find that device when it searches the device catalogs. Remember that SAS/GRAPH searches the GDEVICE n libraries *before* it searches SASHELP.DEVICES and uses the first device it finds whose name matches the one you have specified. (See “Search Order of Device Catalogs” on page 917.)

For example, suppose there is a customized copy of PSCOLOR in your GDEVICE0.DEVICES catalog as well as the original in SASHELP.DEVICES. If you specify DEV=PSCOLOR and if the libref GDEVICE0 is assigned, SAS/GRAPH will search GDEVICE0.DEVICES first and use the copy of PSCOLOR stored there. Unless you cancel the GDEVICE0 libref, SAS/GRAPH will never find the original in SASHELP.DEVICES.

Creating a New Device Entry

Typically you create a new device entry by copying an existing device and modifying its parameters to suit your needs. You can copy and modify a device entry in two ways:

- Use the DIR command on the command line to open the DIRECTORY window, and then use the COPY command to make a copy of an existing device entry. Then edit the new entry and modify its parameters. The existing device entry can be from any catalog. (See the SAS Help facility for information on using GDEVICE windows and commands.)
- In program mode, use the COPY statement to make a copy of the device entry and use the MODIFY statement to change the parameters (see Example 1 on page 936).

If you want to start with a blank device entry and fill in values for the parameters, use the EDIT command from the DIRECTORY window or use the ADD statement with program mode PROC GDEVICE.

With either method, you must provide values for the parameters listed in “Required Arguments” on page 923. If you copy and modify an existing entry, all the required parameters will already have values. If you create a new entry with GDEVICE windows, you are prompted to fill in the appropriate fields.

Note: When you change a field in an Institute-supplied device entry (either the original device entry in SASHELP.DEVICES or a copy), SAS/GRAPH software asks whether you really want to change the entry. Answer Y to change the entry or N to cancel the operation. Δ

Modifying an Existing Device Entry

Typically, you modify an existing device entry when you want to change the device parameters permanently in order to customize a device entry. The process is similar to creating a new entry in that you usually begin by copying the entry you want to modify into your personal catalog and making the changes there. See Example 1 on page 936 for an example of creating a custom device entry.

Changing Device Parameters Temporarily

You can change some device parameters temporarily by overriding their settings with graphics options in a GOPTIONS statement. In this case, the settings remain in effect until you change them or end your SAS session. For details, see “Overriding Device Parameters Temporarily” on page 46.

Examples

The following examples illustrate major features of the GDEVICE procedure.

Example 1: Creating a Custom Device Entry with Program Statements

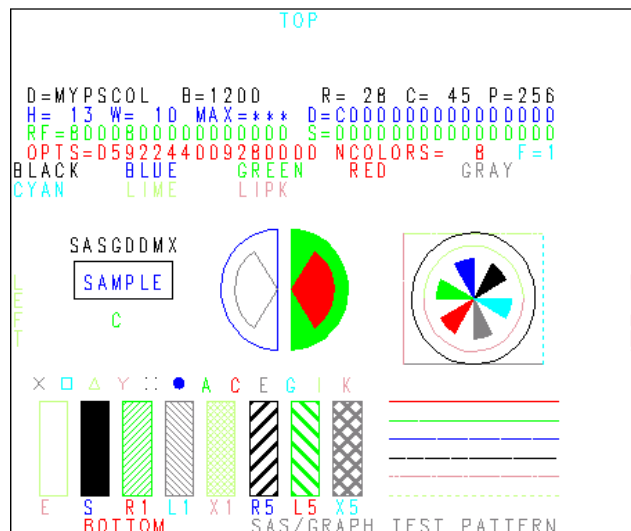
Procedure features:

COPY statement
MODIFY statement

Other features:

PROC GTESTIT

Sample library member: GDVCSTOM



This example shows how to use GDEVICE procedure statements to modify a device entry by copying the original entry into a personal catalog and changing the device parameters. You can submit these statements one at a time or together.

This example permanently changes the default colors list for the PSCOLOR device entry. The contents of the original PSCOLOR entry are shown in Output 31.1. The new device entry is illustrated in the PROC GTESTIT output above.

Assign the libref GDEVICE0. The LIBNAME statement assigns the libref to the aggregate file storage location that contains (or will contain) the DEVICES catalog.

```
libname gdevice0 'SAS-data-library';
```

Start the GDEVICE procedure. NOFS causes GDEVICE to use program mode. CATALOG= assigns GDEVICE0.DEVICES as the current catalog. If the DEVICES catalog does not already exist in the library, it is automatically created.

```
proc gdevice nofs catalog=gdevice0.devices;
```

Copy the original device entry from SASHELP.DEVICES to the current catalog.

NEWNAME= specifies a name for the copy of PSCOLOR that is placed in GDEVICE0.DEVICES. The name of a catalog entry cannot exceed eight characters.

```
copy pscolor from=sashelp.devices newname=mypscol;
```

Modify the new entry. DESCRIPTION= specifies a new device description that appears in the catalog listing. COLORS= defines a new colors list.

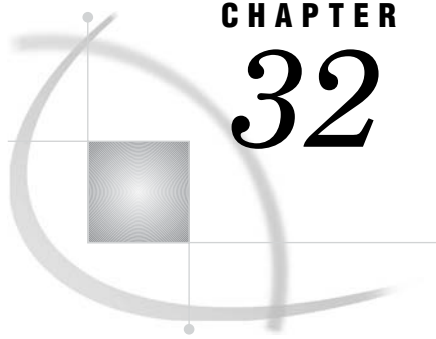
```
modify mypscol
  description='PSCOLOR with new colors list'
  colors=(black blue green red gray cyan
          lime lipk);
```

Exit the procedure.

```
quit;
```

Test the new device entry. The TARGET= graphics option specifies the new device. Since GDEVICE0 is already defined, SAS/GRAPH looks first in that catalog for the specified device entry. The GTESTIT procedure produces a test picture that show the new colors list and a listing in the LOG.

```
goptions target=mypscol;
proc gtestit pic=1;
run;
```

CHAPTER 32

The GFONT Procedure

<i>Overview</i>	939
<i>About Displaying Fonts</i>	940
<i>About Creating Fonts</i>	940
<i>Concepts</i>	940
<i>About Fonts</i>	940
<i>About the Libref GFONTO</i>	941
<i>Procedure Syntax</i>	942
<i>PROC GFONT Statement</i>	942
<i>Creating a Font</i>	951
<i>The Font Data Set</i>	951
<i>Font Data Set Variables</i>	952
<i>Creating a Font Data Set</i>	958
<i>The Kern Data Set</i>	958
<i>Kern Data Set Variables</i>	959
<i>Creating a Kern Data Set</i>	959
<i>The Space Data Set</i>	960
<i>Space Data Set Variables</i>	960
<i>Creating a Space Data Set</i>	961
<i>Examples</i>	962
<i>Example 1: Displaying Fonts and Character Codes</i>	962
<i>Example 2: Creating Figures for a Symbol Font</i>	964

Overview

The GFONT procedure displays new or existing fonts and creates user-generated fonts for use in SAS/GRAPH programs. These fonts can contain standard Roman alphabet characters, foreign language characters, symbols, logos, or figures.

The GFONT procedure

- displays SAS/GRAPH software fonts
- displays fonts that were previously generated with the GFONT procedure (user-generated fonts)
- displays hardware font that are available on your device and have a corresponding Chartype value
- displays the character codes or hexadecimal values that are associated with the characters in a font
- creates stroked fonts or polygon fonts.

Each of these activities has its own requirements, its own process, and its own options (although some options are valid for either process). In this chapter, each topic

to which this distinction applies is divided into two sections: "Displaying Fonts" and "Creating Fonts."

About Displaying Fonts

You can use the GFONT procedure to display a font when you want to do one of the following:

- review the characters that are available in Institute-supplied fonts, hardware fonts, or user-generated fonts
- see the character codes or the hexadecimal values that are associated with the characters in a font.

When you display a font, you can modify the color and height of displayed font characters, draw reference lines around the characters, or display the associated character codes or hexadecimal values. See Example 1 on page 962.

About Creating Fonts

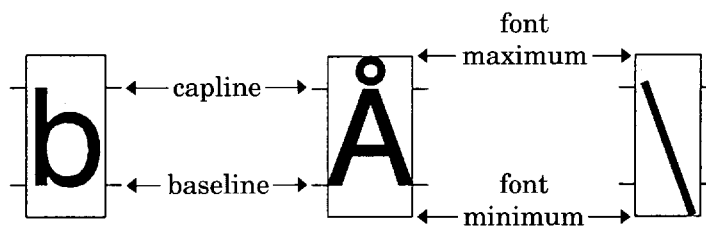
You can use the GFONT procedure to create and store fonts of your own design. The GFONT procedure is not limited to creating alphabet fonts. You can use it to create and store any series of figures that you can draw using X and Y coordinates or that you can digitize. The characters or figures in a font can be displayed with any SAS/GRAPH statement or option that allows for font specification and a text string (for example, a TITLE statement). See "Creating a Font" on page 951 for details.

Concepts

About Fonts

Some specialized terms are associated with font characteristics. The *capline* of a font is the highest point of a normal uppercase letter. The *baseline* is the line upon which the characters rest. The *font maximum* is the highest vertical coordinate in a font. The *font minimum* is the lowest vertical coordinate in a font. Figure 32.1 on page 940 illustrates these GFONT procedure terms:

Figure 32.1 Parts of a Font



Specialized terms also exist for types of fonts. The term *uniform font* refers to a font in which all of the characters occupy exactly the same amount of space, even though the

characters themselves are different sizes. Each character in a uniform font is placed in the center of its space, and a fixed amount of space is added between characters. A *proportional font* is a font in which each character occupies a space that is proportional to its actual width (for example, m occupies more space than i). The characters in a *stroked font* are drawn with discrete line segments or circular arcs. Figure 32.2 on page 941 illustrates a stroked font with several characters from the Simplex font.

Figure 32.2 Characters from a Stroked Font

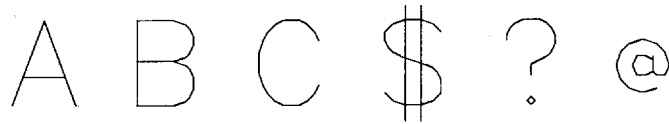
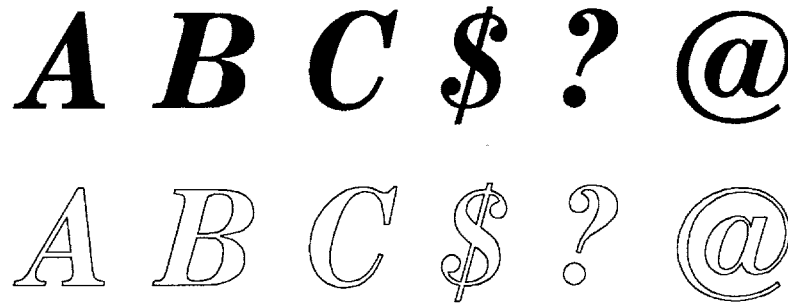


Figure 32.3 on page 941 illustrates two types of *polygon fonts*: filled (CENTBI) and outline (CENTBIE). A *filled font* is a polygon font in which the areas between the lines are solid. An *outline font* is a polygon font in which the areas are empty.

Figure 32.3 Filled and Outline Characters from Polygon Fonts



All font characters, regardless of whether they are stroked or polygon, are drawn with line segments. In the GFONT procedure, the term *line segment* means a continuous line that can change direction. For example, the letter C in Figure 32.2 on page 941 is drawn with one line segment, while the letter A can be drawn with two.

Polygon characters can also be drawn with one or more line segments. In a polygon font, one character can be made up of a single polygon, multiple polygons, or polygons with holes. For example, the letter C in Figure 32.3 on page 941 is a single polygon with one line segment. The question mark (?) is made up of two polygons, each drawn with a separate line segment. The letter A is one polygon with a hole in it. It is drawn with one line segment that is broken to form the outer boundary of the figure and the boundary of the hole.

About the Libref GFONTO

The GFONT procedure stores user-generated fonts in the location that is associated with the libref GFONTO. Therefore, before you create a font or display a user-generated font, you must submit a LIBNAME statement that associates the libref GFONTO with the location where the font is to be stored, as follows:

```
libname gfont0 'SAS-data-library';
```

Since the GFONT0 library is the first place that SAS/GRAPH software looks for fonts, you should always assign that libref to the library that contains your personal fonts. If for some reason you have personal fonts in more than one SAS data library, assign them librefs in the sequence GFONT0, GFONT1, GFONT2, and so forth. The search for entries terminates if there is a break in the sequence; the catalog GFONT1.FONTS is not checked if the libref GFONT0 is undefined. If the libref GFONT0 is not defined, by default SAS/GRAPH software begins searching for fonts in SASHELP.FONTS.

To cancel or redefine the libref GFONT n , submit the following statement:

```
goptions reset=all fcache=0;
```

Note that when you specify RESET=ALL, all graphics options are reset to their default values. Once you have cleared the font cache, you can redefine the libref with another LIBNAME statement.

Procedure Syntax

Requirements: A font name is required. To display a font, include NOBUILD. To create a font, include DATA=.

Global statements: FOOTNOTE, TITLE

Reminder: The procedure can include the SAS/GRAPH NOTE statement.

Supports: Output Delivery System (ODS)

```
PROC GFONT NAME=font-name | hardware-font-name
           mode
           <display-option(s)>
           <creation-option(s)>;
```

PROC GFONT Statement

The PROC GFONT statement can either create user-defined fonts or display existing software fonts. Therefore, it names the font to be created or displayed. If the procedure creates a font it names the input data set. Optionally, the procedure modifies the design and appearance of the fonts that you create or display, and specifies a destination catalog for graphics output.

Syntax

```
PROC GFONT NAME=font-namehardware-font-name
           mode
           <display-option(s)>
           <creation-option(s)>;
```

- *mode* must be one of the following:

```
DATA=font-data-sethardware-font-name
NOBUILD
```

- *display-option(s)* can be one or more of the following:

CTEXT=*text-color*
 GOUT=<*libref.*>*output-catalog*
 HEIGHT=*character-height*<*units*>
 NOKEYMAP
 NOROMAN
 NOROMHEX
 REFCOL=*reference-line-color*
 REFLINES
 ROMCOL=*code-color*
 ROMFONT=*font*
 ROMHEX
 ROMHT=*height*<*units*>
 SHOWALL
 SHOWROMAN

- *creation-option(s)* can be one or more of the following:

BASELINE=*y*
 CAPLINE=*y*
 CHARSPACETYPE=DATA | FIXED | NONE | UNIFORM
 CODELEN=1 | 2
 FILLED
 KERNDATA=*kern-data-set*
 MWIDTH=*character-width*
 NODISPLAY
 NOKEYMAP
 RESOL=1...4
 ROMHEX
 SHOWROMAN
 SPACEDATA=*space-data-set*
 UNIFORM

For more detail on using the GFONT syntax, see “Displaying Fonts: Required Arguments, Options” on page 943 and “Creating Fonts: Required Arguments, Options” on page 946.

Displaying Fonts: Required Arguments, Options

Required Arguments for Displaying Fonts

NAME=*font-name* | *hardware-font-name*

N=*font-name* | *hardware-font-name*

specifies the font to be displayed. *Font-name* can be the name of a SAS software font or a font you previously created. Any hardware font that is available on your device and has a corresponding Chartype value may be used. The *hardware-font-name* must be enclosed in quotes.

See also: “Specifying Alternative Hardware Fonts” on page 80 , “Chartype window” on page 931, and Chapter 5, “SAS/GRAPH Fonts,” on page 75.

NOBUILD

NB

specifies that the GFONT procedure is to display an existing font. The NOBUILD argument tells the procedure that no font is being generated and not to look for an input data set.

Featured in: Example 1 on page 962.

To display a user-generated font, you must define libref GFONT0. See “About the Libref GFONT0” on page 941.

Options for Displaying Fonts

Options that can be used for either font display or font creation are described here and in “Options for Creating Fonts” on page 947.

Options that display a font can be used when you create a font if you also display it (that is, the NODISPLAY option is not used in the PROC GFONT statement). However, none of the display options affect the design and appearance of the stored font except the NOKEYMAP, SHOWROMAN, and ROMHEX options.

When the syntax of an option includes *units*, use one of these:

CELLS	character cells
CM	centimeters
IN	inches
PCT	percentage of the graphics output area
PT	points

If you omit *units*, a unit specification is searched for in this order:

- 1 the value of GUNIT= in a GOPTIONS statement
- 2 the default unit, CELLS.

CTEXT=*text-color*

CT=*text-color*

specifies a color for the body of the characters. If you do not use the CTEXT= option, a color specification is searched for in the following order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the default, the first color in the colors list.

The CTEXT= value is not stored as part of the font.

Featured in: Example 2 on page 964.

GOUT=<libref.>output-catalog

specifies the SAS catalog in which to save the graphics output produced by the display of the font. The GOUT option is ignored if you use the NODISPLAY option in the PROC GFONT statement. You can use the GREPLAY procedure to view the output that is stored in the catalog. If you omit the libref, SAS/GRAPH looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist.

See also: “Storing Graphics Output in SAS Catalogs” on page 53.

HEIGHT=*character-height*<units>

H=character-height<units>

specifies the height of the font characters in number of units, *n*. Height is measured from the minimum font measurement to the capline. By default, HEIGHT=2.

Featured in: Example 1 on page 962.

NOKEYMAP

specifies that the current key map is ignored when displaying the font and its character codes or hexadecimal values. If you do not use the NOKEYMAP option when you display a font, the current key map remains in effect. If any characters in the font are not available through the current key map, they are not displayed and a warning is issued in the SAS log. This happens when the key map is asymmetrical, that is, not all characters in the font are mapped into the current key map.

Displaying a font using the NOKEYMAP option enables you to see all of the characters in the font, including those that are not mapped into your current key map. Note that only those characters that are mapped into your current key map are available (that is, those that are displayed when you display the font without the NOKEYMAP option).

See also: Chapter 5, “SAS/GRAPH Fonts,” on page 75, Chapter 34, “The GKEYMAP Procedure,” on page 983, and the NOKEYMAP option on page 949 for Creating Fonts.

NOROMAN**NR**

turns off the automatic display of character codes that are produced when you use the SHOWROMAN option during font creation.

NOROMHEX**NOHEX**

turns off the automatic display of hexadecimal values that are produced when you use the ROMHEX option during font creation.

REFCOL=reference-line-color

specifies a color for reference lines. By default, the first color in the colors list is used.

REFLINES

draws reference lines around each displayed character. Vertical reference lines show the width of the character. Horizontal reference lines show the font maximum and the font minimum, as well as the baseline and the capline. See Figure 32.1 on page 940 for an illustration of the placement of reference lines.

ROMCOL=code-color**RC=code-color**

specifies the color of the character codes or hexadecimal values that are displayed with the SHOWROMAN and ROMHEX options. If you do not use the ROMCOL= option, a color specification is searched for in the following order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the default, the first color in the colors list.

The ROMCOL= value is not stored as part of the font.

Featured in: Example 1 on page 962.

ROMFONT=font**RF=font**

specifies the font for character codes and hexadecimal values that are displayed by the SHOWROMAN and ROMHEX options. If you do not use the ROMFONT= option, a font specification is searched for in the following order:

- 1 the FTEXT= option in a GOPTIONS statement
- 2 the default hardware font, NONE.

Featured in: Example 1 on page 962.

ROMHEX

HEX

displays hexadecimal values below the font characters. If you use both the ROMHEX and SHOWROMAN options, both the character codes and the hexadecimal values are displayed. You also can use the ROMHEX option when you create a font.

See also: the ROMHEX option on page 950.

ROMHT=*height*<*units*>

RH=*height*<*units*>

specifies the height of the character codes and the hexadecimal values that are displayed with the SHOWROMAN and ROMHEX options in number of units, *n*. If you do not use the ROMHT= option, a height specification is searched for in the following order:

- 1 the HTEXT= option in a GOPTIONS statement
- 2 the default, ROMHT=1.

Featured in: Example 1 on page 962.

SHOWALL

displays the font with a space for every possible character position whether or not a font character exists for that position. The characters that are displayed are those available under your current key map, unless you use the NOKEYMAP option. The SHOWALL option usually is used in conjunction with the ROMHEX option, in which case all possible hexadecimal values are displayed. If, under your current key map, a font character is available for a position, it displays above the hexadecimal value. If no character is available for a position, the space above the hexadecimal value is blank. You can use the SHOWALL option to show where undefined character positions fall in the font.

SHOWROMAN

SR

displays character codes below the font characters even if they are not displayed automatically with the font. If you use both the SHOWROMAN and ROMHEX options, both the character codes and the hexadecimal values are displayed. You can also use the SHOWROMAN option when you create a font.

See also: “About Creating Fonts” on page 940

Featured in: Example 1 on page 962.

Details

To display a font, you must specify the name of the font with the NAME= argument and include the NOBUILD argument. For example, to display the Weather font with character codes that are displayed in the Swiss font, use the following statement:

```
proc gfont name=weather nobuild romfont=swiss;
```

Creating Fonts: Required Arguments, Options

Required Arguments for Creating Fonts

NAME=*font-name*

N=*font-name*

assigns a name to the font that you create. *Font-name* is the name of a catalog entry and must be a valid SAS name of no more than eight characters. Do not use the name of an Institute-supplied font or NONE for the name of a font.

Featured in: Example 2 on page 964.

DATA=*font-data-set*

specifies the SAS data set that the GFONT procedure uses to build the font. The data set must be sorted by the variables CHAR and SEGMENT. By default, the procedure uses the most recently created data set as the font data set.

See also: “SAS Data Sets” on page 29.

Featured in: Example 2 on page 964.

When you create a font, you must define the libref GFONT0. See “About the Libref GFONT0” on page 941 for details.

Note: If a user-generated font has the same name as an Institute-supplied font and if the libref GFONT0 has been defined, the user-generated font is used because GFONT0 is searched first. △

Options for Creating Fonts

Options that can be used for either font display or font creation are described here and in “Options for Displaying Fonts” on page 944.

Options that display a font can be used when you create a font if you also display it (that is, the NODISPLAY option is not used in the PROC GFONT statement). However, none of the display options affect the design and appearance of the stored font except the NOKEYMAP, SHOWROMAN, and ROMHEX options.

When the syntax of an option includes *units*, use one of these:

CELLS	character cells
CM	centimeters
IN	inches
PCT	percentage of the graphics output area
PT	points

If you omit *units*, a unit specification is searched for in this order:

- 1 the value of GUNIT= in a GOPTIONS statement
- 2 the default unit, CELLS.

BASELINE=*y*

B=*y*

specifies the vertical coordinate in the font data set that is the baseline of the characters. The baseline is the line upon which the letters rest. If you do not use the BASELINE= option, the GFONT procedure uses the lowest vertical coordinate of the first character in the font data set.

CAPLINE=*y*

C=*y*

specifies the vertical coordinate in the font data set that is the capline of the characters. The capline is the highest point of normal Roman capitals. If you do not use the CAPLINE= option, the GFONT procedure uses the highest vertical coordinate in the font data set, in which case the capline and the font maximum are the same. See Figure 32.1 on page 940 for an illustration of capline and font maximum.

If you use the CAPLINE= option, then when the GFONT procedure calculates the height of a character, any parts of the character that project above the capline are ignored in the calculation.

You can use this option to prevent an accented capital like A from being shortened to accommodate the accent. For example, if you do not use the CAPLINE= option, the capline and the font maximum are the same and the A is shortened to make room for the accent below the capline. However, if CAPLINE= is used, the top of the letter A is at the capline, and the accent is drawn above the capline and below the font maximum.

CHARSPACETYPE=DATA | FIXED | NONE | UNIFORM

CSP=DATA | FIXED | NONE | UNIFORM

specifies the type of intercharacter spacing. The following are valid values:

DATA

specifies that the first observation for each character sets the width of that character. When CHARSPACETYPE=DATA, the PTYPE variable is required, and the observation that specifies the width of the character must have a PTYPE value of W. See “The Font Data Set” on page 951 for details on the PTYPE variable.

Intercharacter spacing is included in the character’s width. For example, if the first observation for the letter A specifies a character width of 10 units and the A itself occupies only 8 units, the remaining 2 units serve as intercharacter spacing.

Note: The character can extend beyond the width that you specified in the first observation if desired. \triangle

FIXED

adds a fixed amount of space between characters based on the font size. The width of the individual character is determined by the data that generate the character.

NONE

specifies that no space is added between characters. The width of the individual character is determined by the data that generate the character. This type of spacing is useful for script fonts in which the characters should appear connected.

UNIFORM

specifies that the amount of space that is used for each character is uniform rather than proportional. This means that each character occupies the same amount of space. For example, in uniform spacing the letters m and i occupy the same amount of space, whereas in proportional spacing m occupies more space than i. In uniform spacing, the character is always centered in the space and a fixed space is added between characters.

When UNIFORM is specified, the amount of space that is used for each character is one of the following:

- by default, the width of the widest character in the font.
- the width specified by the MWIDTH= option. See the MIDWIDTH= option on page 949 for details.

Specifying CHARSPACETYPE=UNIFORM is the same as using the UNIFORM option.

Note: By default, CHARSPACETYPE=FIXED. \triangle

CODELEN=1 | 2

specifies the length in bytes of the CHAR variable. By default, CODELEN=1. To specify double-byte character sets for languages such as Chinese, Japanese, or Korean, use CODELEN=2. If you specify a double-byte character set, you cannot specify kerning or space adjustment with the KERNDATA= or SPACEDATA= options.

FILLED**F**

specifies that the characters in a user-generated polygon font are filled.

Featured in: Example 2 on page 964.

KERNDATA=*kern-data-set***KERN=*kern-data-set***

specifies the SAS data set that contains kerning information. When the KERNDATA= option is used during font creation, the data that are contained in the kern data set are applied to the font and stored with it. You cannot specify kerning for a double-byte character set that is created by using the option CODELEN=2.

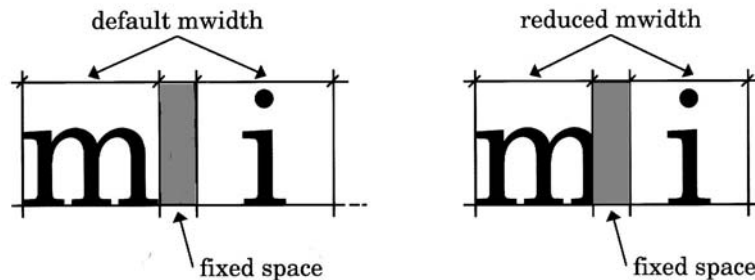
See also: “The Kern Data Set” on page 958.

MWIDTH=*character-width*

specifies the width of a character in a uniform font, where *character-width* is the number of font units. The MWIDTH= option is only valid when you specify uniform spacing by using the UNIFORM option or when you specify CHARSPACETYPE=UNIFORM. If you do not use MWIDTH=, the default is the width of the widest character in the font (usually the letter m).

Typically, you use the MWIDTH= option to tighten the spacing between characters. To do this, specify a smaller value (narrower width) for *character-width*. Figure 32.4 on page 949 shows the effect of decreasing the space that is allowed for uniformly spaced characters.

Figure 32.4 Using the MWIDTH= Option to Modify Spacing



See also: the CHARSPACETYPE= option on page 948 and the UNIFORM option on page 951.

NODISPLAY**ND**

specifies that the GFONT procedure is not to display the font that it is creating.

NOKEYMAP

specifies that the current key map is ignored when you generate and use the font that is being created, and that the character codes you enter are not mapped in any way before being displayed. As a result, the generated font is *never* affected by any setting of the KEYMAP= graphics option.

CAUTION:

Fonts generated with the NOKEYMAP option are never affected by any setting of the KEYMAP= graphics option. \triangle

By default, the NOKEYMAP option is *not* used; in which case, when you build a font, the current key map is applied to the values in the CHAR variable.

However, your current key map may not be symmetrical; that is, two or more input character codes may be mapped to the same output character. For example, if A is mapped to B, then both A and B map to B, but nothing maps to A. In this case, more than one code in your input data set can map to the same character in the resulting font. For example, if A and B are values of CHAR, both map to B. If this happens, a message that indicates the problem characters is displayed in the SAS log. To solve this problem, you can do one of the following:

- change the character code of one of the characters
- eliminate one of the characters
- use the NOKEYMAP option.

When you use the NOKEYMAP option, your font works correctly only if the end user's host or controller encoding is the same as the encoding used to create the input data set.

See also: the NOKEYMAP option on page 945 for Displaying Fonts and Chapter 34, "The GKEYMAP Procedure," on page 983.

RESOL=1...4

R=1...4

controls the resolution of the fonts by specifying the number of bytes (1 through 4) for storing coordinates in the font. The GFONT procedure provides three resolution levels (RESOL=3 produces the same resolution level as RESOL=4). By default, RESOL=1.

The higher the number, the closer together the points that define the character can be spaced. A high value specifies a denser set of points for each character so that the characters approximate smooth curved lines at very large sizes. RESOL=2 works well for most applications; RESOL=3 or 4 may be too dense to be practical.

The table below shows the resolution number and the maximum number of distinct points that can be defined horizontally or vertically.

Resolution	Number of Distinct Points
2	32,766
3	2,147,483,646
4	2,147,483,646

Featured in: Example 2 on page 964.

ROMHEX

HEX

specifies that hexadecimal values display automatically below the font characters when the GFONT procedure displays the font. If you use the ROMHEX option for a font that you create, you can later use the NOROMHEX option to suppress display of the hexadecimal values.

See also: the SHOWROMAN option on page 950, the ROMHEX option on page 946 for Displaying Fonts, and the NOROMHEX option on page 945.

SHOWROMAN

SR

specifies that character codes display automatically below the font characters when the GFONT procedure displays the font. If you use the SHOWROMAN option for a

font you create, you can later use the NOROMAN option to suppress display of the character codes.

See also: the ROMHEX option on page 946, the SHOWROMAN option for Displaying Fonts, and the NOROMAN option on page 945.

SPACEDATA=space-data-set

SPACE=space-data-set

specifies the SAS data set that contains font spacing information. When you use the SPACEDATA= option during font creation, the data contained in the space data set are applied to the font and stored with it. You cannot specify space adjustment for a double-byte character set that is created by using the option CODELEN=2.

See also: “The Space Data Set” on page 960.

UNIFORM

U

specifies that characters are spaced uniformly rather than proportionately. Using the UNIFORM option is the same as specifying CHARSPACETYPE=UNIFORM.

See also: the CHARSPACETYPE= option on page 948 and the MWIDTH= option on page 949.

Creating a Font

To create a font, you must create a data set that contains font information. Typically, you use a DATA step to create a SAS data set from which the GFONT procedure generates the font. The data set is referred to as the *font data set* and you can specify it with the DATA= argument.

To produce the font, invoke the GFONT procedure and specify the data set that contains the font information. In addition you can include options to modify the design and appearance of the font. For example, the following statement uses the data set FONTDATA to generate the font MYLOGO:

```
proc gfont data=fontdata name=mylogo;
```

For a demonstration of the font creation process, see Example 2 on page 964.

The GFONT procedure uses three types of data sets: the font data set, the kern data set, and the space data set. Each type of data set must contain certain variables and meet certain requirements. The following sections explain what each data set contains, how it is built, and what the requirements of the variables are.

The Font Data Set

The font data set consists of a series of observations that include the horizontal and vertical coordinate values and line segment numbers that the GFONT procedure uses to generate each character. In addition, each observation must include a character code that is associated with the font character and is used to specify the font character in a text string. The font data set also determines whether the font is stroked or polygon. A font data set that generates a polygon font produces an outline font by default. You can use the FILLED option with the same data set to generate a filled font.

The variables in the font data set must be assigned certain names and types. The table below summarizes the characteristics of the variables which are described further in “Font Data Set Variables” on page 952.

Table 32.1 Font Data Set Variables

Variable	Description	Type	Length	Valid Values	With Stroked Fonts	With Polygon Fonts
CHAR	the character code associated with the font character	character	1 or 2	keyboard characters or hexadecimal values	required	required
LP	the type of line segment being drawn, either a line or a polygon	character	1	L or P	optional	required
PTYPE	the type of data in the observation	character	1	V or C or W	optional	optional
SEGMENT	the number of the line segment or polygon being drawn	numeric		number	required	required
X	the horizontal coordinate	numeric		number	required	required
Y	the vertical coordinate	numeric		number	required	required

Font Data Set Variables

CHAR

provides a code for the character or figure that you are creating. CHAR is a character variable with a length of 1 or 2 and is required for all fonts.

CAUTION:

Using reserved or undefined hexadecimal codes as CHAR values may require the use of the NOKEYMAP option. △

The CHAR variable takes any character as its value, including characters that you can enter from your keyboard and hexadecimal values from '00'x to 'FF'x. (If you use hexadecimal values as CHAR values, your font may not work correctly under a key map that is different from the one under which the font was created because positions that are not defined in one key map may be defined in another.)

When you specify the code character in a text string, the associated font character is drawn. For example, if you create a Roman alphabet font, typically the characters you specify for CHAR are keyboard characters that match the character in the font. All of the observations that build the letter A have a CHAR value of A. When you specify 'A' in a text string this produces A in the output.

However, if you build a symbol font, the symbols may not have corresponding keyboard characters. In that case, you select a character or hexadecimal value to

represent each symbol in the font and assign it to CHAR. For example, in the Special font, the letter G is assigned as the code for the fleur-de-lis symbol. When you specify the code in a text string, the associated symbol displays.

If the CODELEN= option is set to 2, the values for CHAR represent two characters, such as AA, or a four-digit hexadecimal value, such as '00A5'x.

LP

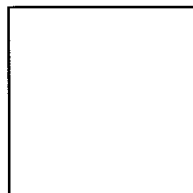
tells the GFONT procedure whether the coordinates of each segment form a line or a polygon. LP is a character variable with a length of 1. It is required for polygon fonts but optional for stroked fonts. You can assign the LP variable either of the following values:

- L lines
- P polygons.

Every group of line segments with an LP value of P is designated as a polygon; if the observations do not draw a completely closed figure, the program closes the figure automatically. For example, the following observations do not contain an LP variable. They produce a shape like the one in Figure 32.5 on page 953.

OBS	CHAR	SEG	X	Y
1	b	1	1	1
2	b	1	1	3
3	b	1	3	3
4	b	1	3	1

Figure 32.5 Using a LP Value of Line

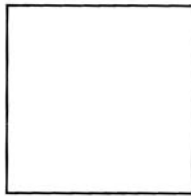


LP (continued)

An LP variable with a value of P for all observations added to the data set produces a complete box like the one in Figure 32.6 on page 954.

OBS	CHAR	SEG	X	Y	LP
1	b	1	1	1	P
2	b	1	1	3	P
3	b	1	3	3	P
4	b	1	3	1	P

Figure 32.6 Using a LP Value of Polygon

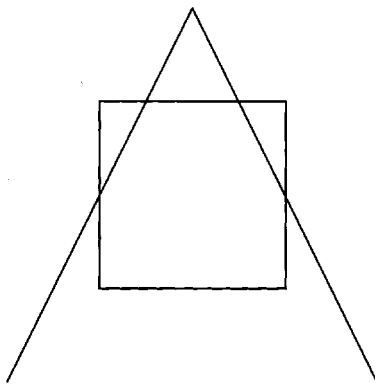


LP (continued)

The LP variable allows you to mix lines and polygons when you create characters in a font. For example, the following observations produce the single figure that is composed of a polygon and a line segment, as shown in Figure 32.7 on page 954:

OBS	CHAR	SEG	X	Y	LP
1	b	1	1	1	P
2	b	1	1	3	P
3	b	1	3	3	P
4	b	1	3	1	P
5	b	2	0	0	L
6	b	2	2	4	L
7	b	2	4	0	L

Figure 32.7 Mixing LP Values of Line and Polygon



PTYPE

tells the GFONT procedure what type of data are in the observation. PTYPE is a character variable of length 1 that is optional for both stroked and polygon fonts. For each observation, the PTYPE variable assigns a characteristic to the point that is determined by the X and Y values. You can assign the PTYPE variable any of the following values:

- V normal point in the line segment
- C center of a circular arc joining two V points
- W width value for CHARSPACETYPE=DATA.

If the GFONT procedure encounters the sequence V-C-V in consecutive observations, it draws an arc that connects the two V points and has its center at the C point. If a circle cannot be centered at C and pass through both V points, the results are unpredictable. Arcs are limited to 106 degrees or less.

If you specify an observation with a PTYPE value of W, it must always be the first observation for a character. Instead of providing digitizing data to the procedure, the observation gives the minimum and maximum X values for the character. Note that in this case, the Y variable observation actually contains the maximum X value. Usually, these values include a little extra space for intercharacter spacing. Use a PTYPE of W only if you have specified CHARSPACETYPE=DATA; otherwise, the points are ignored. For more information on intercharacter spacing, see the description of the CHARSPACETYPE= option.

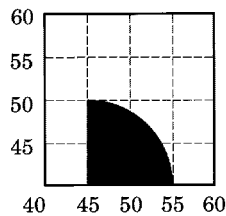
If you do not specify a PTYPE variable in the font data set, all points are assumed to be V-type points.

The following observations illustrate how the PTYPE variable is used to draw an arc similar to Figure 32.8 on page 956. (After the figure was generated, a grid was overlaid on it to show the location of the points.) A comment following each observation explains its function.

OBS	CHAR	SEG	X	Y	LP	PTYPE	Comment
1	a	1	40	60	P	W	define width of character as 20 font units, which is the number of units from left margin, 40, to right margin, 60
2	a	1	45	40	P	V	start line segment at position 45,40
3	a	1	45	50	P	V	draw a line to position 45,50, which is start point of arc

OBS	CHAR	SEG	X	Y	LP	PTYPE	Comment
4	a	1	45	40	P	C	draw an arc whose center is at 45,40
5	a	1	55	40	P	V	finish drawing the arc at 55,40

Figure 32.8 Using the PTYPE Variable to Create an Arc



PTYPE (continued)

Note the following:

- Three observations are required to draw the arc: observation 3 and observation 5 denote the start point and endpoint of the arc, respectively, and observation 4 locates the center of the arc.
- The figure is closed because the line segments have an LP value of P (polygon).
- The font that contains the figure of the arc was generated with a PROC GFONT statement like the following:

```
proc gfont data=arc name=arcfig charspacetype=data filled ;
```

Note that the GFONT procedure uses the CHARSPACETYPE= option with a value of DATA to specify that the first observation sets the width of the character. The FILLED option fills the area of the arc.

SEGMENT

numbers the line segments that compose a character or symbol. SEGMENT is a numeric variable that is required for both polygon and stroked fonts. All the observations for a given line segment have the same segment number. The segment number changes when a new line segment starts.

When the GFONT procedure draws a stroked character with more than one line segment (for example, the letter E), or a polygon character with a hole (for example, the letter A), it needs to know when one line stops and where the next line begins. There are two ways to do this, as follows:

- 1 Change the segment number when a new line segment starts. If the value of LP is L (line), a change in segment numbers tells the GFONT procedure not to connect the last point in line segment 1 and the first point in line segment 2. If the value of LP is P (polygon), a change in segment numbers causes both of the following:

- The last point in line segment 1 is joined to the first point in line segment 1, thus closing the polygon.
- The program starts a new polygon. If the value of CHAR has not changed, the new polygon is part of the same character.

Use this method for characters that are composed of two polygons, such as a question mark (?). If you draw a polygon with a hole in it, such as the letter A, use the second method.

- 2 Keep the same segment number for all lines, but insert an observation with missing values for X and Y between the observation that marks the end of the first line and the observation that begins the next line. For example, if you are drawing the letter O, insert an observation with a missing value between the line that draws the outer circle and the beginning of the line that draws the inner circle.

The first method is preferred, unless you are creating a polygon character with a hole in it. In this case, you should separate the lines with a missing value and keep the same segment numbers. (Note that if you use separate line segments when you create a polygon with a hole, the results may be unpredictable.) For example, observations such as the following from a data set called BOXES were used to draw the hollow square in Figure 32.9 on page 958. The data points that form the figure are laid out on a grid shown next to the square.

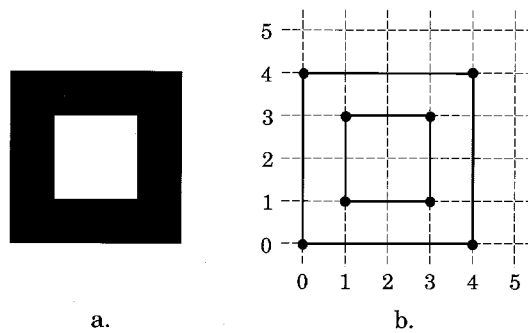
OBS	CHAR	SEG	X	Y	LP
1	b	1	1	1	P
2	b	1	1	3	P
3	b	1	3	3	P
4	b	1	3	1	P
5	b	1	-	-	P
6	b	1	0	0	P
7	b	1	0	4	P
8	b	1	4	4	P
9	b	1	4	0	P

Note that observation 5, which has missing values for X and Y, separates the observations that draw the inner box from those that draw the outer box and that the segment number is the same for all the observations. Figure 32.9 on page 958 was generated with a GFONT statement like the following:

```
proc gfont data=boxes name=boxes filled;
```

Note that the FILLED option is included and that only the space between the two squares is filled.

Figure 32.9 Drawing Nested Polygons



X and Y

specify the horizontal and vertical coordinates of the points for each character. These variables must be numeric, and they must be named X and Y for the horizontal and vertical coordinates, respectively. Their values describe the position of the points on the character. These values can be in any range that you choose, but both variables must describe the character in the same scale or font units. In other words, 10 horizontal units must be the same distance as 10 vertical units. You should define vertical coordinates for all characters on the same baseline.

Note: When you specify PTYPE=W, both X and Y contain horizontal coordinate values. Δ

Creating a Font Data Set

You can create a font data set by digitizing the shape of the characters or figures either manually or with special digitizing equipment. To create a font data set by digitizing the characters manually, follow these steps:

- 1 Determine the coordinate points for each line segment by drawing the characters on a grid.
- 2 Lay out the observations for each character. Each observation describes a move from one point to another along a line segment. For each line segment, enter the coordinate points in the order in which they are drawn. For a stroked font, when you start a new line segment, change the segment number. For a polygon font, when you start a new polygon, change the line segment number.

If the polygon has a hole in it, as in the letter O, keep the line segment number and separate the lines with a missing value. Use the same value for CHAR for all of the observations that describe one character.

- 3 Create a SAS data set that contains the variables CHAR, SEGMENT, X, and Y, and read in the data for each observation. Include the variables LP and PTYPE if necessary.
- 4 Sort the data set by CHAR and SEGMENT.
- 5 Assign the font data set with the DATA= argument.

This process is illustrated in Example 2 on page 964.

The Kern Data Set

The kern data set consists of observations that specify how much space to add or remove between any two characters when they appear in combination. This process,

called *kerning*, increases or decreases space between the characters. Kerning usually is applied to certain pairs of characters that, because of their shape, have too much space between them. Reducing the space between characters may allow part of one character to extend over the body of the next. Examples of some combinations that should be kerned are AT, AV, AW, TA, VA, and WA.

You can apply kerning to the intercharacter spacing that you specify with the CHARSPACETYPE= option (except for uniform fonts). You can refine the kerning of your characters as little or as much as you like. You assign the kern data set with the KERNDATA= option.

Kern Data Set Variables

The kern data set must contain these variables:

CHAR1

specifies the first character in the pair to be kerned. CHAR1 is a character variable with a length of 1.

CHAR2

specifies the second character in the pair to be kerned. CHAR2 is a character variable with a length of 1.

XADJ

specifies the amount of space to add or remove between the two characters. XADJ is a numeric variable that uses the same font units as the font data set. The value of XADJ specifies the horizontal adjustment to be applied to CHAR2 whenever CHAR1 is followed immediately by CHAR2. Negative numbers decrease the spacing, and positive numbers increase the spacing.

Creating a Kern Data Set

Each observation in a kern data set names the pair of characters to be kerned and the amount of space to be added or deleted between them. To create a kern data set, follow these steps:

- 1 Select the pairs of characters to be kerned, and specify the space adjustment (in font units) for each pair as a positive number (more space) or negative number (less space).
- 2 Create a SAS data set that contains the variables CHAR1, CHAR2, and XADJ; produce one observation for each pair of characters and the corresponding space adjustment.

```
data kern1;
  input char1 $ char2 $ xadj;
  datalines;
A T -4
D A -3
T A -4
;
```

- 3 Assign the kern data set with the KERNDATA= option.

```
proc gfont data=fontdata
  name=font2
  charspacetype=data
  kerndata=kern1
  nodisplay;

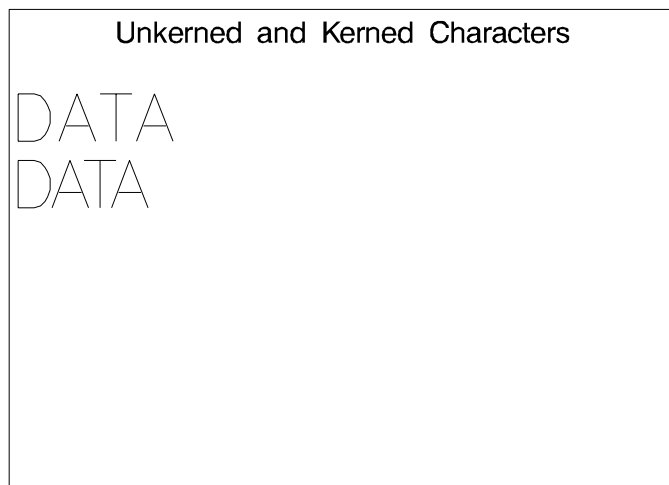
run;
```

Figure 32.10 on page 960 illustrates how you can use the `KERNDATA=` option to create a font in which the space between specified pairs of letters is reduced. The characters A, D, and T are shown as the word `DATA`. The first line uses the unkernd font, `FONT1`, and the second line uses the kerned font, `FONT2`. Note that the characters in `FONT2` are spaced more closely than the characters in `FONT1`.

The following title statements specify the kerned and unkernd fonts and are used with the `GSLIDE` procedure to produce Figure 32.10 on page 960:

```
title2 lspace=6 f=font1 h=10 j=1 'DATA';
title3 lspace=4 f=font2 h=10 j=1 'DATA';
```

Figure 32.10 Comparison of Kerned and Unkernd Text



The Space Data Set

As the height (point size) of a font increases, less space is required between letters in relation to their height. If the point size decreases, more space may be needed. The space data set tells the `GFONT` procedure how much to increase or decrease the intercharacter spacing for a given point size. Like kerning, spacing is added to or subtracted from the intercharacter spacing that is specified by the `CHARSPACETYPE=` option. However, kerning applies the adjustment to specified pairs of characters, while spacing is applied uniformly to all characters.

Values that are specified in the space data set are added to the normal intercharacter spacing and any kerning data. Normal intercharacter spacing is determined by the `CHARSPACETYPE=` option.

Space Data Set Variables

The space data set must contain these variables:

SIZE

specifies the point size of the font. `SIZE` is a numeric variable.

ADJ

specifies the spacing adjustment for the point size in hundredths (1/100) of a point. (A point is equal to 1/72 of an inch.) `ADJ` is a numeric variable. Positive values for

the ADJ variable increase the spacing between characters; negative values reduce the space.

Creating a Space Data Set

Each observation in a space data set specifies a point size (SIZE) and the amount of space (ADJ) to be added or subtracted between characters when a font of that point size is requested. When you specify a point size that is not in the space data set, the adjustment for the next smaller size is used. To create a space data set, follow these steps:

- 1 Determine the amount of adjustment that is required for typical point sizes; positive numbers increase spacing, and negative numbers decrease spacing.
- 2 Create a SAS data set that contains the variables SIZE and ADJ; produce one observation for each point size and corresponding space adjustment.

```
data spacel;
  input size adj;
  datalines;
  6 40
  12 0
  18 -40
  24 -90
  30 -150
  36 -300
  42 -620
  ;
```

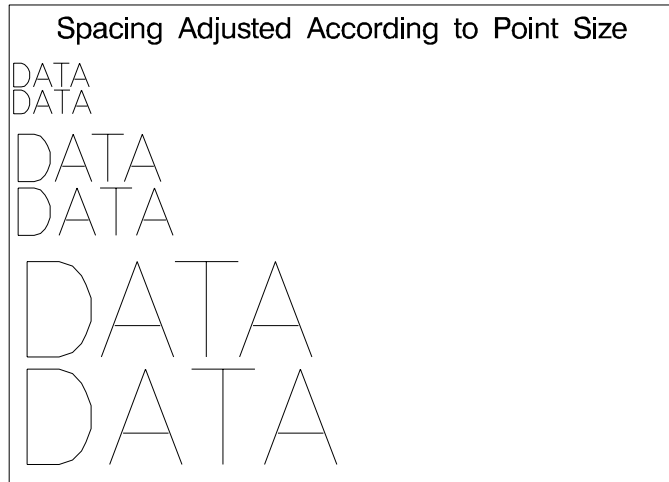
- 3 Assign the space data set with the SPACEDATA= option.

```
proc gfont data=fontdata
  name=font3
  charspacetype=data
  spacedata=spacel
  nodisplay;
run;
```

Figure 32.11 on page 962 illustrates how to use the SPACEDATA= option to generate a font in which intercharacter spacing is adjusted according to the height of the characters. The characters A, D, and T are shown as the word DATA. Each pair of lines displays the word DATA and at the same size uses first the font with spacing adjustment (FONT3) and then the original font (FONT1). Note that as the size of the characters increases, the space between them decreases.

The following title statements are used with the GSLIDE procedure to produce Figure 32.11 on page 962:

```
title2;
title3 f=font3 h=.25in j=1 'DATA'; /* 18 points */
title4 f=font1 h=.25in j=1 'DATA';
title5;
title6 f=font3 h=.50in j=1 'DATA'; /* 36 points */
title7 f=font1 h=.50in j=1 'DATA';
title8;
title9 f=font3 h=1.0in j=1 'DATA'; /* 72 points */
title10 f=font1 h=1.0in j=1 'DATA';
```

Figure 32.11 Comparison of Text with and without Spacing Adjustments

Examples

The following examples illustrate major features of the GFONT procedure.

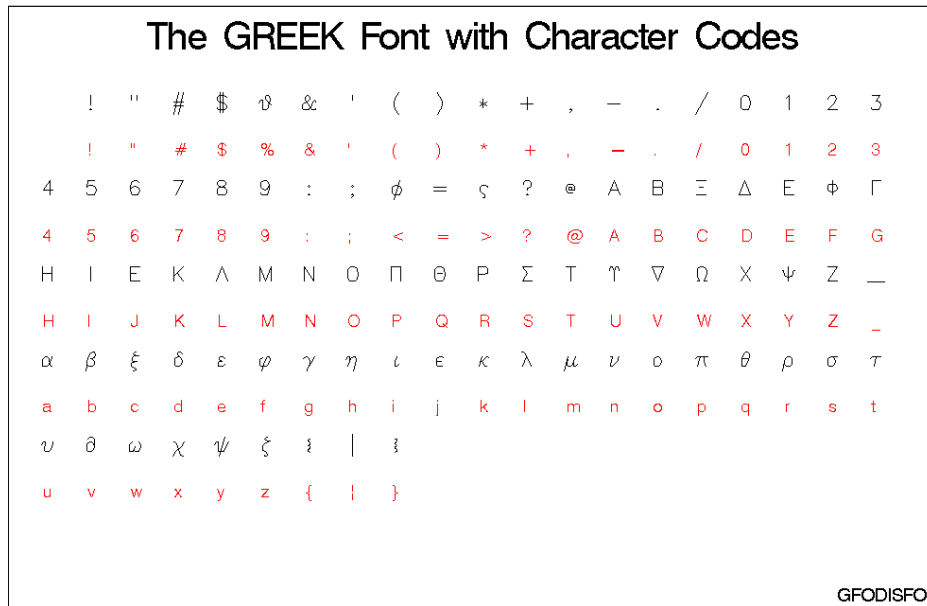
Example 1: Displaying Fonts and Character Codes

Procedure features:

GFONT statement options:

HEIGHT=
 NOBUILD
 ROMCOL=
 ROMFONT=
 ROMHT=
 SHOWROMAN

Sample library member: GFODISFO

Figure 32.12 Display of the Greek Font with Character Codes (GFODISFO)

This example illustrates the `SHOWROMAN` option, which displays the character codes that are associated with the font characters that are being displayed. A display such as this one shows which keyboard character you enter to produce the Greek character you want. In addition, this example shows how to modify the appearance of both the font characters and the character codes when they are displayed.

Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss htitle=6 htext=3;
```

Define title and footnote.

```
title 'The GREEK Font with Character Codes';
footnote j=r 'GFODISFO ';
```

Display the GREEK font with character codes. `NOBUILD` indicates that the font specified in the `NAME=` argument is an existing font. `HEIGHT=` specifies the height of the Greek characters. `ROMCOL=`, `ROMFONT=`, and `ROMHT=` assign the color, type style, and height of the character codes. `SHOWROMAN` displays the character codes.

```
proc gfont name=greek
    nobuild
    height=3.7
    romcol=red
    romfont=swiss1
    romht=2.7
    showroman;
run;
```

```
quit;
```

Example 2: Creating Figures for a Symbol Font

Procedure features:

GFONT statement options:

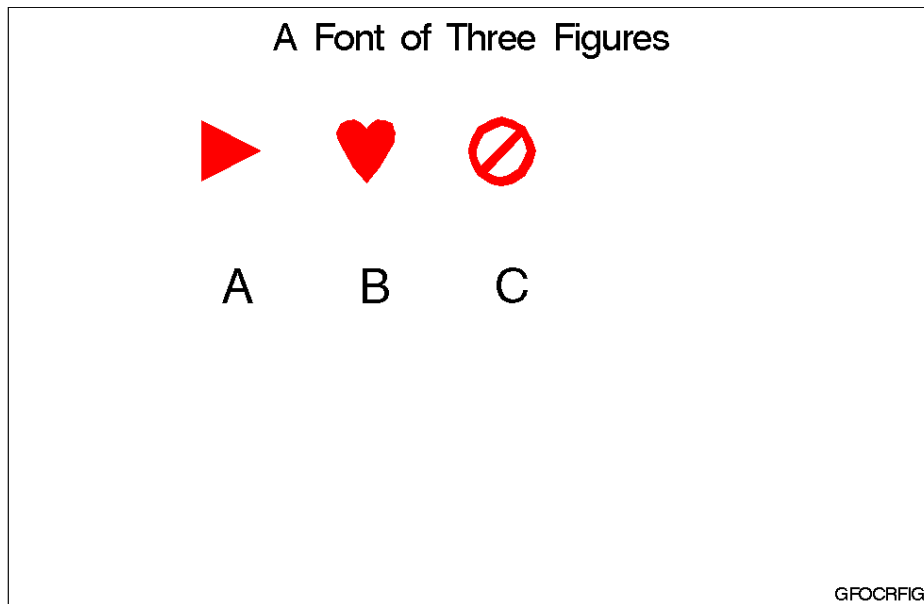
```
CTEXT=
DATA=
FILLED
NAME=
RESOL=
```

Other features:

LIBNAME statement

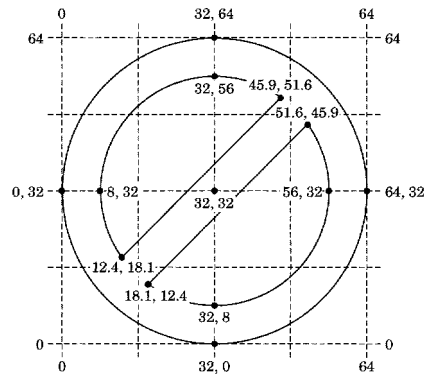
Sample library member: GFOCRFIG

Figure 32.13 Display of Symbols For Characters



This example shows how to create three simple figures for a symbol font. Each figure is laid out on a grid that is 64 font units square. The third figure is a circle with a slash through it. Figure 32.14 on page 965 shows the figure and some of its coordinate points laid out on a grid.

Figure 32.14 Diagram of Circle with Slash Figure



Assign the librefs and set the graphics environment. The LIBNAME statement associates the libref GFONT0 with the SAS data library in which the font catalog is stored.

```
libname gfont0 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ftext=swiss htitle=6 htext=3;
```

Create the font data set FIGURES for a triangle, a heart, and a circle with slash. The first figure, a right-pointing triangle that is assigned the character code A, is a polygon drawn with three straight lines.

```
data figures;
  input char $ ptype $ x y segment lp $;
  datalines;
A   W   0   64   0   P /* triangle pointing right */
A   V   4   4   1   P
A   V  60  32   1   P
A   V   4  60   1   P
A   V   4   4   1   P
```

The second figure, a heart that is assigned the character code B, uses the PTYPE variable combination V-C-V to draw the arcs that make up the top of the heart. Each side requires two arcs. Since the arcs are continuous, the observation that marks the end of one arc is also the beginning of the next arc. The heart drawing begins at the bottom point and continues counterclockwise.

```
B   W   0   64   0   P /* heart */
B   V  32   2   1   P
B   V  44  17   1   P
B   V  58  40   1   P
B   C  46  47   1   P
B   V  56  58   1   P
B   C  46  47   1   P
```

```

B   V   32   52   1   P
B   C   18   47   1   P
B   V    8   58   1   P
B   C   18   47   1   P
B   V    6   40   1   P
B   V   20   17   1   P
B   V   32    2   1   P

```

The third figure, a circle with a slash through it that is assigned the character code C, is composed of three polygons: a circle and two empty arcs. An observation with missing values separates the observations defining each of the three polygons. The outer circle is defined by the first group of observations. The empty arcs are drawn with three continuous arcs using the PTYPE variable pattern V-C-V-C-V-C-V. The straight line that closes the arc is drawn automatically by the GFONT procedure in order to complete the polygon. Because all the polygons are part of one character, the continuous space they define is filled.

```

C   W    0   64   0   P /* circle with slash */
C   V   32   64   1   P
C   C   32   32   1   P
C   V   64   32   1   P
C   C   32   32   1   P
C   V   32    0   1   P
C   C   32   32   1   P
C   V    0   32   1   P
C   C   32   32   1   P
C   V   32   64   1   P
C   V    .    .   1   P
C   V  12.4  18.1  1   P
C   C   32   32   1   P
C   V    8   32   1   P
C   C   32   32   1   P
C   V   32   56   1   P
C   C   32   32   1   P
C   V  45.9  51.6  1   P
C   V    .    .   1   P
C   V  51.6  45.9  1   P
C   C   32   32   1   P
C   V   56   32   1   P
C   C   32   32   1   P
C   V   32    8   1   P
C   C   32   32   1   P
C   V  18.1  12.4  1   P
;

```

Define the title and footnote.

```

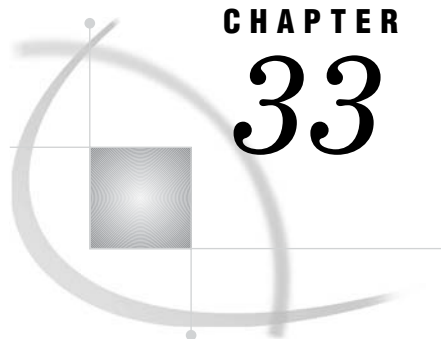
title 'A Font of Three Figures';
footnote j=r 'GFOCRFIG ';

```

Generate and display the font FIGURES. The DATA= argument names the input data set that is used to generate the font. The NAME= argument names the font that the procedure generates and automatically stores it in the GFONT0 catalog. (Note that you do not need to specify GFONT0.) FILLED specifies a filled polygon font. CTEXT= specifies the color of the figures in the font display. The color specification is not stored with the font. RESOL= is set to 2 to improve the resolution of the lines. By default, the newly generated font is displayed (the NODISPLAY option is not used).

```
proc gfont data=figures
    name=figures
    filled
    height=.75in
    ctext=red
    showroman
    romht=.5in
    resol=2;

run;
quit;
```

CHAPTER 33

The GIMPORT Procedure

<i>Overview</i>	969
<i>Concepts</i>	970
<i>About Importing Graphics</i>	970
<i>Specifying a Fileref</i>	970
<i>Importing the File</i>	970
<i>CGM Elements Not Supported</i>	971
<i>About Color Mapping</i>	971
<i>About Pattern Mapping</i>	971
<i>About Font Mapping</i>	972
<i>Procedure Syntax</i>	972
<i>PROC GIMPORT Statement</i>	973
<i>MAP Statement</i>	974
<i>SCALE Statement</i>	975
<i>TRANSLATE Statement</i>	976
<i>Examples</i>	976
<i>Example 1: Creating and Importing a CGM</i>	977
<i>Example 2: Adjusting the Graphics Output</i>	979
<i>References</i>	981

Overview

The GIMPORT procedure enables you to import into SAS/GRAPH software graphics output that is produced with other software applications, graphics output that is produced by SAS/GRAPH software, or graphics output that is produced on other machines. The GIMPORT procedure takes as its input a computer graphics metafile (CGM) and produces graphics output that can be displayed in your SAS/GRAPH session and stored in a SAS catalog. This graphics output can be reviewed and played like any other SAS/GRAPH output using the GREPLAY procedure. The GIMPORT procedure may also write any of the following information to the log:

- any elements used in the CGM that the procedure cannot process
- color mapping information when a color in a CGM is not available on the destination device
- a list of fonts that are used by the application that produced the CGM.

Note: In addition to the GIMPORT procedure, you can use commands in the File pull-down menu in the Image Editor, Graph Editor, and Graph window to import other graphic formats such as GIF, TIFF, and WMF. △

Concepts

About Importing Graphics

A computer graphics metafile (CGM) is a graphics output file that is created according to a standard (ANSI X3.122). Since many graphics applications, including SAS/GRAPH software, can generate and import CGMs, these files can be read by different applications programs or used on different machines.

The GIMPORT procedure imports a CGM with which a fileref has been associated. Using the CGM as input, the procedure displays the graphics output and creates a catalog entry. The following sections address how to assign the fileref to the external file (CGM) and how to import the file.

Specifying a Fileref

You must assign a fileref to the external file that contains the CGM that you want to use as input so that the GIMPORT procedure can locate it. You can do this with a FILENAME statement that has the following form:

```
FILENAMEcgm-fileref'external-file';
```

Replace *cgm-fileref* with any fileref name that you want. Replace *'external-file'* with the complete file name of the CGM. You can omit the FILENAME statement if you have already defined the fileref. You can also specify a fileref with a host command in some operating environments. See “FILENAME Statement” on page 28 for additional information.

Importing the File

The PROC GIMPORT statement reads the input CGM and displays the graphics output. When the CGM is displayed using only the PROC GIMPORT statement, the resulting graphics output may not be sized or positioned correctly for the device on which it is displayed. In these cases, you can use the SCALE and TRANSLATE statements to adjust the size and location of the new graphics output.

In addition, if the CGM contains the FONT LIST element, the procedure lists in the log the fonts used in the CGM. You can change these fonts to SAS software fonts using the MAP statement. If you do not change these fonts to SAS software fonts, the GIMPORT procedure uses a default font.

Because it is easier to determine what adjustments the graphics output needs after it has been displayed, you may want to follow these steps:

- 1 Import the CGM and display the graphics output using only the PROC GIMPORT statement.
- 2 Decide what adjustments you want to make to the size and position of the graphics output.
- 3 If the procedure lists the fonts that are used by the CGM, decide what font substitutions you want to make.
- 4 Run the procedure again with the appropriate MAP, SCALE, or TRANSLATE statements.

Note: Once you have determined the correct values for the SCALE and TRANSLATE statements for the graphics output produced by a particular CGM, you

can use the same values for all other graphics output that is generated by the same software application. △

CGM Elements Not Supported

The GIMPORT procedure does not support certain CGM elements. If the input CGM contains any of the following elements, the GIMPORT procedure writes a message to the log noting that the procedure cannot process them:

- the CELL ARRAY primitive element (a bitmap CGM file)
- the CHARACTER SPACING attribute element
- the APPLICATION DATA element
- the ESCAPE element.

These elements are rarely used and their absence should not affect the graphics output produced by the GIMPORT procedure.

About Color Mapping

If the CGM specifies colors for the graphics elements that it generates, you may or may not be able to map them to the color that you want in your SAS/GRAPH output, depending on the way these colors are specified in the CGM.

You cannot change the color mapping if, in the CGM, the COLOUR SELECTION MODE element is set to DIRECT. In this case, the colors are explicitly defined by the CGM and you cannot change them. However, if the CGM was created with a SAS/GRAPH CGM device driver, you can control the colors by specifying the appropriate colors when you create the graphics output or by changing the colors in the CGM device entry and re-creating the CGM. See Chapter 31, “The GDEVICE Procedure,” on page 915 for details. In addition, you can use a color map with the GREPLAY procedure to remap the colors. In the color map, the FROM color must be specified in RGB format, but the TO color can be any valid color name. See Chapter 43, “The GREPLAY Procedure,” on page 1237 for details on color maps.

You can change the color mapping if the COLOUR SELECTION MODE element is set to INDEXED and there is no color table defined in the CGM file. In this case, you can map the colors from the CGM to the colors of your choice by using the COLORS= graphics option when you run the GIMPORT procedure. The CGM colors are mapped to match the order of the colors in the colors list. If the procedure cannot reproduce the colors specified in the CGM, the following message is written to the log:

```
WARNING: Invalid color index n encountered.
         It has been mapped to color-name.
```

Note: The color name from the CGM is converted to the RGB format for SAS/GRAPH color names; that is, WHITE is converted to CXFFFFFF, and so on. See Chapter 6, “SAS/GRAPH Colors and Images,” on page 91 for details. △

About Pattern Mapping

If the CGM contains pattern specifications, you may be able to map them to patterns of your choice using SAS/GRAPH PATTERN definitions.

If the CGM defines a PATTERN TABLE, then the patterns defined by this table are the patterns that are used and you cannot change them.

If a PATTERN TABLE is not defined in the CGM, under certain conditions you may be able to use SAS/GRAPH PATTERN definitions to control the patterns that are used. If INTERIOR STYLE is set to PATTERN and if a PATTERN TABLE INDEX has been specified, then the GIMPORT procedure uses the PATTERN TABLE INDEX to look up SAS/GRAPH PATTERN definitions. If patterns are defined, the procedure uses the first available pattern. For example, if the PATTERN TABLE INDEX *n* has been defined, the procedure uses SAS/GRAPH PATTERN definition *n*. If the SAS/GRAPH PATTERN definition is not the correct pattern type, the procedure modifies the pattern as necessary. If no PATTERN definitions are currently in effect, an INVALID PATTERN TABLE INDEX warning is issued and no pattern is used.

About Font Mapping

By default, the GIMPORT procedure maps all of the fonts in the CGM to the font that is specified by the FTEXT= graphics option. If the FTEXT= graphics option is not used, the default is the hardware font NONE. However, you may be able to specify a different font either by mapping the fonts or by using a graphics option.

When the CGM is imported, a numbered list of the fonts that are used in the CGM may be displayed in the LOG window. These are the fonts that were available to the application that originally generated the CGM. Depending on how the fonts are represented in the CGM, you may be able to map these fonts to fonts of your choice.

If the font and text in the imported graphics output are produced with move and draw commands that are included in the CGM, then no font name appears in the LOG window and the font cannot be mapped to a different one.

If the fonts used in the imported graphics output are represented in the CGM as a font name accompanied by a text string, they can be mapped to SAS/GRAPH fonts using the MAP statement. You can use the MAP statement if the message "WARNING: Invalid font index *n*. Font has been mapped to *font-name*" appears in the LOG window after the list of fonts. This means that font *n* in the list could not be reproduced and was mapped to the font specified in the FTEXT= graphics option or to the hardware font. You can map this font to a SAS/GRAPH software font of your choice using the MAP statement. See "MAP Statement" on page 974 for more information on mapping fonts.

You can also specify a font with the FTEXT= or CHARTYPE= graphics options if both of the following conditions are true:

- The font has not been mapped with a MAP statement.
- The CGM font contains a font name and text rather than the move and draw commands that draw the text in the specified font. In the latter case, the font name is not included in FONT LIST.

However, using a graphics option causes all fonts to be mapped to the one that is specified. See Chapter 5, "SAS/GRAPH Fonts," on page 75 for details of font specification and Example 2 on page 979.

Procedure Syntax

Supports: Output Delivery System (ODS)

```
PROC GIMPORT FILEREF=cgm-fileref | 'external-file'
  FILETYPE=CGM
  FORMAT=BINARY | CHARACTER | CLEARTEXT
  <GOUT=<libref.>output-catalog>;
```

```

MAP 'cgm-font' TO font ;
SCALE X=factor | Y=factor | X=factor Y=factor;
TRANSLATE X=offset | Y=offset | X=offset Y=offset;

```

PROC GIMPORT Statement

Identifies the input file to be processed, and specifies its file type and format. Optionally specifies an output catalog.

Syntax

```

PROC GIMPORT FILEREF=cgm-fileref | 'external-file'
  FILETYPE=CGM
  FORMAT=BINARY | CHARACTER | CLEARTEXT
  <GOUT=<libref.>output-catalog>;

```

Required Arguments

FILEREF=cgm-fileref | 'external-file'

specifies the computer graphics metafile (CGM) that is input for PROC GIMPORT. Following are the possible values for FILEREF=:

cgm-fileref

a fileref that is associated with the CGM and that has been previously defined using a FILENAME statement or host command.

'external-file'

the complete file name of the CGM that you want to import. See the operating system companion for your system for valid values for *external-file*.

Featured in: Example 2 on page 979.

FILETYPE=CGM

specifies the type of the input file, that is, the graphics standard to which the file conforms. CGM is the only valid value for the FILETYPE= argument. If the FILETYPE= argument is omitted, an error is issued and the procedure stops.

Featured in: Example 2 on page 979.

FORMAT=BINARY | CHARACTER | CLEARTEXT

specifies the format of the input file. CGMs can be encoded in one of the following three formats:

BINARY

specifies binary encoding. It is not printable.

CHARACTER

specifies an encoding suitable for transfer through networks that cannot support binary transfers. It is printable but not readable.

CLEARTEXT

specifies a text format that can be read using a standard text editor.

Most graphics packages use BINARY format. If you specify the wrong format, an "ERROR: Unable to interpret the CGM file" message is issued and the procedure stops. If this occurs, try a different format.

Featured in: Example 2 on page 979.

Options

GOUT=<libref.>output-catalog

specifies the SAS catalog in which to save the graphics output produced by the GIMPORT procedure. If you omit the libref, SAS/GRAPH looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist.

See also: “Storing Graphics Output in SAS Catalogs” on page 53.

MAP Statement

Substitutes a SAS/GRAPH software font for a font in the CGM.

Requirements: Submit a separate MAP statement for each CGM font that you want to map.

Tip: You can submit multiple MAP statements with the procedure.

Featured in: Example 2 on page 979.

Syntax

```
MAP 'cgm-font' TO font;
```

Required Arguments

'cgm-font'

identifies a font in the CGM. The name of the font must be enclosed in single quotation marks and written exactly as it appears in the font list; *cgm-font* is case sensitive. Do not include the font list number in *cgm-font*.

font

specifies the SAS/GRAPH font to which the CGM font is mapped. You can specify software fonts or hardware fonts for the destination device. You can also use fonts that are created by the GFONT procedure.

Note: Remember to specify the libref GFONT0 with a LIBNAME statement if *font* is a user-generated font. \triangle

By default, the GIMPORT procedure maps all of the CGM fonts to the font specified by the FTEXT= graphics option or, if the FTEXT= graphics option is not used, to the default hardware font, NONE.

Details

If the CGM includes the FONT LIST element, the GIMPORT procedure automatically lists the CGM font names in the log. Use this list to select the fonts for mapping. For example, suppose the font list includes the following entry:

3. Times Roman

If the LOG window displays the message "WARNING: Invalid font index *n*," you can map the Times Roman font to the SAS/GRAPH font CENTX with the following statement:

```
map 'Times Roman' to centx;
```

SCALE Statement

Enlarges or reduces the graphics output by increasing or decreasing the values of the *x* and *y* coordinates.

Requirements: You can submit only one SCALE statement.

Tip: You can submit the SCALE statement alone or in conjunction with the TRANSLATE statement, but the SCALE statement is always processed first.

Featured in: Example 2 on page 979.

Syntax

```
SCALE X=factor | Y=factor | X=factor Y=factor;
```

Required Arguments

At least one of the following arguments is required; both may be used and can be listed in either order:

X=*factor*

specifies the enlargement or reduction of the values of the *x* coordinates. *Factor* is the number by which these values are multiplied and cannot be less than or equal to 0. By default, X=1. Values less than 1 reduce the size of the graphics output while values greater than 1 increase the size of the graphics output. There is no limit on the size of *factor*.

Y=*factor*

specifies the enlargement or reduction of the values of the *y* coordinates. *Factor* is the number by which these values are multiplied and cannot be less than or equal to 0. By default, Y=1. Values less than 1 reduce the size of the graphics output while values greater than 1 increase the size of the graphics output. There is no limit on the size of *factor*.

Details

If the shapes in the imported graphics output are too narrow, you can make them wider by increasing the values of the *x* coordinate. To make the elements in the graphics output twice as wide, specify X=2. To make them half as high, specify Y=.5.

For example, if the values of the *x* coordinates range from 5 to 50 and if in the SCALE statement the factor for X= is specified as 2, then the values of all of the *x* coordinates are multiplied by 2 and the range of these values increases. The new range is 10 to 100. And if the values of the *y* coordinates range from 0 to 25 and if in the SCALE statement the factor for Y= is specified as .5, then the values of all of the *y* coordinates are multiplied by .5 and the range of these values decreases. The new range is 0 to 12.5.

If you specify a factor that causes the graphics output to exceed the size of the graphics output area, the procedure draws as much of the graphics output as will fit in the available space.

TRANSLATE Statement

Adjusts the location on the display of the graphics output imported by the procedure. Graphics output can be shifted left or right by offsetting the x values or shifted up or down by offsetting the y values.

Requirements: You can submit only one TRANSLATE statement.

Tip: You can submit the TRANSLATE statement alone or in conjunction with the SCALE statement but the SCALE statement is always processed first.

Featured in: Example 2 on page 979.

Syntax

```
TRANSLATE X=offset | Y=offset | X=offset Y=offset ;
```

Required Arguments

At least one of the following arguments is required; both may be used and can be listed in either order:

X=*offset*

specifies the number of units in percent of the display area to move the graphics output right (positive numbers) or left (negative numbers). The value of *offset* is added to the value of the x coordinate. By default, X=0.

Y=*offset*

specifies the number of units in percent of the display area to move the graphics output up (positive numbers) or down (negative numbers). The value of *offset* is added to the value of the y coordinate. By default, Y=0.

Details

The TRANSLATE statement adjusts the position of the graphics output without changing its size. The amount of the *offset* that is specified for X= or Y= in the TRANSLATE statement is the amount that the graphics output is moved.

For example, suppose your imported graphics output is positioned in the upper-left corner of the display. To move it right 10% and down 5%, use the following statement:

```
translate x=10 y=-5;
```

Examples

The following examples illustrate major features of the GIMPORT procedure. For illustration purposes, these examples create a CGM using SAS/GRAPH software and import the resulting CGM by using the GIMPORT procedure. Ordinarily, you would use the GIMPORT procedure to import graphics output that is generated by another software package.

Note: Because this example uses a CGM device driver to produce a graphics stream file, you may need to respecify a device driver for your output device. In addition, these examples use the HSIZE= and VSIZE= graphics options to set a specific size for the graphics output area for the CGM so that the second example can illustrate the use of

the SCALE and TRANSLATE statements. Depending on the output device that you are using, you may need to adjust the HSIZE= and VSIZE= values in this example and the values in the SCALE and TRANSLATE statements in the second example. △

Example 1: Creating and Importing a CGM

Procedure features:

GIMPORT statement options:

FILEREF=
FILETYPE=
FORMAT=

Other features:

FILENAME statement
GOPTIONS statement

Sample library member: GIPCRCGM



This example creates a CGM in binary format by directing SAS/GRAPH output to a graphics stream file (GSF) and using a CGM device driver. It uses the GIMPORT procedure to import the resulting CGM into SAS/GRAPH where it can be viewed and stored in a catalog. (See Chapter 2, “SAS/GRAPH Programs,” on page 25 for additional information on catalog entries and graphics stream files.) The output shows the imported version of the graphic. Note that the output uses the default font because the specified fonts are unavailable. Example 2 on page 979 shows how to map these fonts to get the output that you want. Also see “About Font Mapping” on page 972 for additional information.

Assign the fileref for a graphics stream file and set the graphics environment. Set graphics stream file characteristics, and select the CGM device driver for binary CGM.

```
filename gsasfile 'external-file';
goptions reset=global gunit=pct border cback=white
        colors=(black)
        gaccess=gsasfile gsfmode=replace
        noprompt device=cgm
        hsize=5 in vsize=5 in
        vpos=60 hpos=150;
```

Define titles and footnote for slide.

```
title1 f=script h=7 'Title One is SCRIPT Font';
title2 f=centb h=5 'Title Two is CENTB Font';
title3 f=zapf h=5 'Title Three is ZAPF Font';
footnote h=3 f=swiss j=r 'GIPCRGM ';
```

Generate a slide. The graphics output is stored in the GSF file that was specified with the `fileref` and in the `GOPTIONS` statement.

```
proc gslide;
run;
quit;
```

Reset the graphics environment.

```
goptions reset=goptions border cback=white
        colors=(black);
```

Import the GSF file created by the CGM device driver. `FILEREF=` specifies the `fileref` where the CGM is located. `FILETYPE=` specifies the type of file to be imported. `FORMAT=` specifies the format of the CGM being imported.

```
proc gimport fileref=gsasfile
        filetype=cgm
        format=binary;
run;
```

Output 33.1 shows the font list that is displayed in the log file. The font list contains all of the fonts that are used by the CGM. The warning messages following the font list indicate which fonts can be remapped using the `MAP` statement.

Output 33.1 Font List

```

.
.
.

NOTE: These fonts are used in this CGM file. You may use the MAP statement
      to map these fonts to SAS/GRAPH

fonts.
1. SIMPLEX
2. BRUSH
3. CENTB
4. CENTBE
5. CENTBI
6. CENTBIE
7. CENTX
8. CENTXE
9. CENTXI
10. CENTXIE
11. GERMAN
12. GITALIC
13. DUPLEX
14. COMPLEX
15. TRIPLEX
16. TITALIC
17. ITALIC
18. OLDENG
19. SCRIPT
20. CSCRIPT
21. SWISS
22. SWISSE
23. SWISSB
24. SWISSBE
25. SWISSBI
26. SWISSBIE
27. SWISSX
28. SWISSXE
29. SWISSXB
30. SWISSXB
31. SWISSXBE
32. SWISSI
33. SWISSIE
34. SWISSL
35. SWISSLE
36. ZAPF
37. ZAPFE
38. ZAPFB
39. ZAPFBE
40. ZAPFBI
41. ZAPFBIE
42. ZAPFI
43. ZAPFIE

WARNING: Unspecified font index 19. Font has been mapped to the default font.
WARNING: Unspecified font index 3. Font has been mapped to the default font.
WARNING: Unspecified font index 36. Font has been mapped to the default font.
WARNING: Unspecified font index 21. Font has been mapped to the default font.

.
.
.

```

Example 2: Adjusting the Graphics Output

Procedure features:

SCALE statement

TRANSLATE statement

MAP statement

Sample library member: GIPGROUT



This example imports the CGM file that was created in the earlier example and modifies the output. This example uses the `SCALE` and `TRANSLATE` statements to correct the size and position of the imported CGM. The `MAP` statement is also used to substitute a SAS/GRAPH software font for a font in the CGM.

Assign the fileref for a GSF file and set the graphics environment.

```
filename gsasfile 'external-file';
goptions reset=goptions gunit=pct border cback=white
          colors=(black) htitle=6 htext=3
          vpos=60 hpos=150;
```

Import the GSF file created by the CGM device driver. The `SCALE` statement specifies the scale factor for the values of the x and y coordinates. The `TRANSLATE` statement specifies the amount that the imported graphics output should be moved horizontally and vertically. The `MAP` statements remap the fonts shown in the first example.

```
proc gimport fileref=gsasfile filetype=cgm format=binary;
  scale x=.7 y=.8;
  translate x=3.5 y=10;
  map 'SCRIPT' to script;
  map 'CENTB' to centb;
  map 'ZAPF' to zapf;
  map 'SWISS' to swiss;
run;
```

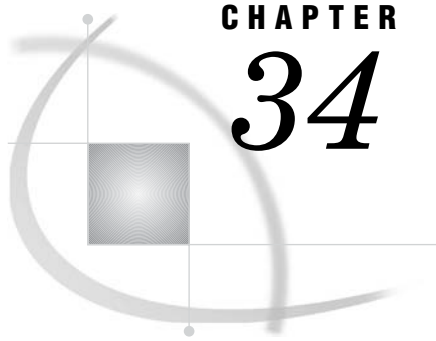
Output 33.2 shows the font list that is displayed in the log file. Note that no warning messages follow the font list because all of the fonts that are used in the CGM have been remapped.

Output 33.2 Font List

```
.  
. .  
. .  
NOTE: These fonts are used in this CGM file. You may use the MAP statement  
      to map these fonts to SAS/GRAPH  
fonts.  
1. SIMPLEX  
2. BRUSH  
3. CENTB  
4. CENTBE  
5. CENTBI  
6. CENTBIE  
7. CENTX  
8. CENTXE  
9. CENTXI  
10. CENTXIE  
11. GERMAN  
12. GITALIC  
13. DUPLEX  
14. COMPLEX  
15. TRIPLEX  
16. TITALIC  
17. ITALIC  
18. OLDENG  
19. SCRIPT  
20. CSCRIPT  
21. SWISS  
22. SWISSE  
23. SWISSB  
24. SWISSBE  
25. SWISSBI  
26. SWISSBIE  
27. SWISSX  
28. SWISSXE  
29. SWISSXB  
30. SWISSXB  
31. SWISSXBE  
32. SWISSI  
33. SWISSIE  
34. SWISSL  
35. SWISSLE  
36. ZAPF  
37. ZAPFE  
38. ZAPFB  
39. ZAPFBE  
40. ZAPFBI  
41. ZAPFBIE  
42. ZAPFI  
43. ZAPFIE  
. .  
. .
```

References

- ANSI X3.122–1986, *Computer Graphics Metafile for the Storage and Transfer of Picture Description Information*.
- Arnold, D.B. and Bono, P.R. (1988), *CGM and CGI: Metafile Interface Standards for Computer Graphics*, New York: Springer-Verlag.



The GKEYMAP Procedure

<i>Overview</i>	983
<i>Concepts</i>	983
<i>About Key Maps and Device Maps</i>	983
<i>What Key Maps Do</i>	985
<i>What Device Maps Do</i>	986
<i>Using Key Maps and Device Maps</i>	986
<i>Asymmetrical Maps</i>	986
<i>Seeing What Characters in a Font are Available</i>	987
<i>About the GKEYMAP Data Set</i>	987
<i>GKEYMAP Data Set Variables</i>	987
<i>Procedure Syntax</i>	988
<i>PROC GKEYMAP Statement</i>	988
<i>Examples</i>	990
<i>Example 1: Modifying a Key Map</i>	990

Overview

The GKEYMAP procedure creates key maps and device maps that compensate for differences between the way that characters are encoded internally by SAS/GRAPH software and the way that they are encoded by different operating environments and output devices. In addition, the GKEYMAP procedure can create SAS data sets from existing key maps and device maps, either Institute-supplied or user-generated. This capability is useful when you want to make minor alterations in a large key map or device map and you do not want to or cannot re-create the original data set with a DATA step.

The Institute supplies key maps for many keyboard configurations and operating-environment character representations. Your SAS Software Consultant should have selected the appropriate key map for your site. If the Institute-supplied device maps and key maps do not meet your needs, you can use this procedure to modify an existing map or create a new one.

Concepts

About Key Maps and Device Maps

The characters A through Z (upper- and lowercase), 0 through 9, and many symbols and national characters are represented by a set of hexadecimal codes. However, a

character may be represented by one code for the keyboard, another code for the operating environment, and yet another for the output device. To resolve these differences, SAS/GRAPH software stores all characters using its own internal encoding scheme, which is a set of hexadecimal values that are associated with all supported characters. Figure 34.1 on page 985 shows these internal character encoding (ICE) codes. To view such a table for yourself, run the following code, which uses the Swiss font:

```
goptions keymap=none;
proc gfont nb name=swiss hex;
run;
quit;
```

To accommodate differences in the encoding of characters, you must be able to translate the hexadecimal codes generated by your keyboard or operating environment into the corresponding SAS/GRAPH internal encoding. A key map gives you this ability.

You also must be able to convert the internal encoding that is used by SAS/GRAPH software to the codes required to produce the corresponding hardware characters on your output device. A device map gives you this ability.

Key maps and device maps are SAS catalog entries. Institute-supplied key maps and device maps are stored in the catalog SASHELP.FONTS. User-generated key maps and device maps are stored in the catalog GFONT0.FONTS. Key maps are stored with the extension KEYMAP (for example, GERMAN.KEYMAP), and device maps are stored with the extension DEVMAP (for example, DEFAULT.DEVMAP).

Figure 34.1 SAS/GRAPH Internal Character Encoding

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
	©	®	■	™	◦	´	¨	œ	ø	‘	’	“	”	□		
10		‡	“	¶	§	∅	ø	↑	↓	→	←	»	↔			
20		‡	”	#	\$	%	&	’	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
80	Ç	Ü	é	â	ä	à	á	ç	ê	ë	è	ï	î	ì	Ä	Å
90	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	Pt	f
A0	á	í	ó	ú	ñ	Ñ	ª	º	¿	×	¬	½	¼	¡	«	»
B0	´	˘	¨	˙	˚	˛	˜	β	˘	˘		ǧ	†	‡		
C0																
D0	ı	ı	´	˘	¨	˙	˚	˛	˜	˘	˘		ǧ	§	§	
E0	∏	◻	•	◻	μ	Đ	Þ	đ	þ	⅛	⅜	⅝	¾	⅞	⅓	⅔
F0	Č	±	≥	≤	Ć	Š	+	Ž	č	ć	š	ž	≠			

Note: Positions 00-1F are reserved.
Note: SAS Institute reserves the right to change, at any time, the character displayed and the hexadecimal code returned for all undefined codes.

Note: Positions 00-1F are reserved. SAS Institute reserves the right to change, at any time, the character displayed and the hexadecimal code returned for all undefined codes. △

What Key Maps Do

A key map changes the code generated by a keyboard key to the value corresponding to the SAS/GRAPH internal character encoding. Otherwise, a different character (or no character) may be drawn when the character is requested in a SAS/GRAPH software font.

Key maps are required when the code that is sent to the operating environment does not match the SAS/GRAPH internal encoding for the character corresponding to the key that is pressed. They are useful for generating a character in a software font that is not

available on your keyboard or when the same key on different keyboards sends a different character to the operating environment. They are also useful for creating new characters by combining existing characters with accent characters (called *diacritics*).

Note: In Figure 34.1 on page 985, the diacritic characters specified by the codes D2 through DB are backspaced before being drawn and can be used to create new characters (characters resulting from codes B0 through B7, B9, and BA are not backspaced before being drawn). See Example 1 on page 990 for an example of using a diacritic character as an accent. Two commonly used characters have already been created for you: the character located in position F0 of the ICE table could be created by combining DA with an uppercase C, and the character located in position BC could be created by combining DB with an uppercase G. \triangle

What Device Maps Do

A device map maps the code stored in the SAS/GRAPH internal encoding to the code required to reproduce the character on the output device when a particular hardware character is requested in a SAS/GRAPH program.

You usually use device maps in these two situations:

- reversing the translation performed by key maps (if needed). To display the proper hardware character, you must use a device map to convert the SAS/GRAPH internal encoding of the character back to the encoding that the device expects.
- accounting for differences between the code that represents a character on the operating environment and the code or codes required to generate the same character as a hardware character on an output device. The problem can be further complicated if you have multiple output devices, each with its own way of generating a particular character using hardware text.

Using Key Maps and Device Maps

You use key maps and device maps by specifying them with the KEYMAP= or DEVMAP= options in a GOPTIONS statement. You also can specify a device map by filling in the DEVMAP field in the Detail window of the device entry for the device driver that you are using.

For example, if you use the GKEYMAP procedure to generate a key map called MYKEYMAP, you can specify it with a statement like this:

```
goptions keymap=mykeymap;
```

Once you specify MYKEYMAP as your current key map, you can press a key and the code it generates is translated by MYKEYMAP into the ICE code that is specified by the key map.

When you specify a device map with the DEVMAP= graphics option and you use a hardware character set, mapped characters are converted from their SAS/GRAPH internal encoding to the codes required to display the corresponding characters on your device. See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for more information on the KEYMAP= and DEVMAP= graphics options.

Asymmetrical Maps

It is possible, and sometimes necessary, to define a key map or device map that is not symmetrical (that is, two or more input character codes map to the same output character code). For example, if you define a key map to map the keyed character A to the internal encoding for B, the keyed characters A and B both map to the internal encoding for B, but no code maps to A. This situation may make it impossible for you to display certain characters defined in software fonts.

Seeing What Characters in a Font are Available

To see what characters in a font can be displayed if a particular key map is used, do the following:

- 1 Use the `KEYMAP=` option in a `GOPTIONS` statement to specify the key map that you are interested in.
- 2 Then, use the `GFONT` procedure with the `ROMHEX` option to display the font that you want to use.

The hexadecimal values and corresponding font characters that are displayed are the ones available under the specified map. If the map is not symmetrical, a warning is issued. See Chapter 32, “The `GFONT` Procedure,” on page 939 for more information on using hexadecimal values to display special characters.

About the GKEYMAP Data Set

To generate a key map or device map, you must create a data set that contains the mapping information and use that data set as input for the `GKEYMAP` procedure. The mapping information is specified as values for the variables in the data set, which should contain one observation for each character or key to be mapped. Any characters not specified in the data set are passed through the map unchanged.

GKEYMAP Data Set Variables

To provide information on the character mapping that is to be performed for a key map or a device map, you must use a variable named `FROM` to specify the character that you are mapping from, and a variable named `TO` to specify the character to map to. For key maps, these are the only variables in the data set. For device maps, you may also need variables named `CHARTYPE` and `TOLEN`.

Here are definitions for these variables:

CHARTYPE

specifies which hardware character set to use when a device requires that you select an alternate character set in order to display certain characters.

`CHARTYPE` is a numeric variable.

All of the characters in the `TO` string for a particular `FROM` value must use the same character set. The `CHARTYPE` variable is required if you use the `MULTFONT` option in the `PROC GKEYMAP` statement; otherwise, it is ignored. (The `CHARTYPE` variable is always ignored when you are creating a key map.) The `CHARTYPE` value must match a value listed in the `Chartype` field in the `Chartype` window of the device entry for the device to which the map is applied. However, you can set the `CHARTYPE` variable to a missing value to specify that the character can be drawn in any hardware character set.

FROM

specifies the character you are mapping from. `FROM` is a character variable. For each observation, the `FROM` variable should contain a single character value. Any characters after the first are ignored. The data set must be sorted by the `FROM` variable.

Featured in: Example 1 on page 990

TO

specifies the string that the character in the `FROM` variable is mapped to. `TO` is a character variable.

For device maps, if the `TO` variable contains more than one character, you must also specify `TYPE=MAP1N` in the `PROC GKEYMAP` statement to indicate that a

single FROM character is being mapped to multiple TO characters. In addition, you must include the TOLEN variable in the data set to specify the length of each TO string. If you specify TYPE=MAP11 in the PROC GKEYMAP statement or if you do not use the TYPE= option, only the first byte of the TO string is recognized.

Featured in: Example 1 on page 990

TOLEN

specifies the length of the string in the TO variable. TOLEN is a numeric variable. The TOLEN variable is used only with device maps and is required if you specify TYPE=MAP1N in the PROC GKEYMAP statement; otherwise, it is ignored.

Procedure Syntax

Requirements: The NAME= argument is always required. To create a key map or device map, the DATA= argument is required. To output a data set, the OUT= argument is required.

```
PROC GKEYMAP NAME=map-name
           data-set-argument
           <option(s)>;
```

PROC GKEYMAP Statement

The PROC GKEYMAP Statement names the key map or device map to be created or output as a data set. If the procedure creates a key map or a device map, it identifies the data set that is used as input. If it outputs a map, it identifies the data set to which the map is written.

Syntax

```
PROC GKEYMAP NAME=map-name
           data-set-argument
           <option(s)>;
```

data-set-argument must be one or more of the following:

```
DATA=keymap-data-set
OUT=output-data-set
```

option(s) can be one or more of the following:

```
DEVICE=device-name
DEVMAP | KEYMAP
TYPE=MAP11 | MAP1N
MULTFONT
```

Required Arguments

NAME=*map-name*

identifies the map that is to be created or converted to a SAS data set. Key maps are stored as *map-name*. KEYMAP, and device maps are stored as *map-name*. DEVMAP. The value of the KEYMAP or DEVMAP option determines the type of map and the extension added to *map-name*. It is possible to use the same *map-name* value for both a key map and a device map.

If you create a key map or device map, the map is stored as an entry in the catalog GFONT*n*.FONTS where *n* is a number from 1 to 9, and you must use a LIBNAME statement to specify a libref for GFONT*n*. See “About the Libref GFONT0” on page 941 for details.

If you specify an existing key map or device map, SAS/GRAPH software searches for the map using the same search path that it uses to search for fonts. See “Font Locations” on page 77 for details .

Featured in: Example 1 on page 990.

DATA=*keymap-data-set*

identifies the input data set for the GKEYMAP procedure. Used only when you are creating a key map or device map.

See also: “SAS Data Sets” on page 29 and “About the GKEYMAP Data Set” on page 987.

Featured in: Example 1 on page 990.

OUT=*output-data-set*

identifies the output data set to which the data from a key map or device map are to be written. Used only when you output an existing key map or device map as a SAS data set.

Featured in: Example 1 on page 990.

Options

You can specify as many options as you want and list them in any order.

DEVICE=*device-name*

specifies the device driver that a device map is associated with, where *device-name* is the name of an entry in a device catalog. DEVICE= is not required when creating a device map, but it can be used if you want to limit the use of the device map to one particular driver. If you do not use DEVICE=, the device map can be used with any device. DEVICE= is valid only if you are creating a device map.

DEVMAP | KEYMAP

specifies whether you are working with a device map or a key map. The default is KEYMAP unless you use an option that can be used only with DEVMAP. This option also specifies the type of map you are outputting as a data set.

Featured in: Example 1 on page 990.

TYPE=MAP11 | MAP1N

specifies whether you are mapping characters in a device map one-to-one or one-to-many. If you specify TYPE=MAP11 (the default), each character in a graphics text string is mapped to only one character on the output device. If you specify TYPE=MAP1N, a single character in a graphics text string can be mapped to multiple characters on the output device. For example, if two characters have to be sent to the graphics output device to display a single hardware character, specify TYPE=MAP1N. Specify TYPE=MAP1N only when you create a device map.

MULTFONT

specifies that an alternate hardware character set is required to display one or more characters in the device map. Specify the MULTFONT option only when you create a device map.

Creating a Data Set from an Existing Key Map or Device Map

To generate a data set from an existing key map or device map, follow these steps:

- 1 Specify the name of the key map or device map with the NAME= argument. If the map is user generated, you must first submit a LIBNAME statement to associate the libref GFONT0 with the location where the map is stored, and NAME= must specify the name that was specified for the key map or device map when it was created. If the map is an Institute-supplied map, it is located in the catalog SASHELP.FONTS, and you do not need to submit a LIBNAME statement to access it.
- 2 In the OUT= argument, specify the name of the data set to which the data are to be written. By default, the data set is written to the temporary library WORK.
- 3 Use the DEVMAP option if a device map is selected.
- 4 Optionally, use the PRINT procedure to display the newly created data set (most values will be unprintable, so you should use a \$HEX2. format for the FROM and TO variables).

Creating and Using Key Maps and Device Maps

To create and use a key map or device map, follow these steps:

- 1 Submit a LIBNAME statement that associates the libref GFONT0 with the location where your map is to be stored.
- 2 Create a data set that contains the mapping information you need. You can use a DATA step to create all of the mapping information for the key map or device map, or you can create a data set from an existing key map or device map, then update that data set with the mappings that you need. This process is illustrated in Example 1 on page 990.
- 3 Use the GKEYMAP procedure to create the key map or device map, using as input the data set that contains the mapping information. The GKEYMAP procedure stores the map in the catalog GFONT0.FONTS.
- 4 Use the KEYMAP= or DEVMAP= option in a GOPTIONS statement to assign the key map or device map in your SAS session. The specified map is used automatically in your SAS/GRAPH programs. (The device map is used only when you use a hardware character set.)

Examples

Example 1: Modifying a Key Map

Procedure features:

GKEYMAP options:

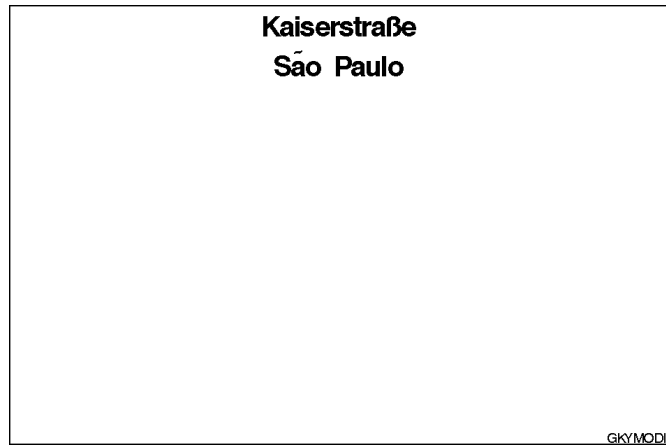
DATA=

```
KEYMAP
NAME=
OUT=
```

Other features:

```
DATA step
GOPTIONS procedure
GOPTIONS statement
LIBNAME statement
SORT procedure
```

Sample library member: GKYMODIF



This example shows how to change multiple characters in an existing key map. It assumes that the national characters ß and ã are not on your keyboard, so you want to create a key map that provides them.

To provide the ß character, this example's key map converts the @ character into the SAS/GRAPH internal encoding for ('B8'x). Whenever the @ character is typed in text that is displayed with a software font, the character ß is drawn instead. In this case, the replacement character uses the text position that would have been used by the typed character.

Note: Once you have modified your key map so that @ is mapped to ß, you can no longer generate @ in a software font from your keyboard when the key map is in effect. \triangle

To provide the ã character, which is not on the keyboard or in the ICE table, this example's key map converts the asterisk (*) into the SAS/GRAPH internal encoding for the accent character 'D5'x (a tilde). In this case, when the character * is typed, the resulting tilde does not take up a text position but is backspaced and used as an accent over the character preceding it in the text. To create the ã character, therefore, the text must contain the two characters a*.

Note: The example updates the current key map rather than creating a new key map so that all of the other character mapping in the key map remains in effect. \triangle

Assign the libref and set the graphics environment. LIBNAME associates the libref GFONTO with the location of the SAS data library where your device maps and key maps are stored.

```
libname gfont0 'SAS-data-library';
options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss ftitle=swissb htext=6;
```

Determine the name of the current key map. The SAS log in Output 34.1 shows that the keymap name is DEFAULT.

```
proc goptions
    option=keymap;
run;
```

Copy the DEFAULT key map to a temporary SAS data set. NAME= specifies the DEFAULT key map as input to the procedure. OUT= specifies the data set TEMP, which is created from the specified key map.

```
proc gkeymap name=default
    out=temp;
run;
```

Create data set NEW. NEW will be used to create the key map for the character conversions. Values for the FROM variable are the keyboard characters to be converted. Values for the TO variable are hexadecimal codes from the SAS ICE table. OUTPUT is required to write a separate observation for each character to be mapped.

```
data new;
    from='@';
    to='b8'x;
    output;
    from='*';
    to='d5'x;
    output;
run;
```

Sort data set NEW and update data set TEMP with the mapping information. The data set NEW must be sorted by the FROM variable before its observations can be used to update data set TEMP.

```
proc sort data=new;
    by from;
data temp;
    update temp new;
    by from;
run;
```

Create a new key map from the modified data set. NAME= assigns a name to the new key map. DATA= specifies the data set TEMP as input to the procedure. KEYMAP specifies that the map being generated is a key map (the default).

```
proc gkeymap name=mykeymap
             data=temp
             keymap;
run;
```

Specify the new key map in a GOPTIONS statement. KEYMAP= specifies the name of the new key map so that when the characters @ and a* are specified in TITLE statements, the characters ß and ã are displayed in the output.

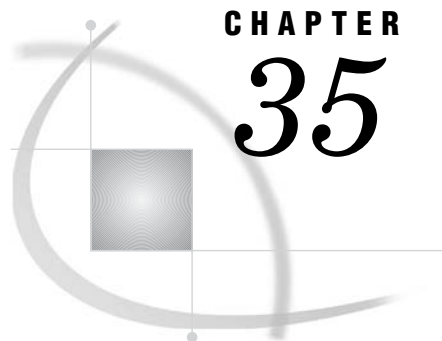
```
goptions keymap=mykeymap;
```

Print two titles with the special characters. The character @ is typed where the character ß should print, and the character * is typed after the character it will accent.

```
title1 'Kaiserstraße';
title2 'Sa*o Paulo';
footnote j=r 'GKYM0DIF ';
proc gslide;
run;
quit;.
```

Output 34.1 Log from GOPTIONS Procedure

<pre>SAS/GRAPH software options and parameters (executing in DMS Process environment) KEYMAP=DEFAULT Input character map for hardware and software text</pre>
--



CHAPTER 35

The GMAP Procedure

<i>Overview</i>	996
<i>About Block Maps</i>	996
<i>About Choropleth Maps</i>	997
<i>About Prism Maps</i>	998
<i>About Surface Maps</i>	998
<i>Concepts</i>	999
<i>About Map Data Sets</i>	999
<i>About Traditional Data Sets</i>	999
<i>Required Variables</i>	999
<i>Segment Variable</i>	1000
<i>LONG and LAT Variables</i>	1000
<i>Traditional Map Data Sets Containing X, Y, LONG, and LAT</i>	1000
<i>Traditional Map Data Sets Containing Only X and Y</i>	1001
<i>About Feature Tables</i>	1001
<i>\$GEOREF format</i>	1001
<i>Merging Feature Tables with Response Data Sets</i>	1001
<i>Viewing Map Data Sets</i>	1001
<i>Speciality Map Data Sets</i>	1003
<i>About Response Data Sets</i>	1003
<i>Using the Response Data Set with the Map Data Sets</i>	1003
<i>About Response Variables</i>	1004
<i>About Response Levels</i>	1004
<i>About Identification Variables</i>	1005
<i>Displaying Map Areas and Response Data</i>	1005
<i>Summary of Use</i>	1006
<i>Accessing SAS Maps Online</i>	1006
<i>Procedure Syntax</i>	1007
<i>PROC GMAP Statement</i>	1007
<i>ID Statement</i>	1009
<i>BLOCK Statement</i>	1009
<i>CHORO Statement</i>	1017
<i>PRISM Statement</i>	1022
<i>SURFACE Statement</i>	1030
<i>Using FIPS Codes and Province Codes</i>	1033
<i>Using Formats for Maps</i>	1035
<i>SAS/GRAPH Map Data Sets Reference Information</i>	1038
<i>Accessing Detailed Descriptions of Map Data Sets</i>	1038
<i>Customizing SAS/GRAPH Map Data Sets</i>	1039
<i>Subsetting Traditional Map Data Sets</i>	1039
<i>Reducing Traditional Map Data Sets</i>	1040
<i>Projecting Traditional Map Data Sets</i>	1040

<i>Controlling the Display of Lakes</i>	1041
<i>Creating Traditional Map Data Sets</i>	1041
<i>Creating a Unit Area that is a Single Polygon</i>	1042
<i>Creating a Unit Area that Contains Multiple Polygons</i>	1042
<i>Creating a Unit Area that Contains Enclosed Polygons as Holes</i>	1043
<i>Creating a Unit Area that Contains Enclosed Polygons as Cities</i>	1044
<i>Examples</i>	1045
<i>Example 1: Producing a Simple Block Map</i>	1045
<i>Example 2: Specifying Response Levels in a Block Map</i>	1047
<i>Example 3: Assigning a Format to the Response Variable</i>	1049
<i>Example 4: Producing a Simple Choropleth Map</i>	1052
<i>Example 5: Creating Maps with Drill-down for the Web</i>	1054
<i>Example 6: Labeling the States on a U.S. Map</i>	1061
<i>Example 7: Producing a Simple Prism Map</i>	1063
<i>Example 8: Specifying Midpoints in a Prism Map</i>	1065
<i>Example 9: Producing a Simple Surface Map</i>	1066
<i>Example 10: Rotating and Tilting a Surface Map</i>	1068
<i>Example 11: Creating a Map Using the Feature Table</i>	1069

Overview

The GMAP procedure produces two-dimensional (choropleth) or three-dimensional (block, prism, and surface) color maps that show variations of a variable value with respect to an area. A wide assortment of map data sets are available with SAS/GRAPH software.

Use the GMAP procedure to

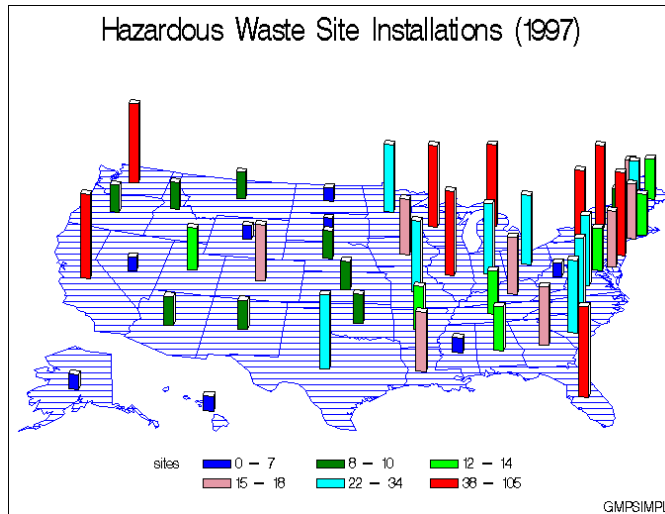
- produce maps
- summarize data that vary by physical area
- show trends and variations of data between geographic areas
- highlight regional differences or extremes.

About Block Maps

Block maps display a block at the approximate center of each map area to convey information about response variable values. The height of each block represents a response level. The height is not directly proportional to the value of the response variable. Instead, the block heights increase in order of the response levels.

Figure 35.1 on page 997 shows a simple block map of hazardous waste sites that are installed in each state. The number of sites in each state (the response value) is represented by the height of the block.

Figure 35.1 Block Map



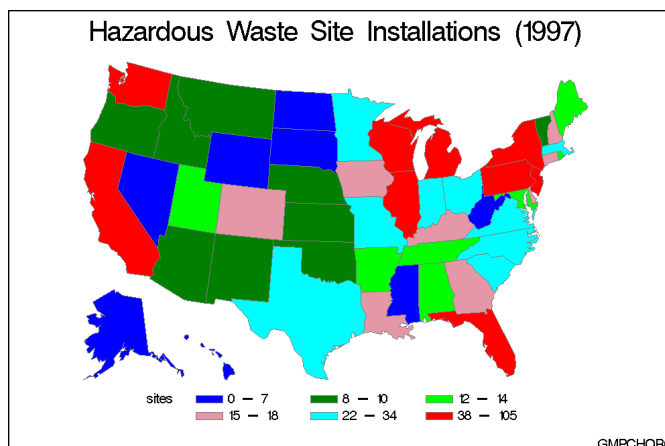
The program for this map is in Example 1 on page 1045. For more information on producing block maps, see “BLOCK Statement” on page 1009.

About Choropleth Maps

Two-dimensional (choropleth) maps indicate levels of magnitude or response levels of the corresponding response variable by filling map areas with different colors and patterns.

Figure 35.2 on page 997 shows a choropleth map of hazardous waste sites that are installed in each state. The number of sites in each state (the response value) is represented by the pattern that is assigned to the state.

Figure 35.2 Two-dimensional (Choropleth) Map



The program for this map is in Example 4 on page 1052.

You can also produce a simple choropleth map that shows an outline of a map’s areas by specifying your map data set as both the map data set and the response data set in a GMAP statement and adding a PATTERN statement with VALUE=EMPTY. For more

information on the PATTERN statement, see “PATTERN Statement” on page 169. For more information on producing choropleth maps, see “CHORO Statement” on page 1017.

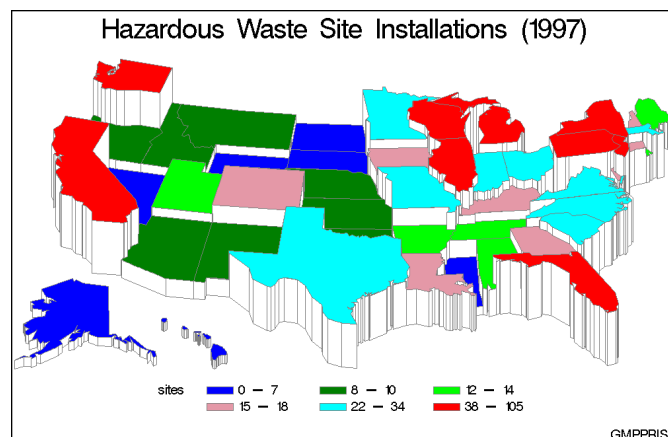
About Prism Maps

Prism maps use polyhedrons (raised polygons) in the shape of each map area to convey information about response variable values. The height of each polyhedron, or prism, represents an ordinal level of the response variable. Prism heights increase in order of response levels. That is, the lowest prisms correspond to the first level, and the tallest prisms correspond to the last level.

You can alter the perspective of the map by selecting a viewing position (the point in space from which you view the map). You can also change the position of the light source so that the shadowing on the prisms enhances the illusion of height.

Figure 35.3 on page 998 shows a prism map of hazardous waste sites installed in each state. The number of sites in each state (the response value) is represented by the height of the state.

Figure 35.3 Prism Map



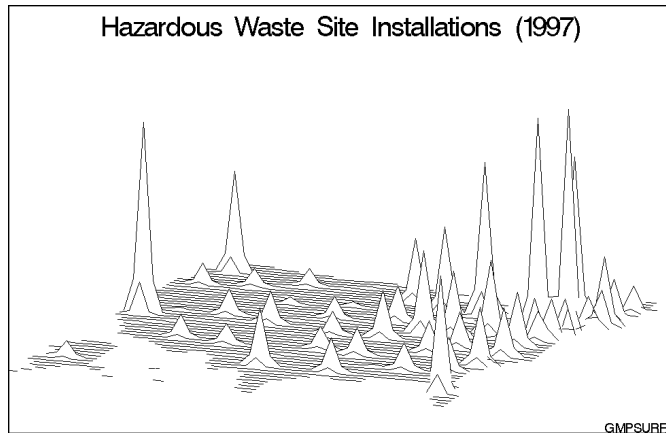
The program for this map is in Example 7 on page 1063. For more information on producing prism maps, see “PRISM Statement” on page 1022.

About Surface Maps

Surface maps display a spike at the approximate center of each map area to convey information about response variable values. The height of the spike corresponds to the relative value of the response variable, not to the actual value of the response variable. Thus, a spike that represents a value of 100 may not be exactly 10 times higher than a spike that represents a value of 10. Map area boundaries are not drawn.

Surface maps provide no clear map area boundaries and no legend. Thus, surface maps provide a simple way to judge relative trends in the response data but are an inappropriate way to represent specific response values.

Figure 35.4 on page 999 shows a surface map of hazardous waste sites that are installed in each state. The number of sites in each state (the response value) is represented by the height of the spike.

Figure 35.4 Surface Map

The program for this map is in Example 9 on page 1066. For more information on producing surface maps, see “SURFACE Statement” on page 1030.

Concepts

Map data sets and response data sets are used in the GMAP procedure. These data sets must contain the required variables or the procedure stops and you get an error message. Depending on the type of map data set used, the map and response data sets can be used individually in the GMAP procedure or merged into a single data set to be used in the GMAP procedure. Each data set must contain the same identification variable.

About Map Data Sets

There are two types of map data sets: traditional map data sets and feature tables. Each uses a different data arrangement to store the spatial information needed to create maps. All of the map data delivered with SAS/GRAPH is available in both the traditional map data set and feature table format.

About Traditional Data Sets

A *traditional map data set* is a SAS data set that contains coordinates that define the boundaries of map areas, such as states or counties.

Required Variables

A traditional map data set must contain at least these variables:

- a numeric variable named X that contains the horizontal coordinates of the boundary points. The value of this variable could be either projected or unprojected. If unprojected, X represents longitude.
- a numeric variable named Y that contains the vertical coordinates of the boundary points. The value of this variable could be either projected or unprojected. If unprojected, Y represents latitude.

- one or more variables that uniquely identify the areas in the map. Map area identification variables can be either character or numeric and are indicated in the ID statement.

The X and Y variable values in the traditional map data set do not have to be in any specific units because they are rescaled by the GMAP procedure based on the minimum and maximum values in the data set. The minimum X and Y values are in the lower-left corner of the map, and the maximum X and Y values are in the upper-right corner.

Traditional map data sets in which the X and Y variables contain longitude and latitude should be projected before you use them with PROC GMAP. See Chapter 39, “The GPROJECT Procedure,” on page 1161 for details.

Segment Variable

Optionally, the traditional map data set also can contain a variable named SEGMENT to identify map areas that comprise noncontiguous polygons. Each unique value of the SEGMENT variable within a single map area defines a distinct polygon. If the SEGMENT variable is not present, each map area is drawn as a separate closed polygon that indicates a single segment.

The observations for each segment of a map area in the map data set must occur in the order in which the points are to be joined. The GMAP procedure forms map area outlines by connecting the boundary points of each segment in the order in which they appear in the data set, eventually joining the last point to the first point to complete the polygon.

LONG and LAT Variables

In addition to the variables described in “Required Variables” on page 999, the SAS/GRAPH map data sets can also contain the following variables:

- a numeric variable named LONG containing the unprojected longitude (in radians or degrees) of the boundary points
- a numeric variable named LAT containing the unprojected latitude (in radians or degrees) of the boundary points.

The GMAP procedure uses the values of the X and Y variables to draw the map. Therefore, if you want to produce an unprojected map by using the values in LONG and LAT, you would have to rename LONG and LAT to X and Y first.

SAS/GRAPH software includes a number of predefined map data sets. These data sets are described in “Viewing Map Data Sets” on page 1001.

Traditional Map Data Sets Containing X, Y, LONG, and LAT

Most of the traditional map data sets that are provided with SAS/GRAPH software contain four coordinate variables (X, Y, LONG, and LAT). In this case, X and Y are always projected values that will be used by the SAS/GRAPH procedures (by default). If you need to use the unprojected values that are contained in the LONG and LAT variables, then you must

- 1 drop the existing X and Y variables
- 2 rename the LONG and LAT variables to X and Y.

The MAP= value in the GMAP procedure automatically uses X and Y. See “Input Map Data Sets that Contain Both Projected and Unprojected Values” on page 1164 for more details.

Traditional Map Data Sets Containing Only X and Y

The traditional map data sets that contain X and Y variables (and no LONG and LAT variables), are usually projected maps. However, there are a few traditional map data sets for the US and Canada that contain X and Y values that are unprojected longitude and latitude. In this case, you will need to use the GPROJECT procedure to project the map (see Chapter 39, “The GPROJECT Procedure,” on page 1161).

Note: You can determine whether a SAS traditional map data set is projected or unprojected by looking at the description of each variable that is displayed when you use the CONTENTS procedure or by browsing the MAPS.METAMAPS data set. Δ

About Feature Tables

An alternative to using the traditional map data set is the feature table. While the traditional map data set stores the spatial information across multiple observations, the feature table uses a data arrangement to store all of the spatial information in a single variable value. The feature table’s data arrangement uses the \$GEOREF SAS/GRAPH format.

\$GEOREF format

The \$GEOREF format stores spatial information in binary data streams, making it possible to store as a single variable value all the information needed to draw a map area. Thus, the feature tables use only a single observation for each map area, and they treat a field of spatial information just like any other information that can be added to a data set. Each \$GEOREF value points to a corresponding traditional map data set to retrieve the coordinate values. The traditional map data set associated with the feature table must be located in the SAS library with the feature table for GMAP to proceed correctly.

To locate the variable that contains the spatial information, run PROC CONTENTS on a feature table. In the Output window, the variable containing the spatial information will have \$GEOREF as the value in the column labelled Format.

Note: Some feature tables, like MAPS.NAMES, have more than one \$GEOREF format variable. Δ

Merging Feature Tables with Response Data Sets

To display response data with a feature table, the feature table must be merged with a response data set. The merged data set is then specified by the DATA= option in the PROC GMAP statement. The combined data set can be used repeatedly for generating maps, without having to merge the map and response data again.

First, a PROC SORT must be used to sort the response and feature tables by a variable that is present within both the data sets. Once sorted, the data sets can then be merged with a SQL or DATA step MERGE with the BY variable being the variable used to sort the data sets. Once the data set is merged, the \$GEOREF formatted variable from the feature table becomes the new data set’s identification variable to be used in the GMAP procedure. See Example 11 on page 1069 for more details.

Viewing Map Data Sets

When viewed in SAS, a data set is displayed as a table, with the variable names or labels displayed as column headings and the variable values arranged in columns and

rows. The data sets that contain geometry objects describe a map by its spatial features, so their data tables are referred to as *feature tables*. Because feature tables store the spatial information in a single variable value, the spatial data and response data is viewed as a 1:1 ratio. The traditional MAP data sets define map areas using geometric coordinates, so their data tables are also referred to as *geometry tables*. Traditional map data sets store the geometric coordinates across multiple observations.

In the MAPS library, there is a data set named METAMAPS, which contains meta data about all of the data sets that are delivered in the library. Among the meta data in MAPS.METAMAPS are the following four variables, which you can use to determine which feature table corresponds to a particular geometry table:

Table 35.1

Variable	Description
MEMNAME	Identifies the names of all of the data sets that are delivered in the MAPS library.
MEMCODE	Indicates whether a data set represents a feature table (F) or a geometry table (G).
F_TABLE	Indicates the corresponding feature table for a geometry table. This variable is blank for rows that contain meta data about a feature table.
F_GEOCOL	Indicates the variable, in the feature table, whose values encapsulate the geometry object.

For example, consider the data sets MAPS.ASIA, MAPS.STATES, and MAPS.US. Each of these represents a geometry table, and to locate the corresponding feature tables, you would look in MAPS.METAMAPS to find the MEMNAME values ASIA, STATES, and US. Here are the relevant values on those rows:

Table 35.2

MEMNAME	MEMCODE	F_TABLE	F_GEOCOL
Asia	G	NAMES	CONT95_GEO
STATES	G	US2	GEO_STATE
US	G	US2	_MAP_GEOMETRY_

From these values, you can see that the data sets that are named ASIA, STATES, and US all represent geometry tables because their MEMCODE values are G. The feature table corresponding to the ASIA data set is the data set NAMES, which stores

the spatial information in the variable CONT95_GEO. The feature tables corresponding to STATES and US are both in the data set US2. The spatial information corresponding to STATES is stored in the variable GEO_STATE, and the spatial information corresponding to US is stored in the variable _MAP_GEOMETRY_.

Speciality Map Data Sets

There are several map data sets available with SAS/GRAPH software that allow you to easily label maps:

MAPS.USCENTER

contains the coordinates of the visual center of each state in the U.S. and Washington, D.C., as well as coordinates in the ocean for states that are too small to contain a label. There are two pairs of variables for locating labels using Annotate data sets. The X and Y variables are projected and can be used with the MAPS.US and MAPS.USCOUNTY data sets. The LONG and LAT variables are unprojected longitude and latitude in degrees and can be used with the MAPS.STATES, MAPS.COUNTIES, and MAPS.COUNTY data sets.

MAPS.USCITY

contains the locations of selected cities in the U.S. Many city names occur in more than one state, so you may have to subset by state to avoid duplication. There are two pairs of variables for locating labels using Annotate data sets. The X and Y variables contain projected coordinates and can be used with the MAPS.US and MAPS.COUNTY data sets. The LONG and LAT variables contain the unprojected longitude and latitude in degrees. These can be used to place labels on the MAPS.STATES, MAPS.COUNTIES, or MAPS.COUNTIES data sets.

MAPS.CANCENS

contains the names of the Canadian census divisions. You can use MAPS.CANCENS with the MAPS.CANADA and MAPS.CANADA3 data sets.

For details on each of these data sets, see the MAPS.METAMAPS data set.

About Response Data Sets

A *response data set* is a SAS data set that contains

- one or more response variables that contain data values that are associated with map areas. Each value of the response variable is associated with a map area in the map data set.
- identification variables that identify the map area to which a response value belongs. These variables must be the same as those that are contained in the map data set.

The response data set can contain other variables in addition to these required variables.

Using the Response Data Set with the Map Data Sets

The traditional map data set and the response data set must be used independently in the PROC GMAP statement, where the response data set is specified by the DATA= option and the traditional map data set is specified by the MAP= option. The values of the map area ID variables in the response data set determine the map areas to be included on the map. Unless the ALL option is used in the PROC GMAP statement, only the map areas with response values are shown on the map. As a result, you do not

need to subset your map data set if you are mapping only a small section of the map. However, if you map the same small section frequently, then create a subset of the map data set for efficiency.

If you have a response data set named WORK.SITES, then the syntax for using GMAP might resemble the following:

```
/* if necessary, define a libref pointing to the SAS maps library */
libname maps 'SAS-data-library';
/* generate a map */
proc gmap map=maps.us data=work.sites;
  id state;
  choro region/discrete;
run;
quit;
```

A feature table and response data set are merged using a variable contained in both data sets. The new combined data set becomes the DATA= value in the PROC GMAP statement. When the response data set and the feature table are merged into one, do not use MAP=*map-data-set* in the PROC GMAP statement. The \$GEOREF formatted variable is the ID variable for the combined data set. See Example 11 on page 1069 for more details.

Note: Response data that does not correspond to a map feature will be included in the legend. \triangle

About Response Variables

The GMAP procedure can produce block, choropleth, prism, and surface maps for both numeric and character response variables. Numeric variables fall into two categories: discrete and continuous.

- Discrete variables* contain a finite number of specific numeric values that are to be represented on the map. For example, a variable that contains only the values 1989 or 1990 is a discrete variable.
- Continuous variables* contain a range of numeric values that are to be represented on the map. For example, a variable that contains any real value between 0 and 100 is a continuous variable.

Numeric response variables are always treated as continuous variables unless the DISCRETE option is used in the action statement.

About Response Levels

Response levels are the values that identify categories of data on the graph. The categories that are shown on the graph are based on the values of the response variable. Based on the type of the response variable, a response level can be determined by any of the following:

- a character value
- the MIDPOINTS= option
- a range of numeric values
- a specific numeric value.

When response levels are determined by a character value, the GMAP procedure treats each unique value as a response level. For example, if the response variable

contains the names of ten regions, each region will be a response level, resulting in ten response levels.

When character response levels are determined by the MIDPOINTS= option, any response variable values that do not match one of the specified response level values are ignored.

When response levels are determined by a range of numeric values, each response level has the same number of observations. These options are exceptions to this:

- The LEVELS= option specifies the number of response levels to be used on the map.
- The DISCRETE option causes the numeric variable to be treated as a discrete variable.
- The MIDPOINTS= option chooses specific response level values as medians of the value ranges.

If the response variable values are continuous, then the GMAP procedure assigns response level intervals automatically unless you specify otherwise. The response levels represent a range of values rather than a single value.

When response levels are determined by specific numeric values, and the DISCRETE option is specified, one level is created for each value. If the response variable has an associated format, then each formatted value is represented by a different response level. Formatted values are truncated to 16 characters.

The BLOCK, CHORO, and PRISM statements assign patterns to response levels. In CHORO and PRISM maps, response levels are shown as map areas. However, in BLOCK maps, response levels are shown as blocks. The default fill pattern for the response level is solid.

PATTERN statements can define the fill patterns and colors for both blocks and map areas. PATTERN definitions that define valid block patterns are applied to the blocks (response levels), and PATTERN definitions that define valid map patterns are applied to map areas.

See “PATTERN Statement” on page 169 for more information on fill pattern values and default pattern rotation.

About Identification Variables

For traditional map data sets and response data sets, *id-variable(s)* identify the map areas (for example, counties, states, or provinces) that make up the map. A *unit area* or *map area* is a group of observations with the same ID value. The GMAP procedure matches the value of the response variables for each map area in the response data set to the corresponding map area in the traditional map data set in order to create the output graphs.

With feature tables, the *geo-variable*, or \$GEOREF formatted variable containing the spatial information, is the identification variable. Each observation in a feature table has a unique \$GEOREF formatted variable value. When merging the feature table with the response data set using a SQL or DATA step statement, the identification variable can be any variable that is contained within both data sets. Once the merged data set has been created, the *geo-variable* is used in the PROC GMAP ID statement for the merged feature table and response data set. See Example 11 on page 1069 for more details.

Displaying Map Areas and Response Data

Whether the GMAP procedure draws a map area and whether it displays patterns for response values depends on the contents of the response data set and on the ALL

and MISSING options. The following table describes the conditions under which the procedure does or does not display map areas and response data.

<i>If the response data set...</i>	<i>And if...</i>	<i>Then the procedure...</i>
includes the map area	the map area has a response value	draws the map area and displays the response data
includes the map area	the map area has no response value (that is, the value is missing)	draws the map area but leaves it empty
includes the map area	the map area has no response value and the MISSING option is used in the map statement	draws the map area and displays a response level for the missing value
does not include the map area	the ALL option is used in the PROC GMAP statement	draws the map area but leaves it empty
does not include the map area	the ALL option is not used	does not draw the map area

Summary of Use

To use the GMAP procedure, you must do the following:

- 1 If necessary, issue a LIBNAME statement for the SAS data library that contains the map data set that you want to display.
- 2 If using a traditional map data set, determine what processing needs to be done to the map data set before it is displayed. Use the GPROJECT, GREDUCE, and GREMOVE procedures or a DATA step to perform the necessary processing.
- 3 Issue a LIBNAME statement for the SAS data set that contains the response data set, or use a DATA step to create a response data set.
- 4 If using a traditional map data set, use the PROC GMAP statement to identify the map data set as the MAP= value and response data set as the DATA= value.
- 5 If using a feature table, use PROC SORT to individually sort the feature table and response data set by a variable common to both data sets. Next, use SQL or the DATA step MERGE to merge the feature table with the response data set by using a variable common to both data sets. Use the combined data set as the DATA= value in the PROC GMAP statement (do not include MAP= in the PROC GMAP statement).
- 6 Use the ID statement to name the *id-variable(s)* or the *geo-variable*.
- 7 Use a BLOCK, CHORO, PRISM, or SURFACE statement to identify the response variable and generate the map.

Accessing SAS Maps Online

Visit SAS Maps Online to download data updates, sample SAS/GRAPH programs that use the map data sets delivered with SAS/GRAPH, and GIF images of maps. SAS Maps Online is located at the following URL:

<http://support.sas.com/rnd/datavisualization/maponline/html/>

Procedure Syntax

Requirements: One ID statement (see “ID Statement” on page 1009)and at least one CHORO (see “CHORO Statement” on page 1017), BLOCK (see “BLOCK Statement” on page 1009), PRISM (see “PRISM Statement” on page 1022), or SURFACE (see “SURFACE Statement” on page 1030)statement is required.

Global statements: FOOTNOTE, LEGEND, PATTERN, TITLE

Reminder: The GMAP procedure can include the BY (see “BY Statement” on page 141), FORMAT, LABEL, and WHERE statements as well as the “TITLE, FOOTNOTE, and NOTE Statements” on page 210.

Supports: RUN-group processing, Output Delivery System (ODS)

```
PROC GMAP <MAP=map-data-set>
  DATA=response-data-set | feature-table
  <ALL>
  <ANNOTATE=Annotate-data-set>
  <GOUT=<libref.>output-catalog>
  <IMAGEMAP=output-data-set>;
  ID id-variable(s) | geo-variable;
  BLOCK response-variable(s) </ option(s)>;
  CHORO response-variable(s) </ option(s)>;
  PRISM response-variable(s)</ option(s)>;
  SURFACE response-variable(s) </ option(s)>;
```

PROC GMAP Statement

Identifies the map data set and the response data set that contain the variables associated with the map. If the response data set and the feature table have been merged, the statement identifies the merged map data set. The statement optionally causes the procedure to display all map areas and specifies annotation and an output catalog.

Requirements: Both a map data set and a response data set are required. This can include a traditional map data set and response data set or a merged response data set and feature table.

```
PROC GMAP <MAP=map-data-set>
  DATA=response-data-set | feature-table
  <ALL>
  <ANNOTATE=Annotate-data-set>
  <GOUT=<libref.>output-catalog>
  <IMAGEMAP=output-data-set>;
```

Required Argument

DATA=response-data-set | feature-table

identifies the SAS data set that contains the response values or the response values and the spatial information that are evaluated and represented on the map. If a

response data set is specified, it must contain the same identification variable or variables as the map data set, along with the values of the response variable. If a feature table is specified, it must contain response data information and spatial geometry information. By default, the GMAP procedure uses the most recently created SAS data set.

See Also: “Concepts” on page 999, “SAS Data Sets” on page 29, and “About Feature Tables” on page 1001.

Options

PROC GMAP statement options affect all of the graphs that are produced by the procedure.

ALL

specifies that all maps generated by the procedure should include every map area from the map data set, even if the response data set does not include an observation for the map area.

When you use the ALL option and a BY statement in a RUN group, the maps generated for each BY group include every map area from the map data set.

See also: “Displaying Map Areas and Response Data” on page 1005, CEMPTY= and the MISSING options.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate all of the maps that are produced by the GMAP procedure. To annotate individual maps, use the ANNOTATE= option in the action statement.

Note: You can use the %MAPLABEL Macro to create the *Annotate-data-set*. See “%MAPLABEL Macro” on page 686 for more information. Δ

See also: Chapter 24, “Using Annotate Data Sets,” on page 587.

GOUT=<libref.>*output-catalog*

specifies the SAS catalog in which to save the graphics output that is produced by the GMAP procedure for later replay. You can use the GREPLAY procedure to view the graphs stored in the catalog. If you do not use the GOUT= option, catalog entries are written to the default catalog WORK.GSEG, which is erased at the end of your session.

Not supported by: Java, ActiveX

See also: “Storing Graphics Output in SAS Catalogs” on page 53.

IMAGEMAP=*output-data-set*

creates a temporary SAS data set that contains information about the graph that is replayed from the graphics catalog. The information in the image map data set includes the shape and coordinates of the elements in the graph, along with values that were associated with those elements in variables that were identified for that purpose in the HTML= and/or HTML_LEGEND= options. The image map data set can be used to generate an HTML image map in an HTML output file using the IMAGEMAP macro. The IMAGEMAP macro takes two arguments, the name of the image map data set and the name or fileref of an HTML output file, as shown in the following example:

```
%imagemap(imgmapds, myimgmap.html);
```

Not supported by: Java, ActiveX

MAP=map-data-set

names a SAS traditional map data set that contains the Cartesian coordinates for the boundary points of each map area. The traditional map data set must contain the same identification variable or variables as the response data set being used. This statement is required if a feature table is not being used.

See also: “About Traditional Data Sets” on page 999.

ID Statement

Identifies the variable or variables in the input data set(s) that define map areas.

Requirements: At least one *id-variable* or *geo-variable* is required.

ID *id-variable(s)* | *geo-variable*;

Required Arguments

id-variable(s)

identifies one or more variables in the map and response data sets that define map area. This argument is used only when map and response data sets are specified. If a feature table is specified, then specify the *geo-variable* argument.

Every variable that is listed in the ID statement must appear in both the map and response data sets. The variable identified by the *id-variable(s)* argument can be of type numeric or character and should have the same name, type, and length in both the response and map data sets.

See also: “About Identification Variables” on page 1005.

Featured in: Example 1 on page 1045, Example 3 on page 1049, and Example 4 on page 1052.

geo-variable

identifies the \$GEOREF formatted variable in the feature table containing the spatial geometry information for the map. The variable identified by the *geo-variable* argument must be of character type.

See also: “About Identification Variables” on page 1005.

Featured in: Example 11 on page 1069.

BLOCK Statement

Creates three-dimensional block maps on which levels of magnitude of the specified response variables are represented by blocks of varying height, pattern, and color.

Requirements: At least one response variable is required. The ID statement must be used in conjunction with the BLOCK statement.

Global statements: FOOTNOTE, LEGEND, PATTERN, TITLE

Description

The BLOCK statement specifies the variable or variables that contain the data that are represented on the map by blocks of varying height, pattern, and color. This statement automatically

- determines the midpoints ranges
- scales the blocks
- assigns patterns to the block faces and map areas. (See “About Block Maps and Patterns” on page 1016 for more information.)

You can use statement options to enhance the appearance of the map. For example, you can specify the width and shape of the blocks, the outline colors for the blocks and the map areas, and the angle of view. Other statement options control the response levels.

In addition, you can use global statements to modify the block patterns, the map patterns, and the legend, as well as to add titles and footnotes to the map. You can also use an Annotate data set to enhance the map.

BLOCK *response-variable(s) </option(s)>*;

The *option(s)* argument can be one or more of the following:

- appearance options:
 - ANNOTATE=*Annotate-data-set*
 - BLOCKSIZE=*size*
 - CBLKOUT=*block-outline-color* | SAME
 - CEMPTY=*empty-area-outline-color*
 - COUTLINE=*area-outline-color* | SAME
 - SHAPE=*3D-block-shape*
 - WOUTLINE=*block-outline-width*
 - XSIZE=*map-width <units>*
 - YSIZE=*map-height <units>*
 - XVIEW=*x*
 - YVIEW=*y*
 - ZVIEW=*z*
- mapping options:
 - AREA=*n* | *response-variable-name*
 - DISCRETE
 - LEVELS=*number-of-response-levels* | ALL
 - MIDPOINTS=*value-list* | OLD
 - MISSING
- legend options:
 - CTEXT=*text-color*
 - LEGEND=LEGEND<*1...99*>
 - NOLEGEND
- description options:
 - DESCRIPTION=*'entry-description'*
 - NAME=*'entry-name'*
- ODS options
 - HTML=*variable*

HTML_LEGEND=*variable*

Required Arguments

response-variable(s)

specifies one or more variables in the response data set, or in the merged response and feature table, that contain response values that are to be represented on the map. Each response variable produces a separate map. All variables must be in the input data set. Multiple response variables are separated with blanks. Blocks are not drawn for missing values for the response variable unless you use the MISSING option in the BLOCK statement.

See also: “About Response Variables” on page 1004.

Options

Options in a BLOCK statement affect all of the maps that are produced by that statement. You can specify as many options as you want and list them in any order.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate onto maps that are produced by the BLOCK statement. Annotate coordinate systems 1, 2, 7, and 8 are not valid with block maps.

Note: You can use the %MAPLABEL Macro to create the *Annotate-data-set*. See “%MAPLABEL Macro” on page 686 for more information. △

See also: Chapter 24, “Using Annotate Data Sets,” on page 587.

AREA=*n* | *response-variable-name*

specifies that a different map pattern be used for the surface of each map area or group of map areas on the map.

The value of *n* indicates which variable in the ID statement determines the groups that are distinguished by a surface pattern. By default, all map unit areas are drawn using the same surface fill pattern. If your ID statement has only one map area identification variable, then use AREA=1 to indicate that each map area surface uses a different pattern. If you have more than one variable in your ID statement, then use *n* to indicate the position of the variable that defines groups that will share a pattern. When you use the AREA= option, the map data set should be sorted in order of the variables in the ID statement.

A column name defined in either the MAP= or DATA= data sets may be indicated with the *column-name* value. If the column name exists in both the MAP= and DATA= data sets, the column in the map= data set will be used. When *column-name* is used, the areas will be colored based on the AREA= value. Duplicate AREA= values may have different patterns assigned

By default, using the AREA= option fills map areas by rotating the default hatch patterns through the colors list, beginning with the M2N0 pattern. The default outline color depends on the PATTERN statement:

- If you do not specify a PATTERN statement, the default outline color is black for the Java or ActiveX devices. Otherwise, the default outline color is the foreground color (the first color in the colors list).
- If you specify the PATTERN statement or the V6COMP graphics option, the default is COUTLINE=SAME.

You can specify pattern fills or colors or both with PATTERN statements that specify map/plot patterns. A separate PATTERN definition is needed for each specified area.

See also: “PATTERN Statement” on page 169.

Featured in: Example 3 on page 1049.

BLOCKSIZE=*size*

specifies the width of the blocks. The unit of *size* is the character cell width for the selected output device. By default, BLOCKSIZE=2.

Featured in: Example 5 on page 1054.

CBLKOUT=*block-outline-color* | SAME

outlines all blocks in the specified color. The SAME value specifies that the outline color of a block, a block segment, or a legend is the same as the interior pattern color.

The default outline color depends on the PATTERN statement:

- If no PATTERN statements are specified, then the default outline color is the foreground color (the first color in the colors list).
- If a PATTERN statement or the V6COMP graphics option is specified, then the default is CBLKOUT=SAME.

CBLKOUT= is not valid when SHAPE=CYLINDER.

Note: If you specify empty block patterns (VALUE=EMPTY in a PATTERN statement), you should not change the outline color from the default value, SAME, to a single color. Otherwise all the outlines will be one color and you will only be able to distinguish between empty areas by their size. Empty block patterns (VALUE=EMPTY in a PATTERN statement) are not supported by DEVICE=ACTIVEX. \triangle

Featured in: Example 1 on page 1045 and Example 3 on page 1049.

CEMPTY=*empty-area-outline-color*

outlines empty map areas in the specified color. This option affects only map areas that are empty. Empty map areas are generated in block maps only when a map area is omitted from the response data set and the ALL option is included in the PROC GMAP statement.

The default outline color is the same as the default COUTLINE= color.

Not supported by: Java

See also: ALL on page 1008 and “Displaying Map Areas and Response Data” on page 1005.

COUTLINE=*area-outline-color* | SAME

outlines non-empty map areas in the specified color. When COUTLINE=*area-outline-color* and DEVICE=JAVA or ACTIVEX, both empty and non-empty map areas are outlined. The SAME value specifies that the outline color of a map area is the same as the interior pattern color.

The default outline color depends on the PATTERN statement:

- If you do not specify a PATTERN statement, the default outline color is black for the Java or ActiveX devices. Otherwise, the default outline color is the foreground color (the first color in the colors list).
- If you specify the PATTERN statement or the V6COMP graphics option, the default is COUTLINE=SAME.

Note: If you specify empty map patterns (VALUE=EMPTY in a PATTERN statement), then you should not change the outline color from the default value

SAME. Otherwise all the outlines will be one color and you will not be able to distinguish between the empty areas. Empty block patterns (VALUE=EMPTY in a PATTERN statement) are not supported by DEVICE=ACTIVEX. △

Featured in: Example 3 on page 1049.

CTEXT=*text-color*

specifies a color for the text in the legend. If you omit the CTEXT= option, a color specification is searched for in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the default, the first color in the colors list.

The CTEXT= color specification is overridden if you also use the COLOR= suboption of a LABEL= or VALUE= option in a LEGEND definition that is assigned to the map legend. The COLOR= suboption determines the color of the legend label or the color of the legend value descriptions, respectively.

For the Java and ActiveX devices, the default color is black.

DESCRIPTION=*'entry-description'*

DES=*'entry-description'*

specifies a descriptive string up to 256 characters long, that appears in the description field of the catalog entry for the map. The description does not appear on the map. By default, the GMAP procedure assigns a description of the form BLOCK MAP OF *variable*, where *variable* is the name of the map variable.

Featured in: Example 5 on page 1054.

DISCRETE

treats a numeric, formatted response variable as though it has discrete values rather than continuous values. When you use DISCRETE, the response variable values are not grouped into ranges; instead, the GMAP procedure uses a separate response level (block height, pattern, and color) for each value of the formatted response variable.

The LEVELS= option is ignored when you use the DISCRETE option.

Use this option only if your numeric response variable is assigned a user-written format.

Note: If the data does not contain a value in a particular range of the format, that formatted range is not displayed in the legend. △

Featured in: Example 3 on page 1049 .

HTML=*variable*

identifies the variable in the input data set whose values create links in the HTML file created by the ODS HTML statement. These links are associated with an area of the map and point to the data or graph you wish to display when the user drills down on the area.

HTML_LEGEND=*variable*

identifies the variable in the input data set whose values create links in the HTML file created by the ODS HTML statement. These links are associated with a legend value and point to the data or graph you want to display in response to drill-down input from the user.

Not supported by: Java, ActiveX

LEGEND=LEGEND<1...99>

specifies the LEGEND statement to associate with the map. The LEGEND= option is ignored if the specified LEGEND definition is not currently in effect. In the GMAP procedure, the BLOCK statement produces a legend unless you use the NOLEGEND option. If you use the SHAPE= option in a LEGEND statement, only the value BAR is valid. Most of the LEGEND options described in “LEGEND Statement” on page

151 are supported by both Java and ActiveX. If a LEGEND option is not supported by Java or ActiveX, it is noted in the LEGEND option definition.

Not supported by: Java (partial), ActiveX (partial)

See also: “LEGEND Statement” on page 151.

Featured in: Example 2 on page 1047 and Example 5 on page 1054.

LEVELS=*number-of-response-levels* | ALL

number-of-response-levels specifies the number of response levels that are to be graphed when the response variables are continuous. Each level is assigned a different block height, pattern, and color combination.

If you specify DEVICE=ACTIVEX or DEVICE=ACTXIMG, and if you specify LEVELS=ALL, then a color ramp is used to assign each response value a continuous color scheme. The response values are assigned lighter and darker values of a color scheme to express lower and higher response values. When used with all other devices, the LEVELS=ALL and DISCRETE options behave exactly the same. For more information, see the DISCRETE option on page 1013.

If you do not use the LEVELS= option or the DISCRETE option, the GMAP procedure determines the number of response levels by using the formula $\text{FLOOR}(1 + 3.3 \log(n))$, where n is the number of unique identification variable values for map areas.

The LEVELS= option is ignored when you use the DISCRETE or MIDPOINTS=*value-list* option. When MIDPOINTS=OLD is used with the LEVELS= option, default midpoints are generated using the Nelder algorithm (*Applied Statistics* 25:94–7, 1976).

Note: LEVELS=ALL is a . \triangle

MIDPOINTS=*value-list* | OLD

specifies the response levels for the range of response values that are represented by each level (block height, pattern, and color combination).

For numeric response variables, *value-list* is either an explicit list of values or a starting and an ending value with an interval increment, or a combination of both forms:

n <...*n*>

n TO *n* <BY *increment*>

n <...*n* > TO *n* <BY *increment*> <*n*<...*n*>>

By default, the increment value is 1. You can specify discrete numeric values in any order. In all forms, *n* can be separated by blanks or commas. For example,

```
midpoints=(2 4 6)
```

```
midpoints=(2,4,6)
```

```
midpoints=(2 to 10 by 2)
```

If a numeric variable has an associated format, the specified values must be the *unformatted* values. With numeric response values, DEVICE=JAVA only uses midpoints that fall in the range of the data being used. Thus, if your data ranged from 30-80, but midpoints were specified at 25, 50, 75, and 100, only 50 and 75 are used.

For character response variables, *value-list* is a list of unique character values enclosed in quotes and separated by blanks:

```
'value-1' <...'value-n'>
```

```
midpoints='Midwest' 'Northeast' 'Northwest'
```

Specify the values in any order. If a character variable has an associated format, the specified values must be the *formatted* values. Character response values specified with the MIDPOINTS= option are not supported by DEVICE=JAVA.

You can selectively exclude some response variable values from the map, as shown here:

```
midpoints='Midwest'
```

Only those observations for which the response variable exactly matches one of the values listed in the MIDPOINTS= option are shown on the map. As a result, observations may be excluded inadvertently if values in the list are misspelled or if the case does not match exactly.

Specifying MIDPOINTS=OLD generates default midpoints using the Nelder algorithm (*Applied Statistics* 25:94–7, 1976). Specifying GOPTIONS V6COMP serves the same purpose.

Not supported by: Java (partial)

Featured in: Example 5 on page 1054.

MISSING

accepts a missing value as a valid level for the response variable.

See also: “Displaying Map Areas and Response Data” on page 1005.

NAME=*entry-name*

If you specify DEVICE=ACTXIMG or DEVICE=JAVAIMG, then the name that you specify will be used for the client image output even if the file exists. For all other devices, if the name duplicates an existing entry name, SAS/GRAPH specifies the name of the catalog entry for the map. The maximum length for *entry-name* is 8 characters. The default name is GMAP. If the specified name duplicates the name of an existing entry, SAS/GRAPH appends a number to the duplicate name to create a unique name, for example, GMAP1.

Featured in: Example 5 on page 1054.

NOLEGEND

suppresses the legend.

SHAPE=*3D-block-shape*

specifies the shape of the blocks. Use this option to enhance the look of the block shape, or to specify a different shape. When using the SHAPE= option, only solid fill patterns will be used. The value of *3D-block-shape* can be one of the following:

- BLOCK | B
- CYLINDER | C
- HEXAGON | H
- PRISM | P
- STAR | S

The CBLKOUT= option is not valid when SHAPE=CYLINDER.

WOULINE=*block-outline-width*

specifies the width, in pixels, of the outline for all outlined blocks and for the outline of the map areas.

Not supported by: Java

XSIZE=*map-width <units>*

YSIZE=*map-height <units>*

specify the physical dimensions of the map to be drawn. By default, the map uses the entire procedure output area.

Valid *units* are CELLS (character cells), CM (centimeters), IN (inches), or PCT (percentage of the graphics output area). The default unit is CELLS.

If you specify values for *map-width* or *map-height* that are greater than the dimensions of the procedure output area, the map is drawn using the default size.

Not supported by: Java, ActiveX

XVIEW=*x*
YVIEW=*y*
ZVIEW=*z*

specify coordinates of the viewing position in the reference coordinate system. In this system, the four corners of the map lie on the X-Y plane at coordinates (0,0,0), (0,1,0), (1,1,0), and (1,0,0). No axes are actually drawn on the maps that are produced by PROC GMAP. Your viewing position cannot coincide with the viewing reference point at coordinates (0.5,0.5,0), the center of the map. The value for *z* cannot be negative.

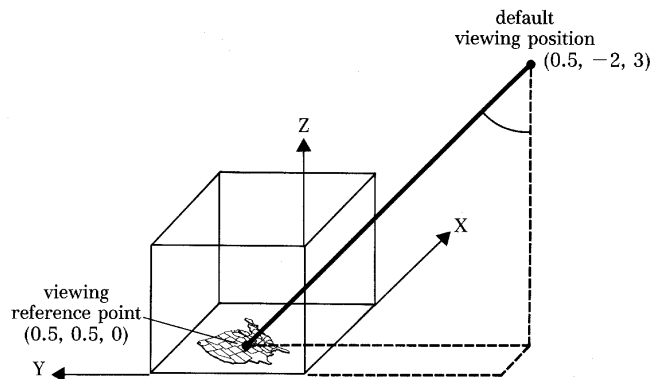
If you omit the XVIEW=, YVIEW=, and ZVIEW= options, the default coordinates are (0.5, -2, 3). This viewing position is well above and to the south of the center of the map. You can specify one, two, or all three of the view coordinates; any that you do not specify are assigned the default values. While you can use the XVIEW= and YVIEW= options with DEVICE=JAVA, ZVIEW= can not be used with DEVICE=JAVA.

Not supported by: Java (partial)

Featured in: Example 2 on page 1047.

Figure 35.5 on page 1016 shows the position of the viewing reference point, as well as the default viewing position.

Figure 35.5 Viewing Position and Viewing Reference Point



About Block Maps and Patterns

Block maps are different from other maps in that they display two different types of areas that use patterns:

- the blocks themselves, which represent the response levels
- the map areas from which the blocks rise.

By default, the blocks use solid pattern fills and the map areas use a hatch pattern of slanting lines. The map areas in block maps are the *only* map areas that by default do not use solid fills. The map areas and their outlines use the first color in the colors list regardless of whether the list is the device's default colors list or one specified with the COLORS= option in a GOPTIONS statement.

The BLOCK statement has the following options that explicitly control the outline colors used by the blocks and the map areas.

- CBLKOUT=
- CEMPTY=

□ COUTLINE=

In addition the AREA= option controls how the map areas are patterned.

When you use PATTERN statements to define the patterns for the map, you must be sure to specify the correct type of pattern for the area. The blocks use bar/block patterns and the map areas use map/plot patterns. See “PATTERN Statement” on page 169 for more information on specifying patterns.

Note: If you specify only one PATTERN statement and include only the COLOR= option, that color will be used for both the blocks and the map areas. For example, this statement makes the blocks solid blue and the map areas blue hatch. △

```
pattern1 color=blue;
```

Note: Empty block patterns (VALUE=EMPTY in a PATTERN statement) are not supported by DEVICE=ACTIVEEX. △

CHORO Statement

Creates two-dimensional maps in which values of the specified response variables are represented by varying patterns and colors.

Requirements: At least one response variable is required. The ID statement must be used in conjunction with the CHORO statement

Global statements: FOOTNOTE, LEGEND, PATTERN, TITLE

Description

The CHORO statement specifies the variable or variables that contain the data represented on the map by patterns that fill the map areas. This statement automatically

- determines the midpoints
- assigns patterns to the map areas.

You can use statement options to enhance the appearance of the map, for example, by selecting the colors and patterns that fill the map areas. Other statement options control the selection of ranges for the response variable.

In addition, you can use global statements to modify the map area patterns and legend, as well as add titles and footnotes to the map. You can also use an Annotate data set to enhance the map.

CHORO *response-variable(s) </option(s)>;*

option(s) can be one or more from any of the following categories:

- appearance options:
 - ANNOTATE=*Annotate-data-set*
 - CEMPTY=*empty-area-outline-color*
 - COUTLINE=*area-outline-color* | SAME
 - WOUTLINE=*area-outline-width*
 - XSIZE=*map-width<units>*

- Ysize=*map-height* <units>
- Xview=*x*
- Yview=*y*
- Zview=*z*
- mapping options:
 - DISCRETE
 - LEVELS=*number-of-response-levels* | ALL
 - MIDPOINTS=*value-list* | OLD
 - MISSING
- legend options:
 - Ctext=*text-color*
 - LEGEND=LEGEND<1...99>
 - NOLEGEND
- description options:
 - DESCRIPTION=*'entry-description'*
 - NAME=*'entry-name'*
- ODS options
 - HTML=*variable*
 - HTML_LEGEND=*variable*

Required Arguments

response-variable(s)

specifies one or more variables in the response data set, or in the merged response and feature table, that contain response values that are to be represented on the map. Each response variable produces a separate map. All variables must be in the input data set. Multiple response variables are separated with blanks.

Missing values for the response variable are not considered valid response values unless you use the MISSING option on the CHORO statement.

Response variables can be either numeric or character in type. Numeric response variables are normally grouped into ranges, or response levels, as determined by the MIDPOINTS= or LEVELS= options. Each response level is assigned a different combination of pattern and color. Character response variables are assigned unique response levels, as are numeric variables when the DISCRETE option is specified.

See also: “About Response Variables” on page 1004.

Options

Options in a CHORO statement affect all graphs that are produced by that statement. You can specify as many options as you want and list them in any order.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate onto maps that are produced by the CHORO statement.

Note: You can use the %MAPLABEL Macro to create the *Annotate-data-set*. See “%MAPLABEL Macro” on page 686 for more information. Δ

See also: Chapter 24, “Using Annotate Data Sets,” on page 587.

Featured in: Example 6 on page 1061.

EMPTY=empty-area-outline-color

outlines empty map areas in the specified color. This option affects only the empty map areas, which are generated in choro maps either

- when there is no response value for a map area and the MISSING option is not used, or
- when a map area is omitted from the response data set and the ALL option is included in the PROC GMAP statement.

The default outline color is the same as the default COUTLINE= color.

Not supported by: Java

See also: ALL on page 1008 and “Displaying Map Areas and Response Data” on page 1005.

COUTLINE=area-outline-color | SAME

outlines non-empty map areas in the specified color. When COUTLINE=area-outline-color and DEVICE=JAVA or ACTIVEX, both empty and non-empty map areas are outlined. The value SAME specifies that the outline color of a map area is the same as the interior pattern color.

The default outline color depends on the PATTERN statement:

- If no PATTERN statement is specified, the default outline color is the foreground color (the first color in the colors list).
- If a PATTERN statement or the V6COMP graphics option is specified, the default is COUTLINE=SAME.

Note: If you specify empty map patterns (VALUE=EMPTY in a PATTERN statement), then you should not change the outline color from the default value SAME to a single color. Otherwise all the outlines will be one color and you will not be able to distinguish between the empty areas. △

Featured in: Example 4 on page 1052.

CTEXT=text-color

specifies a color for the text in the legend. If you omit the CTEXT= option, a color specification is searched for in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the default, the first color in the colors list.

The CTEXT= color specification is overridden if you also use the COLOR= suboption of a LABEL= or VALUE= option in a LEGEND definition that is assigned to the map legend. The COLOR= suboption determines the color of the legend label or the color of the legend value descriptions, respectively.

For the Java and ActiveX devices, the default color is black.

DESCRIPTION='entry-description'

DES='entry-description'

specifies a descriptive string up to 256 characters long that appears in the description field of the catalog entry for the map. The description does not appear on the map. By default, the GMAP procedure assigns a description of the form CHOROPLETH MAP OF *map_variable*.

Featured in: Example 5 on page 1054.

DISCRETE

generates a unique response value (pattern and color combination) for each numeric response variable.

The LEVELS= option is ignored when you use the DISCRETE option.

Be sure to use this option if your numeric response variable is assigned a user-written format.

Note: If the data does not contain a value in a particular range of the format, that formatted range is not displayed in the legend. Δ

Featured in: Example 5 on page 1054 and Example 11 on page 1069.

HTML=variable

identifies the variable in the input data set whose values create links in the HTML file created by the ODS HTML statement. These links are associated with an area of the map and point to the data or graph you wish to display when you drill down on the area.

Featured in: Example 5 on page 1054.

HTML_LEGEND=variable

identifies the variable in the input data set whose values create links in the HTML file created by the ODS HTML statement. These links are associated with a legend value and point to the data or graph you wish to display when you drill down on the value.

Not supported by: Java, ActiveX

Featured in: Example 5 on page 1054.

LEGEND=LEGEND<1...99>

assigns the specified LEGEND statement that is to be applied to the map. The LEGEND= option is ignored if the specified LEGEND definition is not currently in effect. In the GMAP procedure, the CHORO statement produces a legend by default unless you specify the NOLEGEND option. If you use the SHAPE= option in a LEGEND statement, then only the value BAR is valid. Most of the LEGEND options described in “LEGEND Statement” on page 151 are supported by both Java and ActiveX. If a LEGEND option is not supported by Java or ActiveX, it is noted in the LEGEND option definition.

Not supported by: Java (partial), ActiveX (partial)

See also: “LEGEND Statement” on page 151.

Featured in: Example 2 on page 1047.

LEVELS=number-of-response-levels | ALL

specifies the number of response levels to be graphed for numeric response variables, when the DISCRETE or MIDPOINTS= options are not specified. Each response level is assigned a different combination of color and fill pattern.

If you specify DEVICE=ACTIVEX or DEVICE=ACTXIMG, and if you specify LEVELS=ALL, then a color ramp is used to assign each response value a continuous color scheme. The response values are assigned lighter and darker values of a color scheme to express lower and higher response values. When used with all other devices, the LEVELS=ALL and DISCRETE options behave exactly the same. For more information, see the DISCRETE option on page 1019.

If the LEVELS= option is not used, the GMAP procedure determines the number of response levels by using the formula $\text{FLOOR}(1+3.3 \log(n))$, where n is the number of unique map area identification variable values.

The LEVELS= option is ignored when you use the DISCRETE or MIDPOINTS=*value-list* option. When MIDPOINTS=OLD is used with the LEVELS= option, default midpoints are generated using the Nelder algorithm (*Applied Statistics* 25:94–7, 1976).

Note: LEVELS=ALL is a . Δ

Featured in: Example 2 on page 1047.

MIDPOINTS=value-list | OLD

specifies the response levels for the range of response values that are represented by each level (pattern and color combination).

For numeric response variables, the *value-list* argument is either an explicit list of values, a starting and an ending value with an interval increment, or a combination of both forms:

```
n <...n>
n TO n <BY increment >
n <...n> TO n <BY increment > n <...n>
```

By default the increment value is 1. You can specify discrete numeric values in any order. In all forms, *n* can be separated by blanks or commas. For example,

```
midpoints=(2 4 6)
midpoints=(2,4,6)
midpoints=(2 to 10 by 2)
```

If a numeric variable has an associated format, the specified values must be the *unformatted* values. With numeric response values, DEVICE=JAVA only uses midpoints that fall in the range of the data being used. Thus, if your data ranged from 30-80, but midpoints were specified at 25, 50, 75, and 100, only 50 and 75 are used.

For character response variables, *value-list* is a list of unique character values enclosed in quotation marks and separated by blanks:

```
'value-1' <...'value-n'>
```

The values are character strings enclosed in single quotation marks and separated by blanks. For example,

```
midpoints='Midwest' 'Northeast' 'Northwest'
```

Specify the values in any order. If a character variable has an associated format, the specified values must be the *formatted* values. Character response values specified with the MIDPOINTS= option are not supported by DEVICE=JAVA.

You can selectively exclude some response variable values from the map, as shown here:

```
midpoints='Midwest'
```

The only observations that are shown on the map are those observations for which the response variable exactly matches one of the values that are listed in the MIDPOINTS= option. As a result, observations may be excluded inadvertently if values in the list are misspelled or if the case does not match exactly.

Specifying MIDPOINTS=OLD generates default midpoints using the Nelder algorithm (*Applied Statistics* 25:94–7, 1976). Specifying GOPTIONS V6COMP serves the same purpose.

Not supported by: Java (partial)

Featured in: Example 8 on page 1065.

MISSING

accepts a missing value as a valid level for the response variable.

See also: “Displaying Map Areas and Response Data” on page 1005.

NAME=entry-name'

If you specify DEVICE=ACTXIMG or DEVICE=JAVAIMG, then the name that you specify will be used for the client image output even if the file exists. For all other devices, if the name duplicates an existing entry name, SAS/GRAPH specifies the name of the catalog entry for the map. The maximum length of the *entry-name* value is eight characters. The default name is GMAP. If the specified name duplicates the

name of an existing entry, SAS/GRAPH appends a number to the duplicate name to create a unique entry, for example, GMAP1.

Featured in: Example 5 on page 1054.

NOLEGEND

suppresses the legend.

Featured in: Example 6 on page 1061.

WOUTLINE=area-outline-width

specifies the width of all map area outlines, in pixels.

Not supported by: Java

XSIZE=map-width <units>

YSIZE=map-height <units>

specify the physical dimensions of the map. By default, the map uses the entire procedure output area.

Valid *units* are CELLS (character cells), CM (centimeters), IN (inches), or PCT (percentage of the graphics output area). The default unit is CELLS.

If you specify values for *units* that are greater than the dimensions of the procedure output area, the map is drawn using the default size.

If you specify either the XSIZE= or YSIZE= option without specifying the other option, the GMAP procedure scales the dimension for the option that was not specified to retain the original shape of the map.

Not supported by: Java, ActiveX

XVIEW=x

YVIEW=y

ZVIEW=z

specify coordinates of the viewing position in the reference coordinate system. In this system, the four corners of the map lie on the X-Y plane at coordinates (0,0,0), (0,1,0), (1,1,0), and (1,0,0). No axes are actually drawn on the maps that are produced by PROC GMAP. Your viewing position cannot coincide with the viewing reference point at coordinates (0.5,0.5,0), the center of the map. The value for *z* cannot be negative.

If you omit the XVIEW=, YVIEW=, and ZVIEW= options, the default coordinates are (0.5, -2, 3). This viewing position is well above and to the south of the center of the map. You can specify one, two, or all three of the view coordinates; any that you do not specify are assigned the default values. While you can use the XVIEW= and YVIEW= options with DEVICE=JAVA, ZVIEW= can not be used with DEVICE=JAVA.

Figure 35.5 on page 1016 shows the position of the viewing reference point, as well as the default viewing position.

Not supported by: Java (partial)

PRISM Statement

Creates three-dimensional prism maps in which levels of magnitude of the specified response variables are represented by polyhedrons (raised polygons) of varying height, pattern, and color.

Requirements: At least one response variable is required. You must use the ID statement in conjunction with the PRISM statement.

Global statements: FOOTNOTE, LEGEND, PATTERN, TITLE

Description

The PRISM statement specifies the variable or variables that contain the data that are represented on the map by raised map areas. This statement automatically

- determines the midpoints ranges or midpoints
- assigns patterns to the map areas.

You can use statement options to control the ranges of the response values, specify the angle of view, and enhance the appearance of the map.

In addition, you can use global statements to modify the map area patterns and the legend, as well as add titles and footnotes to the map. You can also use an Annotate data set to enhance the map.

Note: For maps that contain intersecting polygons or polygons within polygons, extremely complicated maps, or maps that contain line segments that cross, use the GREduce procedure to reduce and simplify the map if necessary. △

PRISM *response-variable(s) </ option(s)>*;

The *option(s)* can be one or more options from any or all of the following categories:

- appearance options:
 - ANNOTATE=*Annotate-data-set*
 - CEMPTY=*empty-area-outline-color*
 - COUTLINE=*area-outline-color* | SAME
 - WOUTLINE=*area-outline-width*
 - XLIGHT=*x*
 - YLIGHT=*y*
 - XSIZE=*map-width <units>*
 - YSIZE=*map-height <units>*
 - XVIEW=*x*
 - YVIEW=*y*
 - ZVIEW=*x*
- mapping options:
 - AREA=*n* | *response-variable-name*
 - DISCRETE
 - LEVELS=*number-of-response-levels* | ALL
 - MIDPOINTS=*value-list* | OLD
 - MISSING
- legend options:
 - CTEXT=*text-color*
 - LEGEND=LEGEND<1...99>
 - NOLEGEND
- description options:
 - DESCRIPTION=*'entry-description'*
 - NAME=*'entry-name'*
- ODS options
 - HTML=*variable*
 - HTML_LEGEND=*variable*

Required Arguments

response-variable(s)

specifies one or more variables in the response data set, or in the merged response and feature table, that contain response values that are to be represented on the map. Each response variable produces a separate map. All variables must be in the input data set. Multiple response variables are separated with blanks.

Missing values for the response variable are not considered valid unless you use the MISSING option.

Response variables can be either numeric or character. By default, and as determined by the LEVELS= or MIDPOINTS= values, numeric response variables are grouped into ranges, or response levels. Each response level is assigned a different prism height and a different pattern and color combination.

Character variables and numeric variables (when you use the DISCRETE option) have a unique response level for each unique response variable value.

See also: “About Response Variables” on page 1004.

Options

Options in a PRISM statement affect all of the graphs that are produced by that statement. You can specify as many options as you want and list them in any order.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate onto the maps that are produced by the PRISM statement. Annotate coordinate systems 1, 2, 7, and 8 are not valid with Prism maps.

Note: You can use the %MAPLABEL Macro to create the *Annotate-data-set*. See “%MAPLABEL Macro” on page 686 for more information. \triangle

See also: Chapter 24, “Using Annotate Data Sets,” on page 587.

AREA=*n* | *response-variable-name*

specifies that a different map pattern be used for the surface of each map area or group of map areas on the map.

The value of *n* indicates which variable in the ID statement determines the groups that are distinguished by a surface pattern. By default, all map unit areas are drawn using the same surface fill pattern. If your ID statement has only one map area identification variable, then use AREA=1 to indicate that each map area surface uses a different pattern. If you have more than one variable in your ID statement, then use *n* to indicate the position of the variable that defines groups that will share a pattern. When you use the AREA= option, the map data set should be sorted in order of the variables in the ID statement.

A column name defined in either the MAP= or DATA= data sets may be indicated with the *response-variable-name* value. If the column name exists in both the MAP= and DATA= data sets, then the column in the MAP= data set will be used. When *column-name* is used, the areas will be colored based on the AREA= value. Duplicate AREA= values may have different patterns assigned.

By default, the AREA= option fills map areas by rotating the default hatch patterns through the colors list, beginning with the M2N0 pattern. The default outline color depends on the PATTERN statement:

- If no PATTERN statement is specified, the default outline color is the foreground color (the first color in the colors list).
- If a PATTERN statement or the V6COMP graphics option is specified, the default is COUTLINE=SAME.

You can specify pattern fills or colors or both with PATTERN statements that specify map/plot patterns. A separate PATTERN definition is needed for each specified area.

See also: “PATTERN Statement” on page 169.

Featured in: Example 3 on page 1049.

EMPTY=empty-area-outline-color

outlines empty map areas in the specified color. Empty map areas are generated in prism maps either

- when there is no response value for a map area and the MISSING option is not used, or
- when a map area is omitted from the response data set and the ALL option is included in the PROC GMAP statement.

The default outline color is the same as the default COUTLINE= color.

Not supported by: Java

See also: ALL on page 1008 and “Displaying Map Areas and Response Data” on page 1005.

COUTLINE=area-outline-color | SAME

outlines non-empty map areas in the specified color. When COUTLINE=area-outline-color and DEVICE=JAVA or ACTIVEX, both empty and non-empty map areas are outlined. SAME specifies that the outline color of a map area is the same as the interior pattern color.

The default outline color depends on the PATTERN statement:

- If no PATTERN statement is specified, the default outline color is the foreground color (the first color in the colors list).
- If a PATTERN statement or the V6COMP graphics option is specified, the default is COUTLINE=SAME.

Note: If you specify empty map patterns (VALUE=EMPTY in a PATTERN statement), you should not change the outline color from the default value SAME to a single color. Otherwise, all the outlines will be one color and you will not be able to distinguish between the empty areas. Empty block patterns (VALUE=EMPTY in a PATTERN statement) are not supported by DEVICE=ACTIVEX. △

Featured in: Example 7 on page 1063.

CTEXT=text-color

specifies a color for the text in the legend. If you omit the CTEXT= option, a color specification is searched for in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the default, the first color in the colors list.

The CTEXT= color specification is overridden if you also use the COLOR= suboption of a LABEL= or VALUE= option in a LEGEND definition assigned to the map legend. The COLOR= suboption determines the color of the legend label or the color of the legend value descriptions, respectively.

For the Java and ActiveX devices, the default color is black.

DESCRIPTION='entry-description'

DES='entry-description'

specifies the description of the catalog entry for the map. The maximum length for entry-description is 256 characters. By default, the GMAP procedure assigns a description of the form PRISM MAP OF map_variable.

DISCRETE

generates a separate response level (prism height, color, and surface pattern) for each different value of the formatted response variable. The LEVELS= option is ignored when you use the DISCRETE option.

Use this option if your numeric response variable is assigned a user-written format.

Note: If the data does not contain a value in a particular range of the format, that formatted range is not displayed in the legend. \triangle

Featured in: Example 5 on page 1054 and Example 11 on page 1069 (with the CHORO statements).

HTML=variable

identifies the variable in the input data set whose values create links in the HTML file created by the ODS HTML statement. These links are associated with an area of the map and point to the data or graph that are displayed in response to drill-down input.

HTML_LEGEND=variable

identifies the variable in the input data set whose values create links in the HTML file created by the ODS HTML statement. These links are associated with legend values and point to the data or graphs that are displayed in response to drill-down input.

Not supported by: Java, ActiveX

LEGEND=LEGEND<1...99>

specifies the LEGEND definition to associate with the map. LEGEND= is ignored if the specified LEGEND definition is not currently in effect. In the GMAP procedure, the PRISM statement produces a legend unless you use the NOLEGEND option. If you use the SHAPE= option in a LEGEND statement, only the value BAR is valid. Most of the LEGEND options described in “LEGEND Statement” on page 151 are supported by both Java and ActiveX. If a LEGEND option is not supported by Java or ActiveX, it is noted in the LEGEND option definition.

Not supported by: Java (partial), ActiveX (partial)

See also: “LEGEND Statement” on page 151

Featured in: Example 8 on page 1065.

LEVELS=number-of-response-levels | ALL

specifies the number of response levels to be graphed when the response variables are numeric and the DISCRETE and MIDPOINTS= options are not specified. Each response level is assigned a different prism height, surface pattern, and color combination.

If you specify DEVICE=ACTIVEX or DEVICE=ACTXIMG, and if you specify LEVELS=ALL, then a color ramp is used to assign each response value a continuous color scheme. The response values are assigned lighter and darker values of a color scheme to express lower and higher response values. When used with all other devices, the LEVELS=ALL and DISCRETE options behave exactly the same. For more information, see the DISCRETE option on page 1026.

If neither the LEVELS= option nor the DISCRETE option is used, then the GMAP procedure determines the number of response levels by using the formula $\text{FLOOR}(1+3.3 \log(n))$, where n is the number of unique map area identification variable values.

The LEVELS= option is ignored when you use the DISCRETE or MIDPOINTS=*value-list* option. When MIDPOINTS=OLD is used with the LEVELS= option, default midpoints are generated using the Nelder algorithm (*Applied Statistics* 25:94–7, 1976).

Note: LEVELS=ALL is a . \triangle

Featured in: Example 2 on page 1047.

MIDPOINTS=*value-list* | OLD

specifies the response levels for the range of response values that are represented by each level (prism height, pattern, and color combination).

For numeric response variables, *value-list* is either an explicit list of values, or a starting and an ending value with an interval increment, or a combination of both forms:

```
n <...n>
n TO n <BY increment>
n <...n> TO n <BY increment > n <...n>
```

By default the increment value is 1. You can specify discrete numeric values in any order. In all forms, *n* can be separated by blanks or commas. For example,

```
midpoints=(2 4 6)
midpoints=(2,4,6)
midpoints=(2 to 10 by 2)
```

If a numeric variable has an associated format, the specified values must be the *unformatted* values. With numeric response values, DEVICE=JAVA only uses midpoints that fall in the range of the data being used. Thus, if your data ranged from 30-80, but midpoints were specified at 25, 50, 75, and 100, only 50 and 75 are used.

For character response variables, *value-list* has this form:

```
'value-1' <...'value-n'>
```

The values are character strings enclosed in single quotation marks and separated by blanks. For example,

```
midpoints='Midwest' 'Northeast' 'Northwest'
```

Specify the values in any order. If a character variable has an associated format, the specified values must be the *formatted* values. Character response values specified with the MIDPOINTS= option are not supported by DEVICE=JAVA.

You can selectively exclude some response variable values from the map, as shown here:

```
midpoints='Midwest'
```

Only those observations for which the response variable exactly matches one of the values listed in the MIDPOINTS= option are shown on the map. As a result, observations may be inadvertently excluded if values in the list are misspelled or if the case does not match exactly.

Specifying MIDPOINTS=OLD generates default midpoints using the Nelder algorithm (*Applied Statistics* 25:94–7, 1976). Specifying GOPTIONS V6COMP serves the same purpose.

Not supported by: Java (partial)

Featured in: Example 8 on page 1065.

MISSING

accepts a missing value as a valid level for the response variable.

See also: “Displaying Map Areas and Response Data” on page 1005.

NAME=*'entry-name'*

When you specify DEVICE=ACTXIMG or DEVICE=JAVAIMG, specifies the name that will be used for the client image output even in the file exists. For all other devices, if the name duplicates an existing entry name, then SAS/GRAPH specifies the name of the catalog entry for the map. The maximum length for *entry-name* is eight characters. The default name is GMAP. If the specified name duplicates an

existing name, then SAS/GRAPH software appends a number to the duplicate name to create a unique entry, for example, GMAP1.

NOLEGEND

suppresses the legend.

WOULINE=*area-outline-width*

specifies the width, in pixels, of all map area outlines.

Not supported by: Java, ActiveX

XLIGHT=*x***YLIGHT=*y***

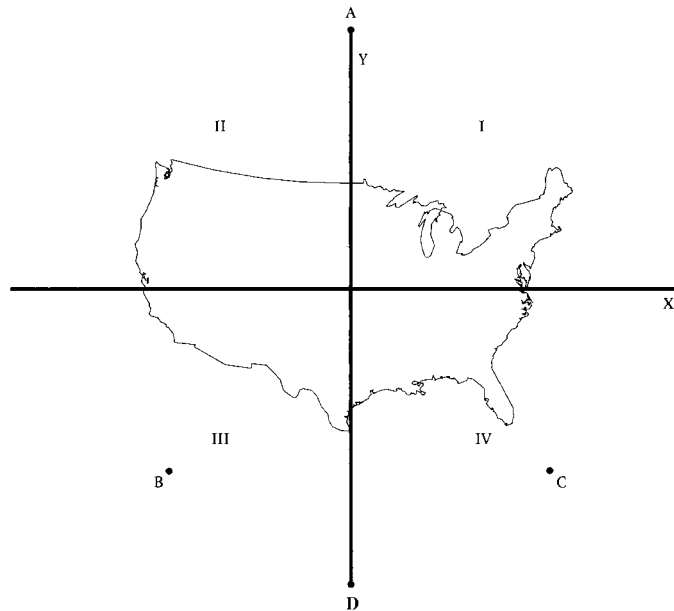
specify the coordinates of the imaginary light source in the map coordinate system. The position of the light source affects the way the sides of the map polygons are shaded. Although you can specify any point for the light source using the XLIGHT= and YLIGHT= options, the light source is actually placed in one of only four positions.

Table 35.3 on page 1028 shows how the point you specify is positioned.

Table 35.3 Light Source Coordinates

Specified Light Source	Light Source Position
in quadrants I or II, or on the X or +Y axis	behind the map (point A), and all side polygons are shadowed
on or within approximately 10 degrees of the Y axis	the viewing position (point D), and none of the side polygons are shadowed
in quadrant III (except within 10 degrees of the Y axis)	to the left of the map (point B), and the right-facing sides of polygons are shadowed
in quadrant IV (except within 10 degrees of the Y axis)	to the right of the map (point C), and the left-facing side polygons are shadowed

Figure 35.6 on page 1029 illustrates the light source positions. Assume that your viewing position, selected by the XVIEW=, YVIEW=, and ZVIEW= options, is point D.

Figure 35.6 Coordinates of Imagined Light Source in a Map Coordinate System

By default, the light source position is the same as the viewing position specified by the `XVIEW=`, `YVIEW=`, and `ZVIEW=` options. The light source position cannot coincide with the viewing reference point (0.5,0.5), which corresponds with the position directly above the center of the map.

Not supported by: Java, ActiveX

See also: `XVIEW=` on page 1029.

Featured in: Example 8 on page 1065.

XSIZE=*map-width* <units>

YSIZE=*map-height* <units>

specify the dimensions of the map that you are drawing. By default, the map uses the entire procedure output area.

Valid *units* are CELLS (character cells), CM (centimeters), IN (inches), or PCT (percentage of the graphics output area). The default unit is CELLS.

If you specify values for *map-width* and *map height* that are greater than the dimensions of the procedure output area, the map is drawn using the default size. If you specify one value and not the other, the dimension is adjusted to maintain the correct aspect ratio.

Not supported by: Java, ActiveX

XVIEW=*x*

YVIEW=*y*

ZVIEW=*z*

specify the viewing position coordinates for the map. In this system, the four corners of the map lie on the X-Y plane at coordinates (0, 0, 0), (0, 1, 0), (1, 1, 0), and (1, 0, 0).

The viewing position cannot coincide with the viewing reference point at coordinates (0.5, 0.5, 0).

The value for *z* cannot be negative.

If you omit the `XVIEW=`, `YVIEW=`, and `ZVIEW=` options, the default coordinates are (0.5, -2, 3). This viewing position is well above and to the south of the center of the map. One, two, or all three view coordinates can be specified; any that are not specified are assigned the default values.

Figure 35.5 on page 1016 shows the position of the viewing reference point, as well as the default viewing position.

To ensure that the polygon edges are distinguishable, the angle from vertical must be less than or equal to 45 degrees. If you specify a ZVIEW= value such that this condition cannot be satisfied (that is, a very small value), PROC GMAP increases the ZVIEW= value automatically so that the angle is 45 degrees or less. While you can use the XVIEW= and YVIEW= options with DEVICE=JAVA, ZVIEW= can not be used with DEVICE=JAVA.

Not supported by: Java (partial)

Featured in: Example 8 on page 1065.

SURFACE Statement

Creates three-dimensional surface maps in which levels of magnitude of the specified response variables are represented by spikes of varying height.

Requirements: At least one response variable is required and must be numeric. The ID statement must be used in conjunction with the SURFACE statement.

Global statements: FOOTNOTE, TITLE

Not supported by: Java, ActiveX

Description

The SURFACE statement specifies the variable or variables that contain the data that are represented on the map by raised map areas. This statement automatically determines the midpoints. You can use statement options to control spike proportions, specify the angle of view, and modify the general appearance of the map. For example, you can select the color and number of lines for the representation of the surface area. You can control the selection of spike heights and base widths.

In addition, you can use global statements to add titles and footnotes to the map. You can also enhance the map with an Annotate data set.

SURFACE *response-variable(s)* </option(s)>;

option(s) can be one or more of the following:

- appearance options:
 - ANNOTATE=*Annotate-data-set*
 - CBODY=*surface-map-color*
 - CONSTANT=*n*
 - NLINES=*number-of-lines*
 - ROTATE=*degrees*
 - TILT=*degrees*
 - XSIZE=*map-width* <*units*>
 - YSIZE=*map-height* <*units*>
- description options:
 - DESCRIPTION=*'entry-description'*
 - NAME=*'entry-name'*

Required Arguments

response-variable(s)

specifies one or more variables in the response data set, or in the merged response and feature table, that contain response values that are to be represented on the map. The *response-variable* must be numeric and must contain only positive values. Each response variable produces a separate map. All variables must be in the input data set. Multiple response variables are separated with blanks.

The GMAP procedure scales response variables for presentation on the map. The height of the spikes on the map correspond to the relative value of the response variable, not to the actual value of the response variable. However, when the viewing angle is changed, the spikes may not appear this way. The spikes in the front may appear to be higher than the spikes in the back, which represent greater values.

See also: “About Response Variables” on page 1004.

Options

SURFACE statement options affect all maps that are produced by that statement.

ANNOTATE=Annotate-data-set

ANNO=Annotate-data-set

specifies a data set to annotate onto maps that are produced by the SURFACE statement. Annotate coordinate systems 1, 2, 7, and 8 are not valid with surface maps.

Note: You can use the %MAPLABEL Macro to create the *Annotate-data-set*. See “%MAPLABEL Macro” on page 686 for more information. Δ

See also: Chapter 24, “Using Annotate Data Sets,” on page 587.

CBODY=surface-map-color

specifies the color that is used to draw the surface map. By default, the first color in the current colors list is used.

CONSTANT=n

specifies a denominator to use in the distance decay function. This function determines the base width of the spike that is drawn at each map area center.

By default, CONSTANT=10. Values greater than 10 yield spikes that are wider at the base. Values less than 10 yield spikes that are narrower at the base.

Let x_k and y_k represent the coordinates, and z_k represent the function value at the center of each map area. The z_k values are scaled from 1 to 11. A square grid of x by y points (where the size of the grid is the NLINES= option value) and the associated function value $f(x,y)$ are generated from the map area center value using this formula:

$$f(x, y) = \sum^k \left(1 - 1.5^k + .5D^{3k} \right) \Delta^{kzk}$$

where

$$D^k = \left(x - x^k \right)^2 + \left(y - y^k \right)^k$$

and

$$\Delta^k = \left[\text{martix cdelim} = \text{XXXXXXXXXXXXXXXXX1 if } D^k < 1, 0 \text{ otherwise.} \right]$$

Featured in: Example 10 on page 1068.

DESCRIPTION='entry-description'

DES='entry-description'

specifies the description of the catalog entry for the map. The maximum length for *entry-description* is 256 characters. By default, the GMAP procedure assigns a description of the form SURFACE MAP OF *variable*, where *variable* is the name of the map variable.

NAME='entry-name'

When you specify DEVICE=ACTXIMG or DEVICE=JAVAIMG, specifies the name that will be used for the client image output even in the file exists. For all other devices, if the name duplicates an existing entry name, then SAS/GRAPH specifies the name of the catalog entry for the map. The maximum length for *entry-name* is eight characters. The default name is GMAP. If the specified name duplicates the name of an existing entry, then SAS/GRAPH software appends a number to the duplicate name to create a unique entry, for example, GMAP1.

NLINES=number-of-lines

N=number-of-lines

specifies the number of lines used to draw the surface map. Values can range from 50 to 100; the higher the value, the more solid the map appears and the more resources used. By default, NLINES=50.

Featured in: Example 10 on page 1068.

ROTATE=degrees

specifies the degrees of the angle at which to rotate the map about the Z axis in the map coordinate system. The *degrees* argument can be any angle. Positive values indicate rotation in the counterclockwise direction. By default, ROTATE=70. The ROTATE= option also affects the direction of the lines that are used to draw the surface map.

Featured in: Example 10 on page 1068.

TILT=degrees

specifies the degrees of the angle at which to tilt the map about the X axis in the map coordinate system. The value of *degrees* can be 0 to 90. Increasing values cause the map to tilt backward and makes the spikes more prominent. Decreasing values make the map shape more distinguishable and the spikes less prominent. TILT=90 corresponds to viewing the map edge-on, while TILT=0 corresponds to viewing the map from directly overhead. By default, TILT=70.

Featured in: Example 10 on page 1068.

XSIZE=map-width <units>

YSIZE=map-height <units>

specify the physical dimensions of the map. By default, the map uses the entire procedure output area.

Valid *units* are CELLS (character cells), CM (centimeters), IN (inches), or PCT (percentage of the graphics output area). The default unit is CELLS.

If you specify values for *map-width* and *map-height* that are greater than the dimensions of the procedure output area, the map is drawn using the default size. And if you specify only one dimension, the other is scaled to maintain the aspect ratio.

Using FIPS Codes and Province Codes

The map area identification variable in some SAS/GRAPH map data sets contain standardized numeric codes. The data sets for the United States contain a variable whose values are FIPS (Federal Information Processing Standards) codes. The data sets for Canada contain standard province codes or census division codes. When you use the GMAP procedure with a traditional map data set, the variables that identify map areas in your response data set must have the same values as the map area identification variables in the traditional map data set.

If both a feature table and a response data set contain FIPS Codes or Province Codes, then once both data sets have been sorted, a SQL or DATA step MERGE can be used to merge the two data sets using the variable containing the codes. However, with the merged response and feature table, the identification variable used in the GMAP procedure must be the \$GEOREF formatted variable that contains the spatial information. See “\$GEOREF format” on page 1001 for more information.

If the map area identification variables in your response data set are state or province names or abbreviations, convert them to FIPS codes or province codes before using the response data set with one of the map data sets supplied by SAS. Table 35.4 on page 1033 lists the FIPS codes for the United States and Table 35.5 on page 1034 lists the standard codes for Canadian provinces.

Table 35.4 U.S. FIPS Codes

FIPS Code	State	FIPS Code	State
01	Alabama	30	Montana
02	Alaska	31	Nebraska
04	Arizona	32	Nevada
05	Arkansas	33	New Hampshire
06	California	34	New Jersey
08	Colorado	35	New Mexico
09	Connecticut	36	New York
10	Delaware	37	North Carolina
11	District of Columbia	38	North Dakota
12	Florida	39	Ohio
13	Georgia	40	Oklahoma
15	Hawaii	41	Oregon
16	Idaho	42	Pennsylvania
17	Illinois	44	Rhode Island
18	Indiana	45	South Carolina
19	Iowa	46	South Dakota
20	Kansas	47	Tennessee
21	Kentucky	48	Texas
22	Louisiana	49	Utah
23	Maine	50	Vermont

FIPS Code	State	FIPS Code	State
24	Maryland	51	Virginia
25	Massachusetts	53	Washington
26	Michigan	54	West Virginia
27	Minnesota	55	Wisconsin
28	Mississippi	56	Wyoming
29	Missouri	72	Puerto Rico

Table 35.5 Canadian Province Codes

Province Code	Province
10	Newfoundland
11	Prince Edward Island
12	Nova Scotia
13	New Brunswick
24	Quebec
35	Ontario
46	Manitoba
47	Saskatchewan
48	Alberta
59	British Columbia
60	Yukon
61	Northwest Territories

Note: The ID variables in Canadian maps are character. \triangle

The CNTYNAME data set contains a cross-reference of names and FIPS codes for all counties in the United States. The CANCELS data set contains a cross-reference of census district names and codes for Canadian provinces.

Base SAS software provides several functions that convert state names to FIPS codes and vice versa. The following table lists these functions and a brief description of each. See *SAS Language Reference: Dictionary* for more information.

Table 35.6 FIPS and Postal Code Functions

Function	Description
STFIPS	converts state postal code to FIPS state code
STNAME	converts state postal code to state name in upper case
STNAMEL	converts state postal code to state name in mixed case
FIPNAME	converts FIPS code to state name in upper case

FIPNAMEL	converts FIPS code to state name in mixed case
FIPSTATE	converts FIPS code to state postal code

Using Formats for Maps

You can specify an output map area name or numeric value using one of the predefined formats for maps. The following prefixes are used in the names of the formats for maps:

CONT	Continent
CNTRY	Country
GLC	Geographic Location Code, distributed by Government Services Administration. USA
ISO	International Standard Organization

The formats for maps are located in the SASHELP.MAPFMTS catalog. See the MAPS.NAMES table to view all the continent and country names and corresponding GLC, ISO, and numeric representation for the continent values.

To use one of the formats for maps, you must specify the SASHELP.MAPFMTS catalog on the FMTSEARCH= option on a SAS OPTIONS statement:

```
options fmtsearch=(sashelp.mapfmts);
```

In addition to using the PUT statement (as shown in the examples in the following table), the formats can also be invoked using a FORMAT statement.

FORMAT	DESCRIPTION	EXAMPLE	OUTPUT
contfmt	use a continent's numeric value to output the continent's name	cont= 91 put (cont, contfmt.);	North America
\$cntrysl	use a country's short name in uppercase to output the country's long name in uppercase	name='IRAN' put (name, cntrysl.);	IRAN, ISLAMIC REPUBLIC OF
glcclu	use the GLC numeric code to output the country's long name in uppercase	id=460 put (id, glcclu.);	IRAN, ISLAMIC REPUBLIC OF
glcnsu	use the GLC numeric code to output the country's short name in uppercase	id=460 put (id, glcnsu.);	IRAN
glcnsm	use the GLC numeric code to output the country's name in mixed case	id=460 put (id, glcnsm.);	Iran

FORMAT	DESCRIPTION	EXAMPLE	OUTPUT
\$glcalu	use the GLC alpha code to output the country's long name in uppercase	country='IR' put (country, glcalu.);	IRAN, ISLAMIC REPUBLIC OF
\$glcsua	use the country's short name in uppercase to output the GLC alpha code name	name='IRAN' put (name, glcsua.);	IR
glcna	use the country's GLC numeric code to output the country's GLC alpha code	id=460 put (id, glcna.);	IR
\$glcsun	use the country's short name in uppercase to output the country's GLC numeric code	name='IRAN' put (name, glcsun.);	460
\$glcan	use the country's GLC alpha code to output the country's GLC numeric code	country='IR' put (country, glcan.);	460
\$glcsma	use the country's short name in mixed-case to output the country's GLC alpha code	mixname='Iran' put (mixname, glcsma.);	IR
\$glcsmn	use the country's short name in mixed-case to output the country's GLC numeric code	mixname='Iran' put (mixname, glcsmn.);	460
\$glcprov	use a province/city name appended by as a delimiter, followed by the country's GLC alpha code to output a province country code, the province/city code, and the country's GLC alpha numeric code	provname='TEHRAN IR' put (provname, glcprov.);	8250460 8250 — province/ city code 460 — country GLC numeric code
\$isosu2a	use the country's short name in uppercase to output the country's ISO alpha2 code	name='IRAN' put (name, \$isosu2a.);	IR
\$isosu3a	use the country's name in uppercase to output the country's ISO alpha3 code	name='IRAN' put (name, \$isosu3a.);	IRN
\$isosun	use the country's short name in uppercase to output the country's ISO numeric code	name='IRAN' put (name, isosun.);	364

FORMAT	DESCRIPTION	EXAMPLE	OUTPUT
ison2a	use the country's ISO numeric code to output the country's ISO alpha2 code	iso=364 put(iso,ison2a.);	IR
ison3a	use the country's ISO numeric code to output the country's ISO alpha3 code	iso=364 put(iso,ison3a.);	IRN
isonlu	use the country's ISO numeric code to output the country's long name in uppercase	iso=364 put(iso,isonlu.);	IRAN, ISLAMIC REPUBLIC OF
isonsu	use the country's ISO numeric code to output the country's short name in uppercase	iso=364 put(iso,isonsu.);	IRAN
isoa2lu	use the country's ISO alpha2 code to output the country's long name in uppercase	alpha2='IR' put(alpha2,isoa2lu.);	IRAN, ISLAMIC REPUBLIC OF
isoa2su	use the country's ISO alpha2 code to output the country's short name in uppercase	alpha2='IR' put(alpha2,isoa2lu.);	IRAN
isoa3lu	use the country's ISO alpha3 code to output the country's long name uppercase	alpha3='IRN' put(alpha3,isoa3lu.);	IRAN, ISLAMIC REPUBLIC OF
isoa3su	use the country's ISO alpha3 code to output the country's short name in uppercase	alpha3='IRN' put(alpha3,isoa3su.);	IRAN
\$isoa2n	use the country's ISO alpha2 code to output the country's ISO numeric code	alpha2='IR' put(alpha2,\$isoa2n.);	364
\$isoa3n	use the country's ISO alpha3 code to output the country's ISO numeric code	alpha3='IRN' put(alpha3,\$isoa3n.);	364
\$isosm2a	use the country's short name in mixed-case to output the country's ISO alpha2 code	mixname='Iran' put(mixname,\$isosn2a.);	IR

FORMAT	DESCRIPTION	EXAMPLE	OUTPUT
\$isosm3a	use the country's short name in mixed-case to output the country's ISO alpha3 code	<pre> mixname='Iran' put(mixname,\$isosn3a.); </pre>	IRN
\$isosmn	use the country's short name in mixed-case to output the country's ISO numeric code	<pre> mixname='Iran' put(mixname,\$isosmn.); </pre>	364

SAS/GRAPH Map Data Sets Reference Information

Before using your map data sets, contact your SAS Support Consultant to verify the name and location of the SAS data library that contains the map data sets at your site. Many sites automatically assign a libref of MAPS to the SAS data library that contains the SAS-supplied map data sets. However, if you use the map data sets regularly and your site does not automatically assign a libref to the data library that contains the map data sets, you can add a LIBNAME statement to your AUTOEXEC file that defines the location of the map data set library. If you do this, the libref for the maps is established automatically whenever you begin a SAS session.

Accessing Detailed Descriptions of Map Data Sets

You may need detailed information on the map data sets in order to determine their type, size, the variables they contain, or, in the case of traditional data sets, whether they are projected or unprojected. You can get this information by using the CONTENTS or DATASETS procedure, or browsing the MAPS.METAMAPS (see “Viewing Map Data Sets” on page 1001) data set in the MAPS library (or the library where your SAS-supplied map data sets reside). If the libref MAPS has automatically been assigned, you can see a complete list of map data sets by viewing the MAPS.METAMAPS data set. See

These statements list the map data sets in the SAS data library that is assigned to the libref MAPS:

```

libname maps 'SAS-data-library';

proc datasets lib=maps;
run;

```

Note: Be sure to replace *SAS-data-library* with the location of the SAS data library that contains map data sets at your site. \triangle

The following statements provide detailed information on a traditional map data set, including the number of observations, the variables in each data set, and a description of each variable:

```

libname maps 'SAS-data-library';

proc contents data=maps.canada3;

```

```
run;
```

To see the contents and descriptions of all of the SAS-supplied map data sets you can specify `DATA=MAPS._ALL_` in the `CONTENTS` procedure. See the *Base SAS Procedures Guide* for more information on the `CONTENTS` and `DATASETS` procedures.

Customizing SAS/GRAPH Map Data Sets

You can customize the area that is displayed on your map by using only part of a particular map data set. There are several ways to accomplish this. You can use `WHERE` processing or a `DATA` step to subset the map data to be used by the `GMAP` procedure.

With the traditional map data set, you can also use the `GPROJECT` procedure to create a rectangular subset of a map data set by using minimum and maximum longitude and latitude values.

You can combine traditional map data sets in either of these situations:

- The map data sets to be combined were originally projected together.
- The map data sets all contain the same type of coordinates. That is, all are in radians or all are in degrees.

SAS-supplied traditional map data sets that have coordinates expressed only as longitude and latitude, with variable names `LONG` and `LAT`, must be renamed `X` and `Y` and should be projected before displaying.

Subsetting Traditional Map Data Sets

Some of the SAS/GRAPH map data sets contain a large number of observations. Programs that use only a few states or provinces will run faster if you exclude the unused portion of the map data set or use an already reduced map data set. SAS provides several ways to accomplish this. One is to use the `WHERE` statement or `WHERE=` data set option within the `GMAP` procedure to select only the states or provinces you want.

The `WHERE` statement and `WHERE=` data set option are most useful when you produce a simple map and do not need to make any other changes to the data set. For example, to use only the observations for Quebec in the `CANADA` traditional map data set, begin the `GMAP` procedure with this statement:

```
proc gmap map=maps.canada(where=(province=24));
```

To use only North Carolina in `US2MERGED`, a data set created by using `SQL` or `DATA` step `MERGE` on the feature table `US2` and a response data set also containing the variable `STATE`, the `GMAP` procedure would begin with the following statement:

```
proc gmap data=work.us2merged(where=(STATE=37));
```

The `WHERE=` data set option applies only to the data set that you specify in the argument in which the `WHERE=` option appears. If you use the `WHERE` statement, the `WHERE` condition applies to the traditional map data set and the response data sets or the merged response and feature table.

Another approach is to use a `DATA` step to create a subset of the larger data set. This code illustrates another way to extract the observations for Quebec from the `CANADA` traditional map data set:

```
data quebec;
  set maps.canada(where=(province=24));
```

This code illustrates another way to extract North Carolina data from the US2 feature table:

```
data ncarolina;
  set maps.us2(where=(STATE=37));
```

This approach is most useful when you want to create a permanent subset of a map data set or when you need to make additional changes to the map data set.

Also see Chapter 42, “The GREMOVE Procedure,” on page 1223 for an example how to use GREMOVE to create a regional map from one of the traditional map data sets that are supplied with SAS/GRAPH.

Reducing Traditional Map Data Sets

A *reduced map data set* is one that can be used to draw a map that retains the overall appearance of the original map but that contains fewer points, requires considerably less storage space, and can be drawn much more quickly. You can improve performance by plotting fewer observations for each map area. You reduce a traditional map data set when you subset it on the variable DENSITY. You can add the variable DENSITY to a map data set by using the GREDUCE procedure. For more information, see Chapter 41, “The GREDUCE Procedure,” on page 1213.

An *unreduced map data set* contains all of the coordinates that were produced when the map was digitized. This type of map data set has more observations than most graphics output devices can accurately plot. Some unreduced map data sets already contain a DENSITY variable like the one calculated by the GREDUCE procedure, so it is not necessary to use the GREDUCE procedure to process these data sets. Values for DENSITY range from 0 through 6 (the lower the density, the coarser the boundary line).

A statement of this form excludes all points with a density level of 2 or greater:

```
proc gmap map=maps.states(where=(density<2));
```

The resulting map is much coarser than one drawn by using all of the observations in the data set, but it is drawn much faster.

Another way to create a reduced map data set is to use a DATA step to exclude observations with larger density values:

```
data states;
  set maps.states(where=(density<2));
```

Projecting Traditional Map Data Sets

Map data can be stored as unprojected or projected coordinates. Unprojected map data contains spherical coordinates, that is, longitude and latitude values usually expressed in radians.*

Each of the feature tables in the MAPS library are projected. A few traditional map data sets that are provided with SAS/GRAPH contain only unprojected coordinates and should be projected before you use them. They are

```
CANADA3
CANADA4
COUNTIES
COUNTY
STATES
```

Projected map data contains Cartesian coordinates. The GMAP procedure is designed to plot maps by using projected map data sets. Feature tables store the

* If your data is in degrees, then it can be converted to radians by multiplying by the degree-to-radian constant [atan(1)/45].

projected data in the \$GEOREF formatted variable (see “\$GEOREF format” on page 1001). Most SAS/GRAPH traditional map data sets contain projected coordinates that are stored as X and Y.

If the projection supplied with the traditional map data set does not meet your needs, then you can use the GPROJECT procedure (on unprojected map coordinates) to create a different projection. For more information on traditional map data sets with unprojected coordinates, see “Traditional Map Data Sets Containing X, Y, LONG, and LAT” on page 1000. You should select a projection method that least distorts the regions that you are mapping. (All projection methods inherently distort map regions.) See Chapter 39, “The GPROJECT Procedure,” on page 1161 for more information.

Note: Using an unprojected traditional map data set with the GMAP procedure can cause your map to be reversed and distorted. Δ

Controlling the Display of Lakes

Some countries contain a lake that is located completely within a single unit area. Occasionally these lakes can be a problem when mapping traditional map data sets. In addition, displaying lakes may not be appropriate for some applications. In these cases, you may want to remove the lakes from the map data set before you proceed.

Traditional map data sets that contain coordinates for a lake that is located within a single internal division are identified by the presence of the numeric variable LAKE. The value of LAKE is 1 for points that correspond to lakes and 0 otherwise. The following statements illustrate how to delete the lakes from your traditional map data sets using WHERE processing:

```
proc gmap map=maps.chile(where=(lake=0))
    data=maps.chile;
    id id;
    choro id / levels=1 nolegend;
    title box=1 f=none h=4
        'Chile with Lakes Removed';
run;
```

You can also create a new traditional map data set that is a subset of the traditional map data set:

```
data nolake;
    set maps.chile;
    if lake=0;
run;
```

Creating Traditional Map Data Sets

In addition to using map data sets that are supplied with SAS/GRAPH software, you can also create your own map data sets. Map data sets are not limited to geographic data; you use them to define other spaces such as floor plans or street diagrams.

A unit area is defined by observations in the map data set that have the same identification (ID) variable value. A unit area may be composed of a single polygon or a collection of polygons. A polygon is defined by all of the observations that have the same SEGMENT variable value within the same unit area.

- If the unit area is a single polygon, then all values of SEGMENT are the same.
- If the unit area contains multiple polygons, such as islands, then the SEGMENT variable has multiple values. For example, in the MAPS.US data set, the state of Hawaii (a unit area) contains six different values in the SEGMENT variable, one for each island in the state.

- If the unit area contains enclosed polygons, such as lakes, then the SEGMENT variable has one value but the interior polygon is defined by separate boundaries. To separate boundaries, a missing X and Y value must be inserted at the separation point. For example, in the CANADA2 data set supplied with SAS/GRAPH, the map data for the Northwest Territories (a unit area) use enclosed polygons for two lakes.

Creating a Unit Area that is a Single Polygon

This DATA step creates a SAS data set that contains coordinates for a unit area with a single polygon, a square:

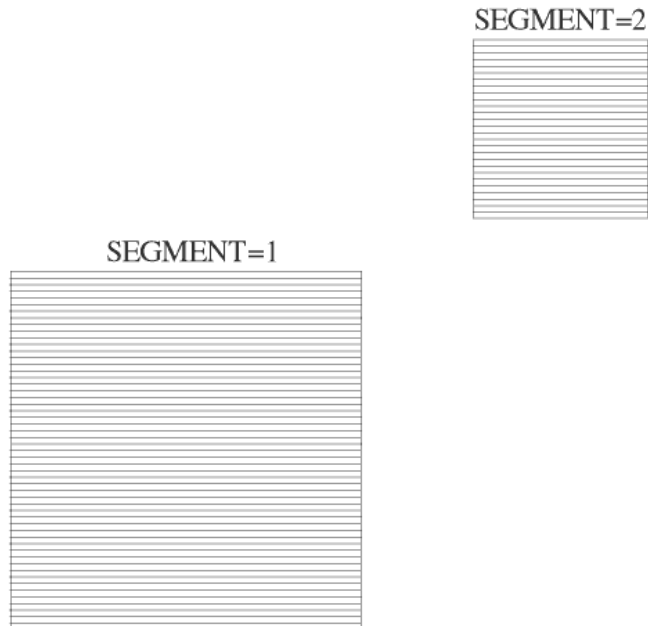
```
data square;
  input id x y;
  datalines;
1 0 0
1 0 40
1 40 40
1 40 0
;
```

This data set does not have a SEGMENT variable.

Creating a Unit Area that Contains Multiple Polygons

Use different values of the SEGMENT variable to create separate polygons within a single unit area. For example, this DATA step assigns two values to the SEGMENT variable. The resulting data set produces a single unit area that contains two polygons, as shown in Figure 35.7 on page 1043:

```
data map;
  input id $ 1-8 segment x y;
  datalines;
square 1 0 0
square 1 0 4
square 1 4 4
square 1 4 0
square 2 5 5
square 2 5 7
square 2 7 7
square 2 7 5
;
```

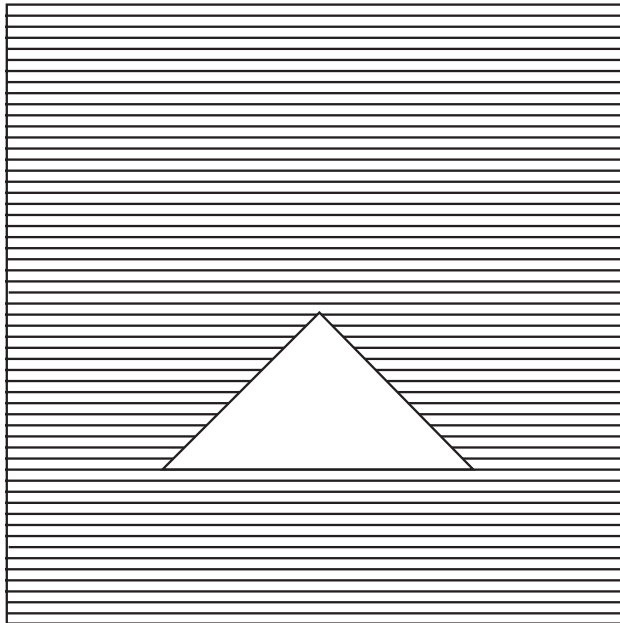

Figure 35.7 Single Unit Area with Two Segments (Polygons)

Creating a Unit Area that Contains Enclosed Polygons as Holes

Use separate boundaries to create an enclosed polygon (that is, a polygon that falls within the primary polygon for a single segment). The separate boundaries are separated from the primary polygon boundary by missing values for X and Y. For example, the data set that is created by this DATA step produces the map shown in Figure 35.8 on page 1044:

```
data map;
  input id $ 1-8 segment x y;
  datalines;
square  1 0 0
square  1 0 4
square  1 4 4
square  1 4 0
square  1 . .
square  1 1 1
square  1 2 2
square  1 3 1
;
```

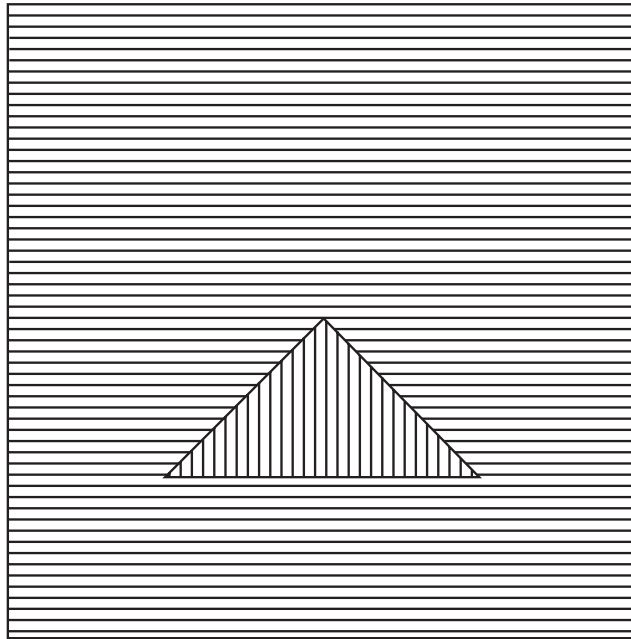
Figure 35.8 Single Unit Area with Hole



Creating a Unit Area that Contains Enclosed Polygons as Cities

Ordinarily, if one unit area is surrounded by another, the pattern of the external unit area is drawn over the pattern for the internal one, instead of around it. Avoid this problem by adding an observation to the map data for the external unit area with missing values for X and Y, followed by the coordinates of the internal unit area, but using the ID values for the external unit area. For example, this DATA step creates a data set that produces the map shown in Figure 35.9 on page 1045:

```
data map;
  input id $ 1-8 segment x y;
  datalines;
square 1 0 0
square 1 0 4
square 1 4 4
square 1 4 0
square 1 . .
square 1 1 1
square 1 2 2
square 1 3 1
triangle 1 1 1
triangle 1 2 2
triangle 1 3 1
;
```

Figure 35.9 Unit Area within a Unit Area

Note: A single map segment (a section of a unit area with a single value of the SEGMENT variable) cannot contain multiple polygons without at least one observation with missing values for X and Y. All segments within the map data sets that are supplied by SAS/GRAPH contain a single polygon that can have one or more separate boundaries, each separated by an observation with missing values for X and Y. Δ

Examples

The following examples include features from one or more of the GMAP statements.

Example 1: Producing a Simple Block Map

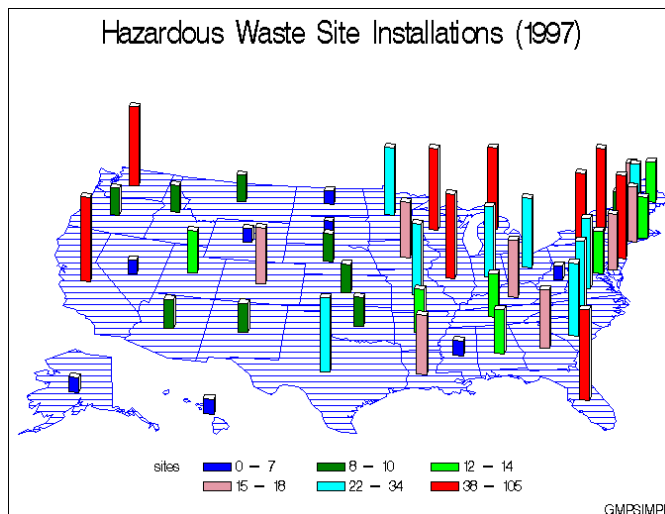
Procedure features:

ID statement

BLOCK statement option:

CBLKOUT=

Sample library member: GMPSIMPL



This example produces a block map that shows the total number of hazardous waste sites in each state in 1997. Since the DISCRETE option is not used, the response variable is assumed to have a continuous range of values. Because neither the LEVELS= nor MIDPOINTS= option is used, the GMAP procedure selects a number of levels based on the number of map areas and then calculates the appropriate response levels.

The blocks use the default pattern, which is a solid fill that rotates through the colors list. Because the colors list is specified in the GOPTIONS statement, all colors are used in the rotation. CBLKOUT= outlines the blocks in black, instead of using the default outline color, which is the first color in the list— in this case, BLUE.

The map areas use the default pattern for map areas in a block map. This is the first hatch pattern for maps, M2N0. By default, both the fill and the outline use the first color in the colors list.

Assign the libref and set the graphics environment. COLORS= specifies the colors list, which is used by the default patterns and outlines. CTEXT= specifies the color for all text on the output.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(blue green lime lipk cyan red)
          ctext=black ftext=swiss htitle=6 htext=3;
```

Create response data set SITES. This data set contains a map area identification variable, STATE, and a response variable, SITES. The STFIPS function is used to convert the state postal codes to FIPS state codes. STATE contains the FIPS codes for each state and matches the values of STATE in the MAPS.US data set. SITES contains the total number of waste sites installed in the state.

```
data sites;
  length stcode $ 2;
  input region stcode $ sites;
  state=stfips(stcode);
  datalines;
6   AR   12
10  AK   7...moredata lines...
```

```

3   WV   6
8   WY   3
;

```

Define title and footnote for map.

```

title1 'Hazardous Waste Site Installations (1997)';
footnote1 j=r 'GMPSIMPL';

```

Produce the block map. The ID statement specifies the variable that is in both the map data set and the response data set and defines map areas. The BLOCK statement specifies the variable in the response data set that contains the response values for each of the map areas. CBLKOUT= specifies the color for the block outlines.

```

proc gmap map=maps.us data=sites;
  id state;
  block sites / cblkout=black;
run;
quit;

```

Example 2: Specifying Response Levels in a Block Map

Procedure features:

BLOCK statement options:

```

LEGEND=
LEVELS=
SHAPE=
XVIEW=
ZVIEW=

```

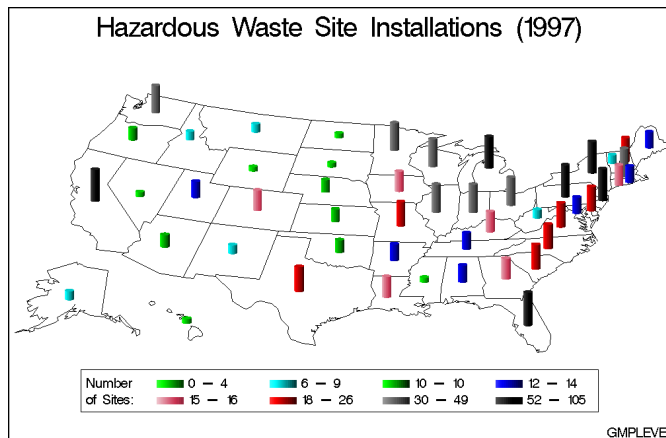
Other features:

```

LEGEND statement
PATTERN statement

```

Sample library member: GMPLEVEL



This example uses `LEVELS=` to specify the number of response levels for the blocks. `LEVELS=` tells `GMAP` how many response levels and `GMAP` calculates the quantiles. Eight `PATTERN` statements explicitly define a color for each of these response levels.

A single `PATTERN` statement uses the `REPEAT=` option to define an empty map/plot pattern outlined in black for all the map areas.

The example also changes the viewpoint by rotating the map to provide a better view of the northeast states. As a result, the blocks appear shorter.

Assign the libref and set the graphics environment.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ctext=black ftext=swiss htitle=6 htext=3;
```

Create response data set `SITES`. This data set contains a map area identification variable, `STATE`, and a response variable, `SITES`. The `STFIPS` function is used to convert the state postal codes to FIPS state codes. `STATE` contains the FIPS codes for each state and matches the values of `STATE` in the `MAPS.US` data set. `SITES` contains the total number of waste sites installed in the state.

```
data sites;
  length stcode $ 2;
  input region stcode $ sites;
  state=stfips(stcode);
  datalines;
6   AR  12
10  AK  7...moredata lines...
3   WV  6
8   WY  3
;
```

Define title and footnote for map.

```
title1 'Hazardous Waste Site Installations (1997)';
footnote1 j=r 'GMPLEVEL';
```

Define the patterns for the blocks. PATTERN statements 1-8 specify bar/block patterns and cannot be used by the map areas. They are applied to the blocks in order of the response level.

```
pattern1 value=solid color=lime;
pattern2 value=solid color=cyan;
pattern3 value=solid color=green;
pattern4 value=solid color=blue;
pattern5 value=solid color=lipk;
pattern6 value=solid color=red;
pattern7 value=solid color=gray;
pattern8 value=solid color=black;
```

Define a pattern for the map areas. PATTERN9 defines a single map pattern that is repeated for each of the 50 map areas (states). The pattern is an empty fill with a black border. VALUE= defines a map/plot pattern, which cannot be used by the blocks. Specifying a color causes PATTERN9 to generate only one pattern definition. REPEAT= specifies the number of times to repeat the pattern definition.

```
pattern9 value=empty color=black repeat=50;
```

Define legend characteristics. LABEL= produces a two line label and places it to the left of the legend values. FRAME draws a border around the legend using the first color in the colors list.

```
legend1 value=(justify=left)
        label=('Number' justify=left 'of Sites:'
              position=(middle left))
        frame;
```

Produce the block map. LEVELS= specifies the number of response levels for the graph. SHAPE= draws the blocks as 3D cylinders. XVIEW= changes the viewpoint for the map so that the map appears to be slightly rotated. ZVIEW= raises the height of the viewpoint. LEGEND= assigns the LEGEND1 statement to the map legend.

```
proc gmap map=maps.us data=sites;
  id state;
  block sites / levels=8
              shape=cylinder
              xview=0.75
              zview=5
              legend=legend1;
run;
quit;
```

Example 3: Assigning a Format to the Response Variable

Procedure features:

BLOCK statement options:

```

AREA=
CBLKOUT=
COUTLINE=
DISCRETE
WOUTLINE=

```

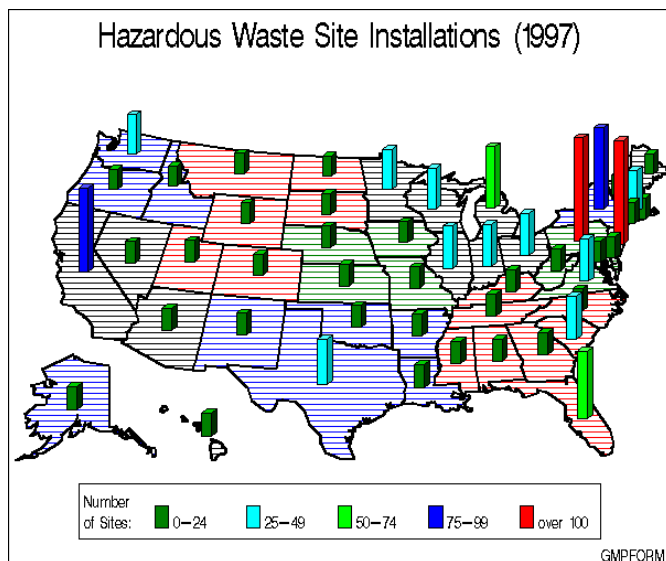
Other features:

```

FORMAT statement
LEGEND statement
PATTERN statement

```

Sample library member: GMPFORMT



This example creates a format that defines the ranges of values for the response values and assigns this format to the response variable. These ranges appear in the legend and make the map easier to understand. When a format is assigned to a numeric response variable, the DISCRETE option must be used so that each formatted value is treated as a separate response level.

The example also patterns the map areas by region. To do this, both data sets must contain the ID variable, REGION. The response data set, SITES, already contains REGION, so the program only needs to add it to the map data set. Then the map data set is sorted by both the ID variables, REGION and STATE. Finally, the AREA= option specifies that the ID variable REGION is the one by which the map areas are patterned.

Assign the libref and set the graphics environment.

```

libname maps 'SAS-data-library';
options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss htitle=6 htext=3;

```

Create map data set STATES1 by adding REGION to the MAPS.US map data set.


```

data states1;
  set maps.us;
  select;
    when (state in (9,23,25,33,44,50))      region=1;
    when (state in (34,36))                region=2;
    when (state in (10,11,24,42,51,54))    region=3;
    when (state in (1,12,13,21,28,37,45,47)) region=4;
    when (state in (17,18,26,27,39,55))    region=5;
    when (state in (5,22,35,40,48))        region=6;
    when (state in (19,20,29,31))          region=7;
    when (state in (8,30,38,46,49,56))    region=8;
    when (state in (4,6,15,32))            region=9;
    otherwise                               region=10;
  end;
run;

```

Sort the new map data set. The map data must be sorted in the order of the ID variables.

```

proc sort data=states1 out=states2;
  by region state;
run;

```

Create response data set SITES. This data set contains a map area identification variable, STATE, and a response variable, SITES. The STFIPS function is used to convert the state postal codes to FIPS state codes. STATE contains the FIPS codes for each state and matches the values of STATE in the MAPS.US data set. SITES contains the total number of waste sites installed in the state.

```

data sites;
  length stcode $ 2;
  input region stcode $ sites;
  state=stfips(stcode);
  datalines;
6  AR  12
10 AK  7...moredata lines...
3  WV  6
8  WY  3
;

```

Create a format for SITES. SITESFMT. defines and labels the ranges of values for SITES.

```

proc format;
  value sitesfmt low-24='0-24'
                 25-49='25-49'
                 50-74='50-74'
                 75-99='75-99'
                 100-high='over 100';
run;

```

Define title and footnote for map.

```

title1 'Hazardous Waste Site Installations (1997)';
footnote j=r 'GMPFORMT';

```

Define a hatch pattern for the map areas. PATTERN1 defines a dense hatch pattern for the map areas. Because there are four colors in the colors list, the pattern rotation must be repeated three times to create enough patterns for the ten regions.

```

pattern1 value=m3n0 r=3;

```

Define a solid pattern for the blocks. PATTERN2 through PATTERN6 define the patterns for the block surfaces.

```

pattern2 value=solid color=green;
pattern3 value=solid color=cyan;
pattern4 value=solid color=lime;
pattern5 value=solid color=blue;
pattern6 value=solid color=red;

```

Define legend characteristics.

```

legend1 shape=bar(2,4)
value=(j=1)
label=('Number' j=1 'of Sites:')
frame;

```

Produce the block maps. The FORMAT statement assigns SITESFMT. to the response variable. DISCRETE specifies that each formatted value is a separate response level. AREA= specifies that the map surface should be patterned by the first variable in the ID statement, REGION. CBLKOUT= and COUTLINE= specify the color that outlines the blocks and the regions, respectively. WOUTLINE= specifies the width of the block outline in pixels.

```

proc gmap map=states2 data=sites;
  format sites sitesfmt.;
  id region state;
  block sites / discrete
    area=1
    legend=legend1
    shape=block
    cblkout=black
    coutline=black
    woutline=3;
run;
quit;

```

Example 4: Producing a Simple Choropleth Map

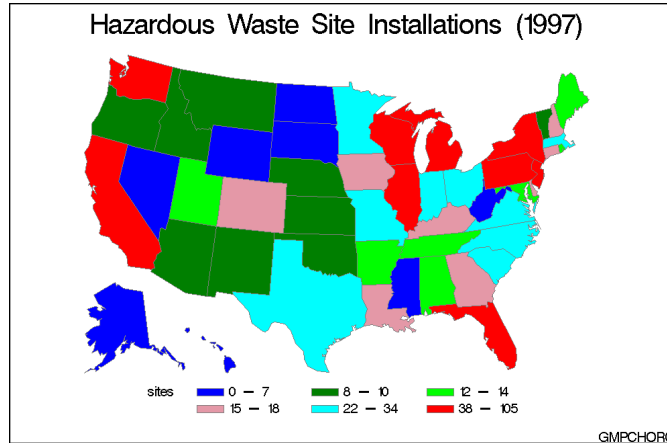
Procedure features:

ID statement

CHORO statement option:

COUTLINE=

Sample library member: GMPCHORO



This example produces a choropleth (2D) map that shows the total number of hazardous waste sites in each state in 1997. Since the DISCRETE option is not used, the response variable is assumed to have a continuous range of values. Because neither the LEVELS= nor MIDPOINTS= option is used, the GMAP procedure selects a number of levels based on the number of map areas and then calculates the appropriate response levels. The legend shows the midpoint value of each level.

The map areas use the default pattern, which is a solid fill that rotates through the colors list. Because the colors list is specified in the GOPTIONS statement, all colors are used in the rotation. COUTLINE= outlines the map areas in gray, instead of the default outline color, which is the first color in the list, in this case, BLUE.

Assign the libref and set the graphics environment. COLORS= specifies the colors list, which is used by the default patterns and outlines. CTEXT= specifies the color for all text on the output.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
         colors=(blue green lime lipk cyan red)
         ctext=black ftext=swiss htitle=6 htext=3;
```

Create response data set SITES. This data set contains a map area identification variable, STATE, and a response variable, SITES. The STFIPS function is used to convert the state postal codes to FIPS state codes. STATE contains the FIPS codes for each state and matches the values of STATE in the MAPS.US data set. SITES contains the total number of waste sites installed in the state.

```
data sites;
  length stcode $ 2;
  input region stcode $ sites;
```

```

state=stfips(stcode);
datalines;
6  AR  12
10 AK  7...moredata lines...
3  WV  6
8  WY  3
;

```

Define title and footnote for map.

```

title1 'Hazardous Waste Site Installations (1997)';
footnote1 j=r 'GMPCHORO';

```

Produce the choropleth map. The ID statement specifies the variable that is in both the map data set and the response data set that defines map areas. COUTLINE= specifies the color for the map area outlines.

```

proc gmap map=maps.us data=sites;
  id state;
  choro sites / outline=gray;
run;
quit;

```

Example 5: Creating Maps with Drill-down for the Web

Procedure Features:

CHORO statement options:

```

DES=
DISCRETE
HTML=
NAME=

```

BLOCK statement options:

```

BLOCKSIZE=
DES=
MIDPOINTS=
NAME=

```

ODS Features:

ODS HTML statement :

```

BODY=
CONTENTS=
FRAME=
NOGTITLE
PATH=

```

Other Features:

```

BY statement
GOPTIONS statement
LEGEND statement

```

PATTERN statement

TITLE statement

Sample library member: GMPDRILL

This example shows how to create a 2D choropleth map with simple drill-down functionality for the Web. When this map is displayed in a browser, you can select an area of the map and display additional information about the data.

The example explains how to use the ODS HTML statement and the HTML procedure options to create the drill-down. It shows how to

- explicitly name the HTML files and open and close them throughout the program
- use BY-group processing with ODS HTML, including storing multiple graphs in one file and incrementing anchor names, catalog entry names, and graphics file names
- use the PATH= option to specify the destination for the HTML and GIF files created by the ODS HTML statement
- use the NAME= option to name the graphics catalog entries
- assign anchor names to the graphics output with the ANCHOR= option in the ODS HTML statement
- add an HTML HREF string to a data set to define a link target
- assign link targets with the HTML= procedure option
- use DES= to control the text of the table of contents entry
- suppress the titles in the GIF files and display them in the HTML file.

For more information, see “ODS HTML Statement” on page 164.

The example also illustrates other CHORO and BLOCK statement options.

The program produces one choro map that shows Environmental Protection Agency (EPA) regions and block maps of the states in each region. Each block map shows the number of hazardous waste sites for each state in the selected region. Figure 35.10 on page 1055 shows the map of the EPA regions.

Figure 35.10 Browser View of Regional Map

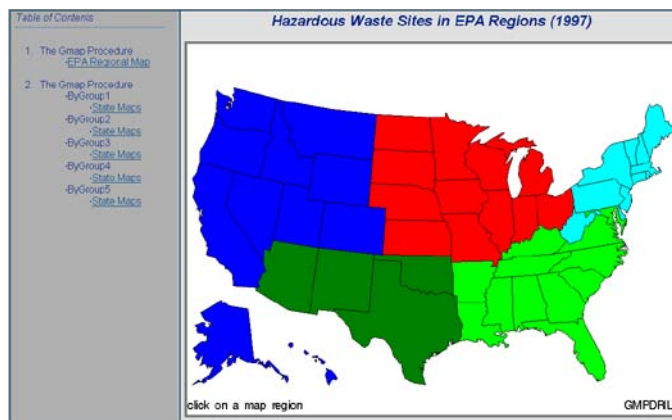
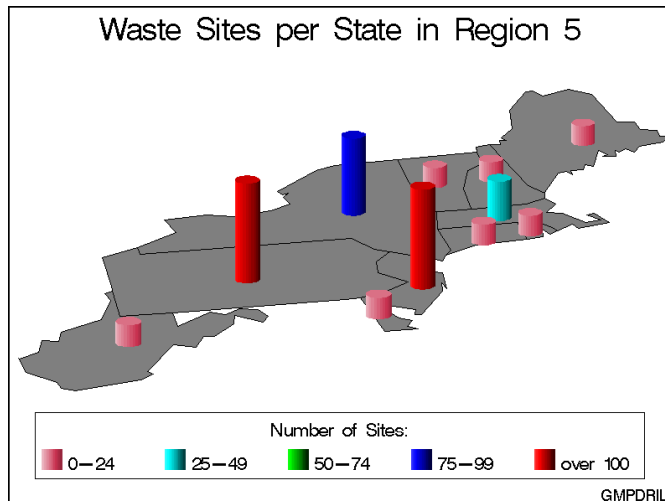


Figure 35.11 on page 1056 shows the block map that appears when you select Region 5 in the map.

Figure 35.11 Browser View of Region 5 Block Map



Assign the libref and the Web-server path. FILENAME assigns the fileref ODSOUT, which specifies a destination for the HTML and GIF files produced by the example program. To assign that location as the HTML destination for program output, ODSOUT is specified later in the program on the ODS HTML statement's PATH= option. ODSOUT must point to a Web-server location if procedure output is to be viewed on the Web.

```
libname maps 'SAS-MAPS-library';
filename odsout 'path-to-Web-server-space';
```

Close the ODS Listing destination for procedure output, and set the graphics environment. To conserve system resources, ODS LISTING CLOSE closes the Listing destination for procedure output. Thus, the graphics output is not displayed in the GRAPH window, although it is written to the catalog.

```
ods listing close;
options reset=global gunit=pct cback=white
        colors=(black blue green red)
        ftext=swiss htitle=6 htext=3.5;
```

Create the data set SITES. SITES contains the FIPS codes for each state and the total number of hazardous waste sites installed in each state. The STFIPS function converts state postal codes to FIPS state codes.

```
data sites;
  length stcode $ 2;
  input region stcode $ sites;
  state=stfips(stcode);
  datalines;
1  AK  12
4  AL  7
4  AR  12
```

```

2 AZ 10
1 CA 90
1 CO 15
5 CT 15
5 DE 18
4 FL 52
4 GA 15
1 HI 4
3 IA 16
1 ID 8
3 IL 38
3 IN 30
3 KS 10
4 KY 16
4 LA 15
5 MA 30
4 MD 13
5 ME 12
3 MI 72
3 MN 30
3 MO 22
4 MS 1
1 MT 8
4 NC 22
3 ND 0
3 NE 10
5 NH 18
5 NJ 105
2 NM 9
1 NV 1
5 NY 78
3 OH 34
2 OK 10
1 OR 10
5 PA 100
5 RI 12
4 SC 26
3 SD 2
4 TN 14
2 TX 26
1 UT 12
4 VA 25
5 VT 8
1 WA 49
3 WI 40
5 WV 6
1 WY 3
;

```

Add the HTML variable to SITES and create the NEWSITES data set. The HTML variable REGIONDRILL contains the targets for the values of the variable REGION.

```

data newsites;
  length regiondrill $40;
  set sites;
  if region=1 then
    regiondrill='HREF="hazsite_statebody.html#Region1"';
  if region=2 then
    regiondrill='HREF="hazsite_statebody.html#Region2"';
  if region=3 then
    regiondrill='HREF="hazsite_statebody.html#Region3"';
  if region=4 then
    regiondrill='HREF="hazsite_statebody.html#Region4"';
  if region=5 then
    regiondrill='HREF="hazsite_statebody.html#Region5"';
run;

```

Assign graphics options for producing the ODS HTML output. DEVICE=GIF causes the ODS HTML statement to generate the graphics output as GIF files. TRANSPARENCY causes the graphics output to use the Web-page background as the background of the graph.

```
goptions device=gif transparency;
```

Open the ODS HTML destination. BODY= names the file for storing HTML output. CONTENTS= names the HTML file that contains the table of contents to the HTML procedure output. The contents file links to each of the body files written to the HTML destination. FRAME= names the HTML file that integrates the contents and body files. NOGTITLE suppresses the graph titles from the SAS/GRAPH output and displays them through the HTML page. PATH= specifies the ODSOUT fileref as the HTML destination for all the HTML and GIF files.

```

ods html body='hazsite_mapbody.html'
      contents='hazsite_contents.html'
      frame='hazsite_frame.html'
      nogtitle
      path=odsout;

```

Define the title and footnote for the map of the EPA regions.

```

title1 'Hazardous Waste Sites in EPA Regions (1997)';
footnote1 h=3 j=1 'click on a map region' j=r 'GMPDRILL1';

```

Define a map pattern for each region. Each PATTERN statement defines one map/plot pattern. The patterns are assigned to the map areas that represent the EPA regions

```

pattern1 value=msolid color=blue;
pattern2 value=msolid color=green;
pattern3 value=msolid color=red;
pattern4 value=msolid color=lime;
pattern5 value=msolid color=cyan;

```


Generate the regional map. The ID statement specifies the variable that defines the map areas and is in both the map data set and the response data set. DISCRETE specifies that each value of the numeric response variable, STATE, be treated as a separate response level. HTML= specifies REGIONDRILL as the variable that contains the targets for the map regions. Specifying HTML variables causes SAS/GRAPH to add an image map to the HTML body file. DES= specifies the description that is stored in the catalog and used in the Table of Contents. NAME= specifies the name of the graphics catalog entry. Because the PATH= destination is a file storage location and not a specific file name, the catalog entry name EPAMAP is automatically assigned to the GIF file.

```
proc gmap map=maps.us data=newsites;
  id state;
  choro region / discrete
             html=regiondrill
             coutline=black
             nolegend
             des='EPA Regional Map'
             name='epamap';
run;
quit;
```

Open a new body file for the state maps. Assigning a new body file closes HAZSITE_MAPBODY.HTML. The contents and frame files, which remain open, will provide links to all body files. ANCHOR= specifies the name of the anchor that identifies the link target. This name is automatically incremented when the graphics output is generated. GTITLE uses titles in the GIF files.

```
ods html body='hazsite_statebody.html'
      anchor='Region1'
      gtitle
      path=odsout;
```

Assign new graphics options for ODS HTML output. The active device is still GIF.

```
goptions notransparency
      border;
```

Sort the response data set NEWSITES in order of the BY variable. The data must be in sorted order before running the GMAP procedure with BY-group processing.

```
proc sort data=newsites;
  by region;
run;
```

Define legend characteristics for the state maps. VALUE= specifies text for the legend values that describes the ranges specified by MIDPOINTS= in the BLOCK statement.

```
legend1 shape=bar(3,4)
      label=('Number of Sites'
            position=(top center))
```

```
value=(j=1 '0-24' '25-49' '50-74' '75-99' 'over 100')
frame;
```

Define a pattern for the map areas. Because the procedure uses BY-group processing to generate the maps, all map areas use the same map pattern.

```
pattern1 value=ms color=gray;
```

Define the patterns for the blocks.

```
pattern2 value=solid color=lipk;
pattern3 value=solid color=cyan;
pattern4 value=solid color=green;
pattern5 value=solid color=blue;
pattern6 value=solid color=red;
```

Suppress the default BY-line and define a title that includes the BY-values. #BYVAL inserts the value of the BY variable into the title of each block map.

```
options nobyline;
title1 'Wastes Sites per State in Region #byval(region)';
footnotel h=3 j=r 'GMPDRIL2';
```

Generate the block maps for each region. MIDPOINTS= defines the midpoints of the ranges described in the legend. NAME= is a full 8 characters ending in 1 so the incremented names match the regions. NAME= specifies the name of the first catalog entry. Because BY-group processing generates multiple graphs from one BLOCK statement, the name assigned by NAME= is incremented to provide a unique name for each piece of output. These names are automatically assigned to the GIF files. DES= specifies the description that is stored in the catalog and used in the Table of Contents. Because BY-group processing is used, the same description is assigned to all the output.

```
proc gmap map=maps.us data=newsites;
  by region;
  id state;
  block sites / midpoints=(12 37 62 87 112)
    legend=legend1
    shape=cylinder
    blocksize=4
    coutline=black
    des='State Maps'
    name='states01';
run;
quit;
```

Close the ODS HTML destination, and open the ODS Listing destination. You must close the HTML destination before you can view the output with a browser.

```
ods html close;
ods listing;
```

Example 6: Labeling the States on a U.S. Map

Procedure features:

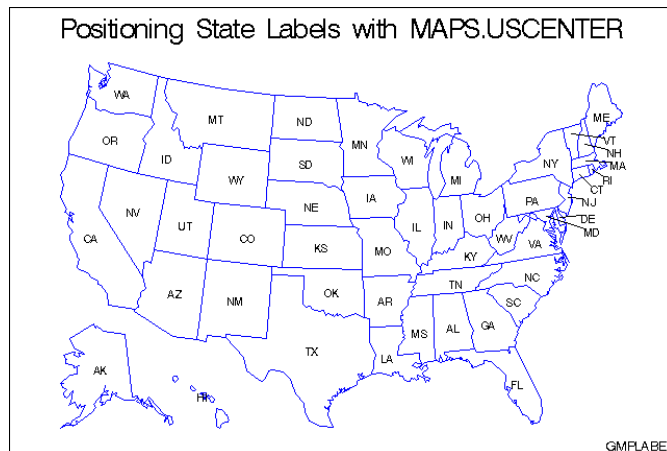
CHORO statement options:

```
ANNOTATE=
NOLEGEND
```

Other features:

Annotate Facility

Sample library member: GMPLABEL



This example uses the MAPS.USCENTER data set and the Annotate facility to add postal code labels to each state. The program first builds an Annotate data set that contains the instructions for drawing the labels. Some of the labels are in the center of the state and others use external labeling with leader lines. The CHORO statement assigns the Annotate data set to the map.

Note: The coordinates in MAPS.USCENTER have been projected to match coordinates in the MAPS.US data set. \triangle

Assign the libref and set the graphics environment.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ftext=swiss htitle=6 htext=3;
```

Create annotate data set, CENTER, from MAPS.USCENTER. The annotate data set labels each state with a two-letter abbreviation. MAPS.USCENTER provides the x and y coordinates for the labels. FLAG, which is initially turned off, signals when external labeling is in effect. The labels are drawn after the map because the value of WHEN is **a** (after). The FIPSTATE function converts the FIPS codes to state postal codes.

```
data center;
  length function $ 8;
  retain flag 0 xsys ysys '2' hsys '3' when 'a'
         style 'swiss';
  set maps.uscenter
      (where=(fipstate(state) ne 'DC')
       drop=long lat);
```

The FIPSTATE function creates the label text by converting the FIPS codes from MAPS.USCENTER to two-letter postal codes.

```
function='label';
text=fipstate(state);
size=2.5;
position='5';
```

If the labeling coordinates are outside the state (**OCEAN='Y'**), Annotate adds the label and prepares to draw the leader line. **Note:** OCEAN is a character variable and is, therefore, case sensitive. **OCEAN='Y'** must specify an uppercase Y.

```
if ocean='Y' then
  do;
    position='6';
    output;
    function='move';
    flag=1;
  end;
```

When external labeling is in effect, Annotate draws the leader line and resets the flag.

```
else if flag=1 then
  do;
    function='draw';
    size=.25;
    flag=0;
  end;
  output;
run;
```

Define title and footnote for map.

```
title 'Positioning State Labels with MAPS.USCENTER';
footnote j=r 'GMP LABEL';
```

Define pattern characteristics. PATTERN1 defines a single map pattern that is repeated for each of the 50 map areas (states). The pattern is an empty fill with a blue border. VALUE= defines a map/plot pattern, which cannot be used by the blocks. Specifying a color causes PATTERN1 to generate only one pattern definition. REPEAT= specifies the number of times to repeat the pattern definition.

```
pattern1 value=empty color=blue repeat=50;
```

Produce the choropleth map. NOLEGEND suppresses the legend. ANNOTATE= specifies the data set to annotate the map.

```
proc gmap data=maps.us map=maps.us;
  id state;
  choro state / nolegend
              annotate=center;
run;
quit;
```

Example 7: Producing a Simple Prism Map

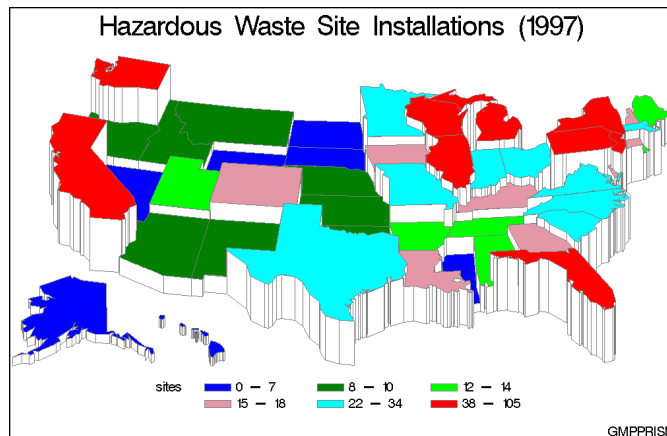
Procedure features:

ID statement

PRISM statement option:

COUTLINE=

Sample library member: GMPPRISM



This example produces a prism map of the hazardous waste sites. Since the DISCRETE option is not used, the response variable is assumed to have a continuous range of values. Because neither the LEVELS= nor MIDPOINTS= option is used, the GMAP procedure selects a number of levels based on the number of map areas and then calculates the appropriate response levels.

The map areas use the default pattern, which is a solid fill that rotates through the colors list. Because the colors list is specified in the GOPTIONS statement, all colors are used in the rotation. COUTLINE= outlines the map areas in gray, instead of the default outline color, which is the first color in the list, in this case, BLUE.

Since the XVIEW=, YVIEW=, and ZVIEW= options are not used, the default viewing position, above and to the east and south of the center of the map, is used. Since the XLIGHT= and YLIGHT= options are not used, none of the side polygons of the prisms are shadowed. The light source is the same as the viewing position.

Assign the libref and set the graphics environment. COLORS= specifies the colors list, which is used by the default patterns and outlines. CTEXT= specifies the color for all text.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
         colors=(blue green lime lipk cyan red)
         ctext=black ftext=swiss htitle=6 htext=3;
```

Create response data set SITES. This data set contains a map area identification variable, STATE, and a response variable, SITES. The STFIPS function is used to convert the state postal codes to FIPS state codes. STATE contains the FIPS codes for each state and matches the values of STATE in the MAPS.US data set. SITES contains the total number of waste sites installed in the state.

```
data sites;
  length stcode $ 2;
  input region stcode $ sites;
  state=stfips(stcode);
  datalines;
6   AR   12
10  AK   7...moredata lines...
3   WV   6
8   WY   3
;
```

Define title and footnote for the map.

```
title1 'Hazardous Waste Site Installations (1997)';
footnote1 j=r 'GMPPRISM';
```

Produce the prism map. The ID statement specifies the variable in the map data set and the response data set that defines map areas. COUTLINE= specifies the map area outline color.

```
proc gmap map=maps.us data=sites;
  id state;
  prism sites / coutline=gray;
run;
quit;
```

Example 8: Specifying Midpoints in a Prism Map

Procedure features:

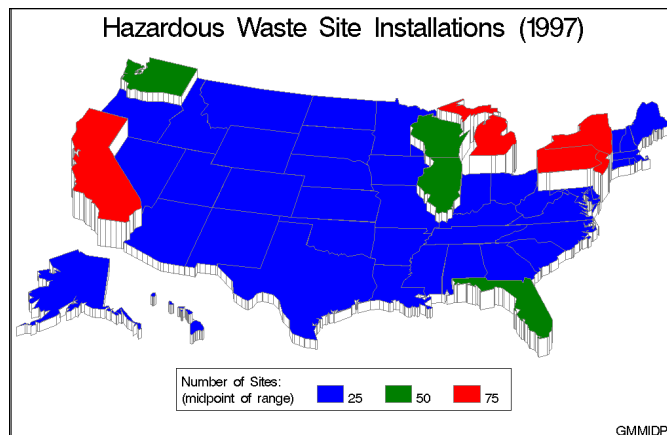
PRISM statement options:

LEGEND=
MIDPOINTS=
XLIGHT=
XVIEW=
ZVIEW=

Other features:

LEGEND statement

Sample library member: GMPMIDPT



This example explicitly specifies the midpoints for three response levels. Each response level uses the default solid pattern and a color from the colors list.

The example also changes the map viewpoint and light source.

Assign the libref and set the graphics environment. COLORS= specifies the colors list, which is used by the default patterns and outlines. CTEXT= specifies the color for all text.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(blue green red) ctext=black
          ftext=swiss htitle=6 htext=3;
```

Create response data set SITES. This data set contains a map area identification variable, STATE, and a response variable, SITES. The STFIPS function is used to convert the state postal codes to FIPS state codes. STATE contains the FIPS codes for each state and matches the values of STATE in the MAPS.US data set. SITES contains the total number of waste sites installed in the state.

```
data sites;
  length stcode $ 2;
```

```

input region stcode $ sites;
state=stfips(stcode);
datalines;
6 AR 12
10 AK 7...moredata lines...
3 WV 6
8 WY 3
;

```

Define title and footnote for map.

```

title1 'Hazardous Waste Site Installations (1997)';
footnotel j=r 'GMPMIDPT';

```

Define legend characteristics. CBORDER= draws a black frame around the legend. If FRAME were specified, it would be BLUE, the first color in the colors list.

```

legend shape=bar(4,4)
value=(j=1)
label=('Number of Sites:'
      j=1 '(midpoint of range)')
cborder=black;

```

Produce the prism map. MIDPOINTS= specifies three response levels for the map. XLIGHT= moves the light source to the right and adds shadows to the left-side polygons of the prisms. XVIEW= and ZVIEW= shift the viewing point to the right and upward, respectively. This reduces the number of prisms that are partially hidden by taller neighbors.

```

proc gmap map=maps.us data=sites;
id state;
prism sites / midpoints=25 50 75
           xlight=5
           xview=.75
           zview=5
           legend=legend
           coutline=gray;

run;
quit;

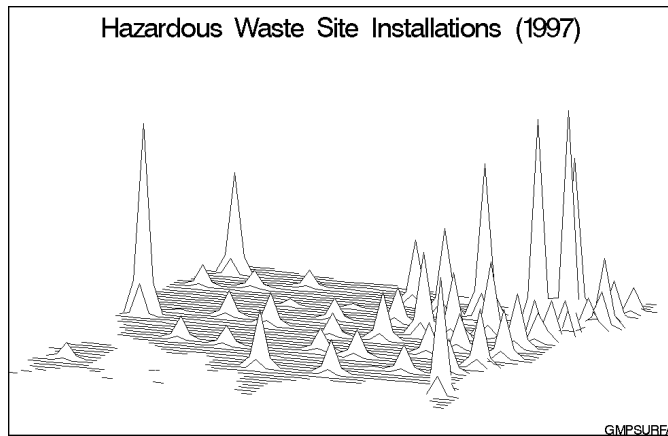
```

Example 9: Producing a Simple Surface Map

Procedure features:

SURFACE statement

Sample library member: GMPSURFA



This example produces a surface map that shows the total number of hazardous waste sites in each state in 1997. Because the `CONSTANT=` and `NLINES=` options are not used, the GMAP procedure draws a surface that consists of 50 lines and uses the default decay function to calculate spike height and base width. And because the `ROTATE=` and `TILT=` options are not used, the map is rotated 70 degrees around the Z axis and tilted 70 degrees with respect to the X axis.

Assign the libref and set the graphics environment. `COLORS=` specifies the colors list. By default the map uses the first color in the list.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(black blue green red)
          ftext=swiss htitle=6 htext=3;
```

Create response data set SITES. This data set contains a map area identification variable, `STATE`, and a response variable, `SITES`. The `STFIPS` function is used to convert the state postal codes to FIPS state codes. `STATE` contains the FIPS codes for each state and matches the values of `STATE` in the `MAPS.US` data set. `SITES` contains the total number of waste sites installed in the state.

```
data sites;
  length stcode $ 2;
  input region stcode $ sites;
  state=stfips(stcode);
  datalines;
6   AR  12
10  AK  7...moredata lines...
3   WV  6
8   WY  3
;
```

Define title and footnote for the map.

```
title1 'Hazardous Waste Site Installations (1997)';
footnote1 j=r 'GMPSURFA';
```

Produce the surface map. The ID statement specifies the variable in the map data set and the response data set that defines the map areas.

```
proc gmap map=maps.us data=sites;
  id state;
  surface sites;
run;
quit;
```

Example 10: Rotating and Tilting a Surface Map

Procedure features:

SURFACE statement options:

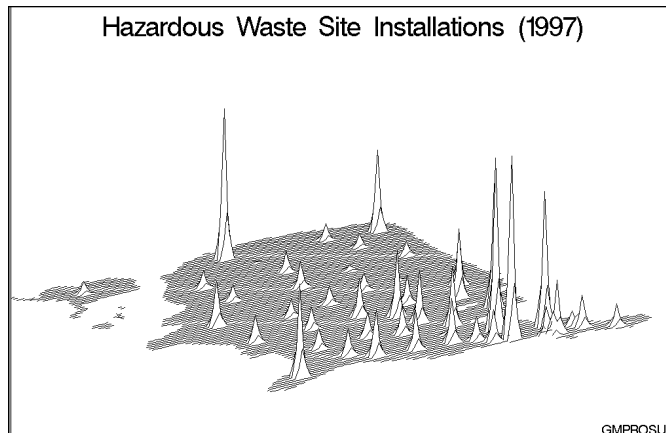
CONSTANT=

NLINES=

ROTATE=

TILT=

Sample library member: GMPROSUR



This example tilts and rotates the surface map and uses more lines to draw the surface.

Assign the libref and set the graphics environment.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
  colors=(black blue green red)
  ftext=swiss htitle=6 htext=3;
```

Create response data set SITES. This data set contains a map area identification variable, STATE, and a response variable, SITES. The STFIPS function is used to convert the state postal codes to FIPS state codes. STATE contains the FIPS codes for each state and matches the values of STATE in the MAPS.US data set. SITES contains the total number of waste sites installed in the state.

```

data sites;
  length stcode $ 2;
  input region stcode $ sites;
  state=stfips(stcode);
  datalines;
6   AR  12
10  AK  7...moredata lines...
3   WV  6
8   WY  3
;

```

Define title and footnote for the map.

```

title1 'Hazardous Waste Site Installations (1997)';
footnote1 j=r 'GMPROSUR';

```

Produce the surface map. CONSTANT= specifies a value that is less than the default value so the spikes are narrower at the base. NLINES= specifies the maximum number of map lines, which gives the best map shape resolution. ROTATE= and TILT= adjust the map orientation to make the crowded spikes in the northeast portion of the map easier to distinguish.

```

proc gmap map=maps.us data=sites;
  id state;
  surface sites / constant=4
                  nlines=100
                  rotate=40
                  tilt=60;
run;
quit;

```

Example 11: Creating a Map Using the Feature Table

Procedure Features:

- ID statement
- CHORO statement option:
 - DISCRETE

ODS Features:

- ODS HTML statement:
 - BODY=

Other Features:

- MERGE statement
- GOPTIONS statement

Sample library member: GMPSPATL

When you use a feature table on PROC GMAP, you merge the feature table with your response data set before generating a map, storing the combined data in a new data set.

On PROC GMAP, you use the DATA= option to name the combined data set, and you use the ID statement to identify the variable that contains the spatial information.

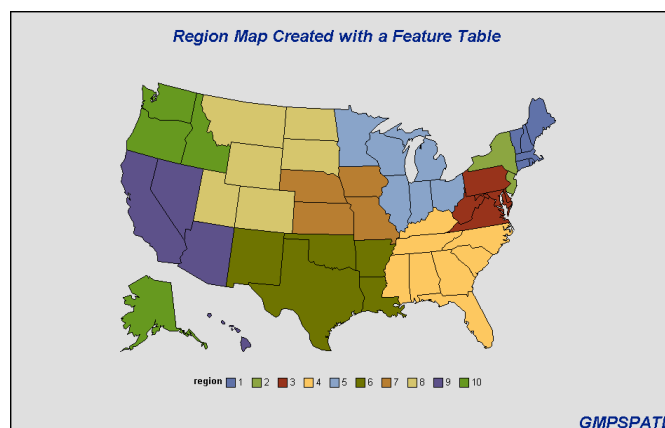
To illustrate the use of a feature table, assume you want to generate a map of the United States. Rather than using the traditional map data set MAPS.US, you want to use its corresponding feature table. To determine which feature table corresponds to a traditional map data set, look in the MAPS.METAMAPS data set:

- The feature table MAPS.US2 corresponds to the traditional map data set MAPS.US.
- In MAPS.US2, the values of the variable `_MAP_GEOMETRY_` encapsulate the geometry object.

The sample program uses the following procedures and statements:

- PROC SORT sorts the current data set, WORK.SITES, by the values of variable STATE. This prepares SITES for a merge with the feature table MAPS.US2, which is delivered with SAS/GRAPH. The variable STATE identifies the map areas in both SITES and MAPS.US2.
- PROC SORT sorts the feature table MAPS.US2. The OUT= option specifies that the sorted data be written to a new data set WORK.MAPS.
- In the DATA step, the MERGE statement merges the feature table with the response data. The combined data set is saved to a new data set named BOTH. The data set BOTH is stored in WORK, a temporary library. To use the combined data set in other SAS/GRAPH programs, you would need to save the merged data set to a permanent library.
- On the PROC GMAP statement, the DATA= option points to the combined data set, BOTH. The ID statement specifies `_MAP_GEOMETRY_` as the variable that contains the spatial data. Because both the map and response data are stored in a single data set, the MAP= option is not required on the PROC GMAP statement.

The following example creates the response data set SITES and merges it with the feature table US2. It then uses the combined data set to generate a map as a SAS/GRAPH Control for ActiveX.



Specify a valid file name and assign the libref for the SAS Maps library. This program generates one HTML file. FILENAME assigns the fileref ODSOUT, which specifies a destination for the HTML file that is produced by the sample program. 'External-html-file' needs to be replaced with the complete path specifying where the files will be located. The following two lines are the only lines that need to be changed to run the program.

```
filename odsout 'external-html-file' ;
libname maps 'SAS-data-library';
```

Create the data set SITES with regional data. Sites contains a region number for each state and the total number of hazardous waste sites in each state. The STFIPS function converts the state postal codes to FIPS state codes.

```
data sites;
  length stcode $ 2;
  input region stcode $ sites;
  state=stfips(stcode);
  datalines;
6   AR  12
10  AK  7
4   AL  12
9   AZ  10
9   CA  90
8   CO  15
1   CT  15
3   DE  18
4   FL  52
4   GA  15
9   HI  4
7   IA  16
10  ID  8
5   IL  38
5   IN  30
7   KS  10
4   KY  16
6   LA  15
1   MA  30
3   MD  13
1   ME  12
5   MI  72
5   MN  30
7   MO  22
4   MS  1
8   MT  8
4   NC  22
8   ND  0
7   NE  10
1   NH  18
2   NJ 105
6   NM  9
9   NV  1
2   NY  78
5   OH  34
6   OK  10
10  OR  10
3   PA 100
1   RI  12
4   SC  26
8   SD  2
4   TN  14
6   TX  26
```

```

8   UT  12
3   VA  25
1   VT   8
10  WA  49
5   WI  40
3   WV   6
8   WY   3
;

```

Sort the response and the feature tables in the order of the BY variable. By default, the first PROC SORT sorts the response data set created in the code above. Both sorted data sets are stored in the SAS temporary library WORK. To allow the data sets to be merged, the same BY variable is used to sort both the response and feature tables.

```

proc sort;
  by state;
run;

proc sort data=maps.us2 out=maps;
  by state;
run;

```

Merge the data sets.

```

data both;
  merge maps sites;
  by state;
run;

```

Specify the ACTIVEX driver and HTML output. To conserve system resources, ODS LISTING CLOSE closes the Listing destination for procedure output. In the program's ODS HTML statement, the BODY= option names the file for storing HTML output. ODSOUT is defined in the beginning of the code in the FILENAME statement.

```

goptions reset=all device=activex;
ods listing close;
ods html body='odsout.html'
path=odsout;

```

Define title and footnote for the map.

```

title1 'Region Map Created with a Feature Table';
footnotel j=r 'GMPSATL';

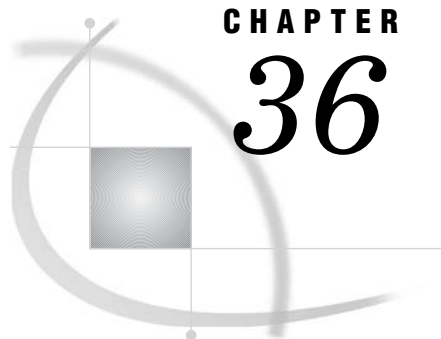
```

Generate the choropleth map using the merged response data set and feature table. The ID variable is the \$GEOREF formatted variable containing the spatial information. DISCRETE specifies that each level of REGION is a separate response level.

```
proc gmap data=both;
  id _map_geometry_;
  choro region/discrete;
run;
quit;
```

Close the HTML destination and open the listing destination. The HTML destination must be closed before you can view the output with a browser. ODS LISTING opens the Listing destination again so that the destination is again available for displaying output during this SAS session.

```
ods html close;
ods listing;
```

CHAPTER

36

The GOPTIONS Procedure

Overview 1075

Procedure Syntax 1076

PROC GOPTIONS Statement 1077

Examples 1078

Example 1: Displaying TITLE and FOOTNOTE Statements 1078

Example 2: Displaying Graphics Options without the Description 1079

Overview

The GOPTIONS procedure provides information about the values of graphics options and the global statement definitions that are currently in effect in your session. The values displayed are either the defaults of the current device driver or user-defined values that have been assigned in your SAS session. You can use the GOPTIONS procedure to

- list the current values of all of the graphics options, or of one specified option
- display the values of all of the AXIS, FOOTNOTE, LEGEND, PATTERN, SYMBOL, and TITLE definitions that are currently in effect.

Note: Do not confuse the GOPTIONS procedure with the GOPTIONS statement. The GOPTIONS procedure lists the values that are defined in a GOPTIONS statement as well as in any other global statement definitions. See “GOPTIONS Statement” on page 146 for a list of the graphics options that you can set with the GOPTIONS statement. See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for a complete description of each graphics option. Δ

The list of graphics options displays in the SAS LOG window and includes the names of the options, the current values, and a brief description of each one. You can use PROC GOPTIONS statement options to control what information is listed and where it appears in the LOG window. Output 36.1 contains part of a sample LOG listing.

Output 36.1 Partial Output from the GOPTIONS Procedure

```

                SAS/GRAPH software options and parameters
                (executing in DMS Programming Environment environment)
NOADMGDF          GDDM driver output an ADMGDF file
ASPECT=          Aspect ratio (width/height) for software characters
NOAUTOCOPY       Automatic hardcopy after display
NOAUTOFEED       Automatic paper feed after plot
NOAUTOSIZE       Change character cell size to preserve device
                  catalog rows and columns
BAUD=            Communications line speed
BINDING=NOBINDING Binding edge
NOBORDER         Draw a border around display or plot
CBACK=          Background color
CBY=            BY line color
CELL            Hardware characters must be on cell boundaries
CHARACTERS       Use hardware characters
CHARTYPE=       Select hardware font
CIRCLEARC       Use hardware circle/arc generator
NOCOLLATE        Collate output
COLORS=( )      Default color list
CPATTERN=       Default pattern color
CSYMBOL=        Default symbol color
CTEXT=          Default text color
CTITLE=         Default title, footnote and note color
DASH            Use hardware dashed line generator
DASHSCALE=      Dash pattern scale factor
DELAY=          Animation delay time in milliseconds
DEVADDR=        IBM Device address, qname, or node name
DEVICE=         Default device driver
DEVMAP=DEFAULT  Output character map for hardware text
DISPLAY         Display graph on device
DISPOSAL=NONE   Image animation disposal method
DRVINIT=        Host command executed before driver initialization
DRVTERM=        Host command executed after driver termination
NODUPLEX        Duplex printing
NOERASE         Erase graph upon completion
FASTTEXT        Use quicker, less precise, integer font rendering
                  routines; generally unsuitable for multiple device
                  or templated replay situations.

```

Note: All of the graphics options that are displayed by the GOPTIONS procedure are described in Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261. \triangle

Procedure Syntax

```
PROC GOPTIONS <option(s)>;
```

PROC GOPTIONS Statement

Lists the graphics options, and their values and descriptions in the LOG window. Optionally, it lists the currently defined global statements. By default, each listed item is displayed on a separate line.

Syntax

PROC GOPTIONS <*option(s)*>;

option(s) can be one or more options from the following categories:

- item request options
 - AXIS
 - FOOTNOTE
 - LEGEND
 - OPTION=*graphics-option*
 - PATTERN
 - SYMBOL
 - TITLE
- listing format options
 - CENTIMETERS
 - NOLIST
 - NOLOG
 - SHORT

Options

You can specify as many options as you want and list them in any order.

AXIS

A

requests a list of all current AXIS definitions. AXIS also lists the current values for all graphics options, unless you use the NOLIST option. If you have not defined any AXIS statements, the GOPTIONS procedure issues a message.

CENTIMETERS

CM

displays the values of the HORIGIN=, HSIZE=, PAPERFEED=, PAPERLIMIT=, VORIGIN=, and VSIZE= graphics options in units of centimeters (CM). These graphics options use units of IN or CM only, and their values are always stored as inches even if you specify CM. Therefore, the GOPTIONS procedure displays these values in inches, unless you specify the CENTIMETERS option.

Note: The CENTIMETERS option does not affect the graphics options that can use unit specifications of CELLS, CM, IN, PCT, and PT. △

FOOTNOTE

F

requests a list of all of the current FOOTNOTE and TITLE definitions. FOOTNOTE also lists the current values for all of the graphics options, unless you use the NOLIST option. If you have not defined any FOOTNOTE or TITLE statements, the GOPTIONS procedure issues a message.

Featured in: Example 1 on page 1078

LEGEND

L

requests a list of all of the current LEGEND definitions. LEGEND lists the current values for all of the graphics options, unless you use the NOLIST option. If you have not defined any LEGEND statements, the GOPTIONS procedure issues a message.

NOLIST

N

suppresses the display of graphics options. Use the NOLIST option in conjunction with the appropriate statement request option when you want to list only the current AXIS, FOOTNOTE, LEGEND, PATTERN, SYMBOL, or TITLE definitions.

Featured in: Example 1 on page 1078

NOLOG

displays the output in the OUTPUT window instead of the LOG window.

OPTION=*graphics-option*

requests information on the specified graphics option. For these options, requesting one displays the value of both:

- HSIZE= and VSIZE=
- HPOS= and VPOS=
- XMAX= and YMAX=
- XPIXELS= and YPIXELS=

PATTERN

P

requests a list of all of the current PATTERN definitions. PATTERN lists the current values for all of the graphics options, unless you use the NOLIST. If you have not defined any PATTERN statements, the GOPTIONS procedure issues a message.

SHORT

suppresses the descriptions of the graphics options and displays the graphics options values in an alphabetical list in paragraph form.

Featured in: Example 2 on page 1079

SYMBOL

S

requests a list of all of the current SYMBOL definitions. SYMBOL lists the current values for all of the graphics options, unless you use the NOLIST. If you have not defined any SYMBOL statements, the GOPTIONS procedure issues a message.

TITLE

T

requests a list of all of the current TITLE and FOOTNOTE definitions. TITLE lists the current values for all of the graphics options, unless you use the NOLIST option. If you have not defined any FOOTNOTE or TITLE statements, the GOPTIONS procedure issues messages.

Examples

Example 1: Displaying TITLE and FOOTNOTE Statements

Procedure features:

PROC GOPTIONS statement:
 FOOTNOTE
 NOLIST

Sample library member: GOPTIFT

This example uses the FOOTNOTE option to display the current definitions of both the FOOTNOTE and TITLE statements. It also uses the NOLIST option to suppress the list of graphics options. Output 36.2 shows the listing that appears in the LOG.

Output 36.2 Using the NOLIST Option (GOPTIFT)

```
TITLE1 HEIGHT=6 COLOR=BLUE FONT=SWISSB 'Production Quality' ;
TITLE2 HEIGHT=4 COLOR=BLUE FONT=SWISSB 'January through June';

FOOTNOTE1 HEIGHT=3 COLOR=GREEN FONT=SWISS 'Data from SASDATA.QUALITY' ;
FOOTNOTE2 HEIGHT=3 COLOR=GREEN FONT=SWISS '* denotes approximations' ;
```

Clear all global statements.

```
goptions reset=global;
```

Define titles and footnotes.

```
title1 h=6 c=blue f=swissb 'Production Quality';
title2 h=4 c=blue f=swissb 'January through June';
footnote1 h=3 c=green f=swiss 'Data from SASDATA.QUALITY';
footnote2 h=3 c=green f=swiss '* denotes approximations';
```

Produce the listing. The NOLIST and FOOTNOTE options control the information that appears in the LOG window.

```
proc goptions nolist footnote;
run;
```

Example 2: Displaying Graphics Options without the Description

Procedure features:

PROC GOPTIONS statement:
 SHORT

Sample library member: GOPSHORT

This example uses the SHORT option to display only the values of graphics options without the description of each graphics option. Output 36.3 shows the listing that appears in the LOG window.

Output 36.3 Using the SHORT Option (GOPSHORT)

```

                SAS/GRAPH software options and parameters
                (executing in DMS Programming Environment environment)
NOADMGDF ASPECT= NOAUTOCOPY NOAUTOFEED NOAUTOSIZE BAUD= BINDING=NOBINDING
BORDER CBACK= CBY= CELL CHARACTERS CHARTYPE= CIRCLEARC NOCOLLATE COLORS=( BLUE
GREEN RED ) CPATTERN=BLUE CSYMBOL= CTEXT=RED CTITLE=GREEN DASH DASHSCALE=
DELAY= DEVADDR= DEVICE= DEVMAP=DEFAULT DISPLAY DISPOSAL=NONE DRVINIT= DRVTERM=
NODUPLEX NOERASE FASTTEXT FBY= FCACHE=3 FILECLOSE= FILL FILLINC= FONTRES=NORMAL
FTEXT=SWISSB FTITLE= FTRACK=TIGHT GACCESS= GCLASS=G GCOPIES=(0, 20)
GDDMCOPY=FSCOPY GDDMNICKNAME= GDDMTOKEN= GDEST=LOCAL GEND= GEPILOG= GFORMS=
NOGOPT10 NOGOPT11 NOGOPT12 NOGOPT13 NOGOPT14 NOGOPT15 GOPTINT1=0 GOPTINT2=0
GOPTDBL1= GOPTDBL2= GOPTSTR1= GOPTSTR2= GOUTMODE=APPEND GOUTTYPE=INDEPENDENT
GPROLOG= GPROTOCOL= GRAPHRC GSFLEN= GSFMODE=PORT GSFNAME= NOGSFPROMPT GSIZE=
GSTART= GUNIT=PERCENT GWAIT= GWRITER=SASWTR HANDSHAKE= HBY=4 HORIGIN= HPOS=
HSIZE= HTEXT=3 HTITLE=6 INBIN= INTERPOL= ITERATION= NONINTERLACED
KEYMAP=DEFAULT LFACTOR= OFFSET= OFFSHADOW=(0.0625 in., -0.0625 in.) OUTBIN=
PAPERFEED= PAPERLIMIT= PAPERSIZE= PAPERTYPE= PENMOUNTS= PENSORT PIEFILL NOPCLIP
POLYGONCLIP POLYGONFILL POSTGEPILOG= POSTGRAPH= POSTGPROLOG= PPDFILE=
PREGEPILOG= PREGRAPH= PREGPROLOG= PROMPT PROMPTCHARS='000A010D05000000'X
RENDER=MEMORY RENDERLIB=WORK REPAINT= NOREVERSE NOROTATE SIMFONT= SPEED= NOSWAP
SYMBOL TARGETDEVICE= NOTRANSARENCY TRANTAB= UCC= NOUSERINPUT NOV5COMP NOV6COMP
VORIGIN= VPOS= VSIZE= XMAX= XPIXELS= YMAX= YPIXELS=

```

Set the graphics environment. The values of the graphics options specified in this statement appear in the LOG listing.

```

goptions reset=global gunit=pct border
        ftext=swissb htitle=6 htext=3
        ctext=red cpattern=blue ctitle=green
        colors=(blue green red) hby=4;

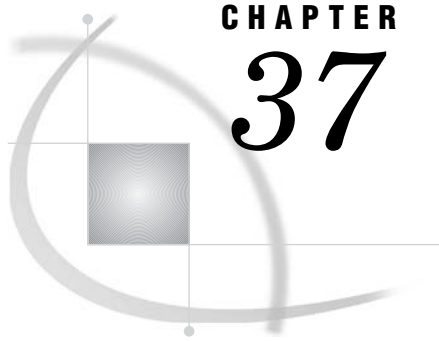
```

Produce the listing. The SHORT option suppresses the display of the description of each graphics option.

```

proc goptions short;
run;

```



CHAPTER 37

The GPLOT Procedure

<i>Overview</i>	1081
<i>About Plots of Two Variables</i>	1082
<i>About Plots with a Classification Variable</i>	1083
<i>About Bubble Plots</i>	1083
<i>About Plots with Two Vertical Axes</i>	1084
<i>About Interpolation Methods</i>	1085
<i>Concepts</i>	1085
<i>Parts of a Plot</i>	1085
<i>About the Input Data Set</i>	1086
<i>Missing Values</i>	1087
<i>Values Out of Range</i>	1087
<i>Sorted Data</i>	1087
<i>Logarithmic Axes</i>	1087
<i>Procedure Syntax</i>	1088
<i>PROC GPLOT Statement</i>	1088
<i>BUBBLE Statement</i>	1090
<i>BUBBLE2 Statement</i>	1098
<i>PLOT Statement</i>	1101
<i>PLOT2 Statement</i>	1115
<i>Examples</i>	1120
<i>Example 1: Generating a Simple Bubble Plot</i>	1120
<i>Example 2: Labeling and Sizing Plot Bubbles</i>	1122
<i>Example 3: Adding a Right Vertical Axis</i>	1124
<i>Example 4: Plotting Two Variables</i>	1126
<i>Example 5: Connecting Plot Data Points</i>	1129
<i>Example 6: Generating an Overlay Plot</i>	1131
<i>Example 7: Filling Areas in an Overlay Plot</i>	1134
<i>Example 8: Plotting Three Variables</i>	1135
<i>Example 9: Plotting with Different Scales of Values</i>	1138
<i>Example 10: Creating Plots with Drill-down for the Web</i>	1141

Overview

The GPLOT procedure plots the values of two or more variables on a set of coordinate axes (X and Y). The coordinates of each point on the plot correspond to two variable values in an observation of the input data set. The procedure can also generate a separate plot for each value of a third (classification) variable. It can also generate bubble plots in which circles of varying proportions representing the values of a third variable are drawn at the data points.

The procedure produces a variety of two-dimensional graphs including

- simple scatter plots
- overlay plots in which multiple sets of data points display on one set of axes
- plots against a second vertical axis
- bubble plots
- logarithmic plots (controlled by the AXIS statement).

In conjunction with the SYMBOL statement the GPLOT procedure can produce join plots, high-low plots, needle plots, and plots with simple or spline-interpolated lines. The SYMBOL statement can also display regression lines on scatter plots.

The GPLOT procedure is useful for

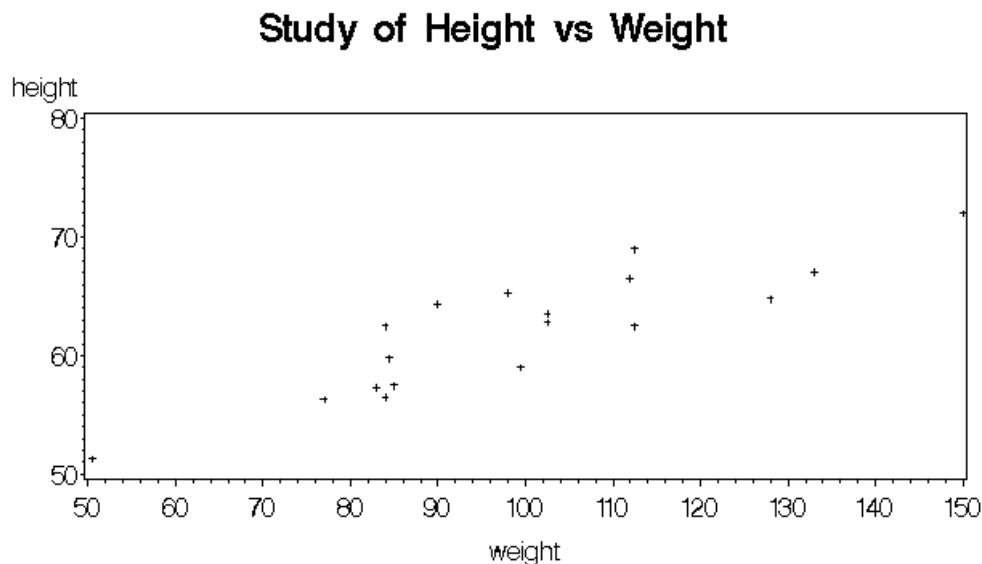
- displaying long series of data, showing trends and patterns
- interpolating between data points
- extrapolating beyond existing data with the display of regression lines and confidence limits.

About Plots of Two Variables

Plots of two variables display the values of two variables as data points on one horizontal axis (X) and one vertical axis (Y). Each pair of X and Y values forms a data point.

The following figure shows a simple scatter plot that plots the values of the variable HEIGHT on the vertical axis and the variable WEIGHT on the horizontal axis. By default, the PLOT statement scales the axes to include the maximum and minimum data values and displays a plus sign (+) at each data point. It labels each axis with the name of its variable or an associated label and displays the value of each major tick mark.

Figure 37.1 Scatter Plot of Two Variables (GPLVRBL1(a))



Source: T. Lewis & L. R. Taylor
Introduction to Experimental Ecology

GPLVREL1(a)

The program for this plot is in Example 4 on page 1126. For more information on producing scatter plots, see “PLOT Statement” on page 1101.

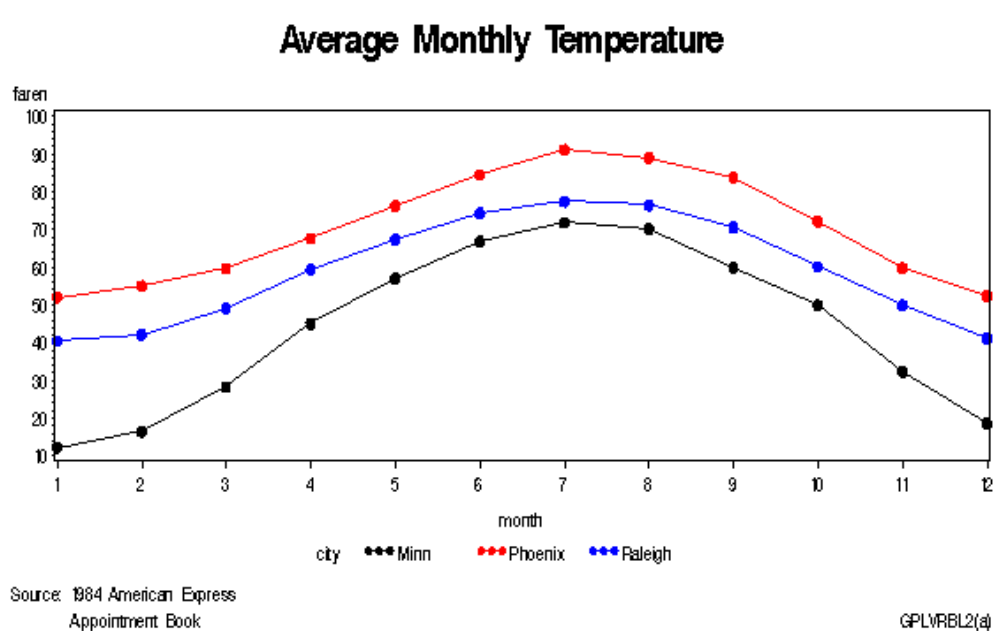
You can also overlay two or more plots (multiple sets of data points) on a single set of axes and you can apply a variety of interpolation techniques to these plots. See “About Interpolation Methods” on page 1085.

About Plots with a Classification Variable

Plots that use a classification variable produce a separate set of data points for each unique value of the classification variable and display all sets of data points on one set of axes.

The following figure shows multiple line plots that compare yearly temperature trends for three cities. The legend explains the values of the classification variable, CITY.

Figure 37.2 Plot of Three Variables with Legend (GPLVRBL2(a))



By default, plots with a classification variable generate a legend. In the code that generates the plot for Example 8 on page 1135, a `SYMBOL` statement connects the data points and specifies the plot symbol that is used for each value of the classification variable (CITY). The program for this plot is in Example 8 on page 1135. For more information on how to produce plots with a classification variable, see “PLOT Statement” on page 1101.

About Bubble Plots

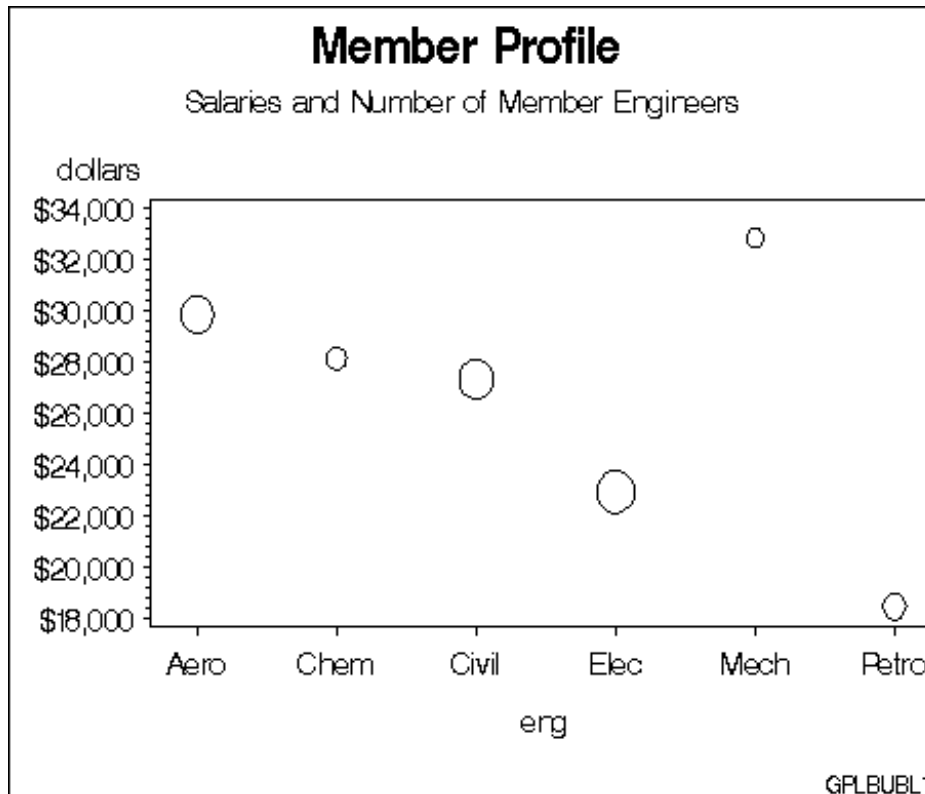
Bubble plots represent the values of three variables by drawing circles of varying sizes at points that are plotted on the vertical and horizontal axes. Two of the variables determine the location of the data points, while the values of the third variable control the size of the circles.

Figure 37.3 on page 1084 shows a bubble plot in which each bubble represents a category of engineer that is shown on the horizontal axis. The location of each bubble in

relation to the vertical axis is determined by the average salary for the category. The size of each bubble represents the number of engineers in the category relative to the total number of engineers in the data.

By default, the BUBBLE statement scales the axes to include the maximum and minimum data values and draws an unlabeled circle at each data point. It labels each axis with the name of its variable or an associated label and displays the value of each major tick mark.

Figure 37.3 Bubble Plot (GPLBUBL1)



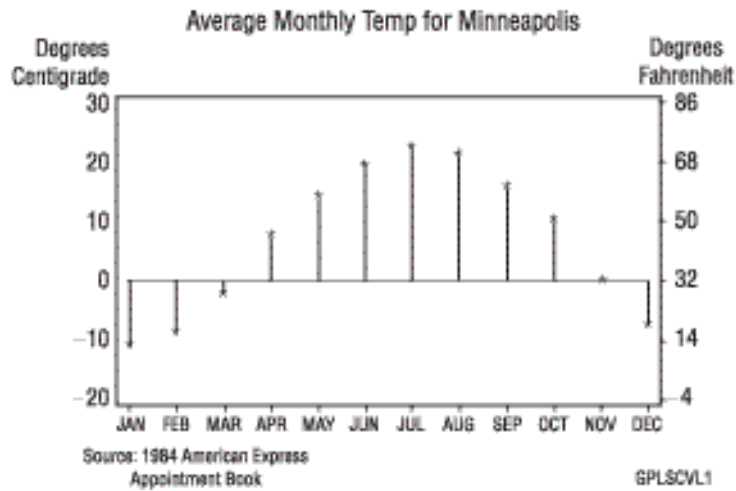
The program for this plot is in Example 1 on page 1120. For more information on producing bubble plots, see “BUBBLE Statement” on page 1090.

About Plots with Two Vertical Axes

Plots with two vertical axes have a right vertical axis that can

- display the same variable values as the left axis
- display left axis values in a different scale
- plot a second response (Y) variable, thereby producing one or more overlay plots.

In the following figure, the right axis displays the values of the vertical coordinates in a different scale from the scale that is used for the left axis.

Figure 37.4 Plot with a Right Vertical Axis (GPLSCVL1)

The program for this plot is in Example 9 on page 1138. For more information on how to produce plots with a right vertical axis, see “PLOT2 Statement” on page 1115 and “BUBBLE2 Statement” on page 1098.

About Interpolation Methods

In addition to these graphs, you can produce other types of plots such as box plots or high-low-close plots by specifying various interpolation methods with the SYMBOL statement. Use the SYMBOL statement to

- connect the data points with straight lines
- specify regression analysis to fit a line to the points and, optionally, display lines for confidence limits
- connect the data points to the zero line on the vertical axis
- display the minimum and maximum values of Y at each X value and mark the mean value, display standard deviations that connect the data points with lines or bars, generate box plots, or plot high-low-close stock market data
- specify that a pattern fill the polygon that is defined by data points
- smooth plot lines with spline interpolation
- use a step function to connect the data points

“SYMBOL Statement” on page 183 describes all interpolation methods.

Concepts

Parts of a Plot

Some terms used with GPLOT procedure are illustrated in Figure 37.5 on page 1086 and Figure 37.6 on page 1086.

Figure 37.5 GPlot Procedure Terms

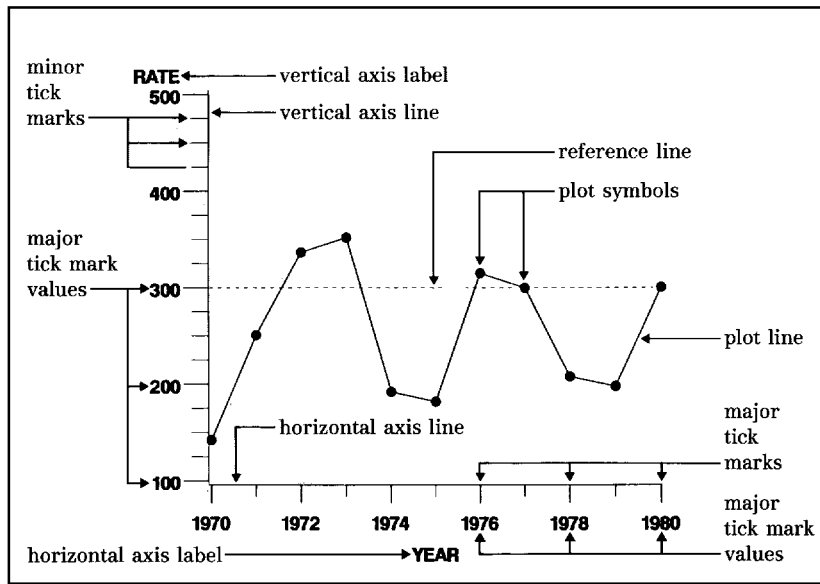
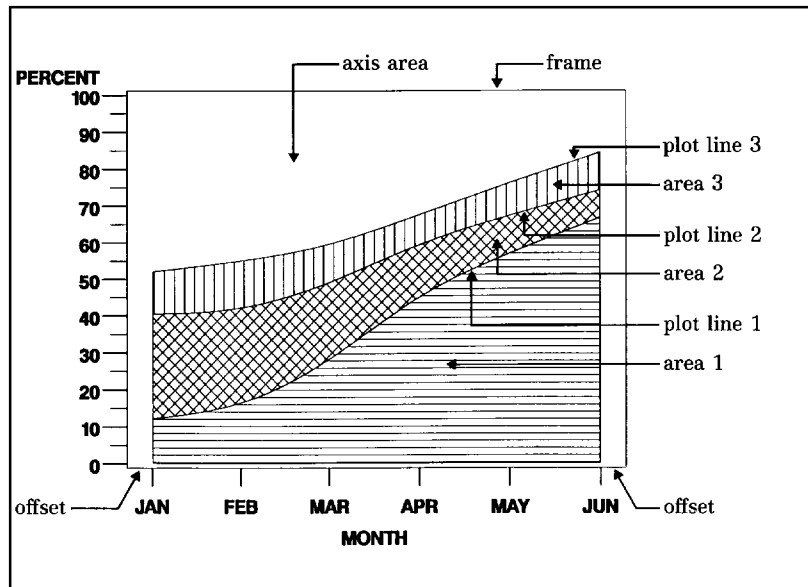


Figure 37.6 Additional GPlot Procedure Terms



About the Input Data Set

The input data set that is used by the GPlot procedure must contain at least one variable to plot on the horizontal axis and one variable to plot on the vertical axis. Typically, the horizontal axis shows an independent variable (time, for example), and the vertical axis shows a dependent variable (temperature, for example). Variables can

be character or numeric. Graphs are automatically scaled to the values of the character data or to include the values of numeric data, but you can control scaling with procedure options or with associated **AXIS** statements.

Missing Values

If the value of either of the plot variables is missing, the GPLOT procedure does not include the observation in the plot. If you specify interpolation with a **SYMBOL** definition, the plot is not broken at the missing value. To break the plot line or area fill at the missing value, use the **PLOT** statement's **SKIPMISS** option. **SKIPMISS** is enabled only for **JOIN** interpolations.

Values Out of Range

Exclude data values from a graph by restricting the range of axis values with the **VAXIS=** or **HAXIS=** options or with the **ORDER=** option in an **AXIS** statement. When an observation contains a value outside of the specified axis range, the GPLOT procedure excludes the observation from the plot and issues a message to the log.

If you specify interpolation with a **SYMBOL** definition, by default values outside of the axis range are excluded from interpolation calculations and as a result may change interpolated values for the plot. Values that are omitted from interpolation calculations have a particularly noticeable effect on the high-low interpolation methods: **HILO**, **STD**, and **BOX**. In addition, regression lines and confidence limits will represent only part of the original data.

To specify that values out of range are included in the interpolation calculations, use the **MODE=** option in a **SYMBOL** statement. When **MODE=INCLUDE**, values that fall outside of the axis range are included in interpolation calculations but excluded from the plot. The default (**MODE=EXCLUDE**) omits observations that are outside of the axis range from interpolation calculations. See the **MODE=** option of the **SYMBOL** statement in "SYMBOL Statement" on page 183 for details.

Sorted Data

Data points are plotted in the order in which the observations are read from the data set. Therefore, if you use any type of interpolation that generates a line, sort your data by the horizontal axis variable.

Logarithmic Axes

If your data contain logarithmic values or if the data values vary over a wide range or contain large values, you may want to specify a logarithmic axis for the horizontal or vertical axis. Logarithmic axes can be specified with the **AXIS** statement options **LOGBASE=** and **LOGSTYLE=**. See "AXIS Statement" on page 124 for a complete discussion.

Procedure Syntax

Requirements: At least one PLOT or BUBBLE statement is required. A PLOT2 or BUBBLE2 statement can be used in conjunction with a PLOT or BUBBLE statement.

Global statements: AXIS“AXIS Statement” on page 124, FOOTNOTE“TITLE, FOOTNOTE, and NOTE Statements” on page 210, GOPTIONS“GOPTIONS Statement” on page 146, LEGEND“LEGEND Statement” on page 151, PATTERN“PATTERN Statement” on page 169, SYMBOL“SYMBOL Statement” on page 183, TITLE“TITLE, FOOTNOTE, and NOTE Statements” on page 210

Reminder: The procedure can include BY, FORMAT, LABEL, WHERE, and NOTE statements.

Supports: RUN-group processing Output Delivery System (ODS)

```
PROC GPLOT <DATA=input-data-set>
  <ANNOTATE=Annotate-data-set>
  <GOUT=<libref.>output-catalog>
  <IMAGEMAP=output-data-set >
  <UNIFORM>;

  BUBBLE plot-request(s) </option(s)>;
  BUBBLE2 plot-request(s) </option(s)>;

  PLOT plot-request(s) </option(s)>;
  PLOT2 plot-request(s) </option(s)>;
```

PROC GPLOT Statement

Identifies the data set that contains the plot variables. Optionally specifies uniform axis scaling for all graphs as well as annotation and an output catalog.

Requirements: An input data set is required.

Syntax

```
PROC GPLOT <DATA=input-data-set>
  <ANNOTATE=Annotate-data-set>
  <GOUT=<libref.>output-catalog>
  <IMAGEMAP=output-data-set >
  <UNIFORM>;
```

Options

ANNOTATE=Annotate-data-set

ANNO=Annotate-data-set

specifies a data set to annotate all graphs that are produced by the GPLOT procedure. To annotate individual graphs, use ANNOTATE= in the action statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

DATA=*input-data-set*

specifies the SAS data set that contains the variables to plot. By default, the procedure uses the most recently created SAS data set.

See also: “SAS Data Sets” on page 29 and “About the Input Data Set” on page 1086.

GOUT=< *libref.* >*output-catalog*

specifies the SAS catalog in which to save the graphics output that is produced by the GPLOT procedure. If you omit the libref, SAS/GRAPH looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist.

See also: “Storing Graphics Output in SAS Catalogs” on page 53.

IMAGEMAP=*output-data-set*

creates a temporary SAS data set that is used to generate an image map in an HTML output file. The IMAGEMAP= option can be used only if the PLOT or PLOT2 statements are used, and the PLOT or PLOT2 statement must use the HTML= option or the HTML_LEGEND= option or both.

If HTML= is used on the PLOT or PLOT2 statement, the plot points are defined as hot zones, unless AREA= is also used, in which case there are not plot points and the areas between plot lines are defined as hot zones. If HTML_LEGEND= is used, the legend symbols are defined as hot zones. Information for the links is stored in the variables referenced by the HTML= and/or HTML_LEGEND= options.

The %IMAGEMAP macro generates the image map in the HTML output file. The macro takes two arguments, the name of the image map data set and the name or fileref of the HTML output file, as shown in the following example:

```
%imagemap(imgmapds, myimgmap.html);
```

UNIFORM

specifies that the same axis scaling is used for all graphs that are produced by the procedure. By default, the range of axis values for each axis is based on the minimum and maximum values in the data and, therefore, may vary from graph to graph and among BY groups. Using the UNIFORM option forces the value range for each axis to be the same for all graphs. Thus, if the procedure produces multiple graphs with both left and right vertical axes, the UNIFORM option scales all of the left axes the same and all of the right axes the same, based on the minimum and maximum data values.

In addition, UNIFORM forces the assignment of SYMBOL statements for the category variable without regard to the BY-group variable, and, if a legend is generated, makes the legend the same across graphs.

Not supported by: Java, ActiveX

BUBBLE Statement

Creates bubble plots in which a third variable is plotted against two variables represented by the horizontal and vertical axes; the value of the third variable controls the size of the bubble.

Requirements: At least one plot request is required.

Global statements: AXIS“AXIS Statement” on page 124, FOOTNOTE“TITLE, FOOTNOTE, and NOTE Statements” on page 210, TITLE“TITLE, FOOTNOTE, and NOTE Statements” on page 210

Description

The BUBBLE statement specifies one or more plot requests that name the horizontal and left vertical axis variables and the variable that controls the size of the bubbles. This statement automatically

- centers each circle at a data point that is determined by the values of the vertical and horizontal axes variables
- scales the axes to include the maximum and minimum data values
- labels each axis with the name of its variable or associated label
- displays each major tick mark value
- draws circles for values that are located within the axes.

You can use statement options to control axis scaling, draw reference lines, modify the appearance of axes, control the display of the bubbles, specify a backplane color or image, and specify annotation.

In addition, you can use global statements to modify axes (AXIS statement), and add text to the graph (TITLE, NOTE, and FOOTNOTE statements). You can also use the Annotate data set to enhance the plot.

Syntax

BUBBLE *plot-request(s)* <*option(s)*>;

option(s) can be one or more options from any or all of the following categories:

- bubble appearance options:
 - BCOLOR=*bubble-color*
 - BFONT=*font*
 - BLABEL
 - BSCALE=AREA | RADIUS
 - BSIZE=*multiplier*
- plot appearance options:
 - ANNOTATE=*Annotate-data-set*
 - CAXIS=*axis-color*
 - CFRAME=*background-color*
 - CTEXT=*text-color*
 - FRAME | NOFRAME
 - GRID
 - HREVERSE
 - IFRAME= *fileref* | '*external-file*'
 - IMAGESTYLE = TILE | FIT
 - NOAXIS
- horizontal axis options:
 - AUTOHREF

CAUTOHREF=*reference-line-color*
 CHREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*
 HAXIS=*value-list* | AXIS<1...99>
 HMINOR=*number-of-minor-ticks*
 HREF=*value-list*
 HZERO
 LAUTOHREF=*reference-line-type*
 LHREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*

□ vertical axis options:

AUTOVREF
 CAUTOVREF=*reference-line-color*
 CVREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*
 LAUTOVREF=*reference-line-type*
 LVREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*
 VAXIS=*value-list* | AXIS<1...99>
 VMINOR=*number-of-minor-ticks*
 VREF=*value-list*
 VREVERSE
 VZERO

□ catalog entry description options:

DESCRIPTION=*'entry-description'*
 NAME=*'entry-name'*

Required Arguments

plot-request(s)

each specifies the variables to plot and produces a separate graph. All variables must be in the input data set. Multiple plot requests are separated with blanks. A plot request must have this form:

*y-variable***x-variable*=*bubble-size*

plots the values of two variables and draws a circle (bubble) at each data point. The value of the third variable determines the size of the bubble.

y-variable

variable plotted on the left vertical axis.

x-variable

variable plotted on the horizontal axis.

bubble-size

variable that dictates the size of the bubbles. *Bubble-size* must be numeric. If the value of *bubble-size* is positive, bubbles are drawn with a solid line; if it is negative, bubbles are drawn with a dashed line.

Note: If you specify the JAVA, JAVAMETA, or JAVAIMG device drivers, then either the *x-variable* or the *y-variable* must be numeric. △

Options

Options in a BUBBLE statement affect all graphs that are produced by that statement. You can specify as many options as you want and list them in any order.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate plots that are produced by the BUBBLE statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

AUTOHREF

draws reference lines at all major tick marks on the horizontal axis. To specify line types for these reference lines, use the LAUTOHREF= option. To specify colors for these reference lines, use the CAUTOHREF= option. To specify labels for these reference lines, use the HAXIS= option.

AUTOVREF

draws reference lines at all major tick marks on the vertical axis. To specify line types for these reference lines, use the LAUTOVREF= option. To specify colors for these reference lines, use the CAUTOVREF= option. To specify labels for these reference lines, use the VAXIS= option.

BCOLOR=*bubble-color*

specifies the color for the bubbles. If you omit the BCOLOR= option, the first color in the colors list is used for the bubble color.

Featured in: Example 2 on page 1122 and Example 3 on page 1124.

BFONT=*font*

specifies the font to use for bubble labels. See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for details on how to specify *font*. If you omit the BFONT= option, a font specification is searched for in this order:

- 1 the FTEXT= option in a GOPTIONS statement
- 2 the default hardware font.

See also: The BLABEL option for information on the location and color of labels.

Featured in: Example 2 on page 1122.

Not supported by: Java, ActiveX

BLABEL

labels the bubbles with the values of the third variable. If the variable has a format, the formatted value is used. By default, bubbles are not labeled.

The procedure normally places labels directly outside of the circle at 315 degrees rotation. If a label in this position does not fit in the axis area, other 45-degree placements (that is, 45, 135, and 225 degrees) are attempted. If the label cannot be placed at any of the positions (45, 135, 225, or 315 degrees) without being clipped, the label is omitted. However, labels may collide with other bubbles or previously placed labels.

Labels display in the color specified by the CTEXT= option. If you omit CTEXT=, the default is the first color in the colors list.

Featured in: Example 2 on page 1122.

BSCALE=AREA | RADIUS

specifies whether the bubble-scaling proportion is based on the area of the circles or the radius measure. By default, BSCALE=AREA.

The value that is assigned to the BSCALE= option affects how large the bubbles appear in relation to each other. For example, suppose the third variable value is twice as big for one bubble as it is for another. If BSCALE=AREA, the area of the larger bubble will be twice the area of the smaller bubble. If BSCALE=RADIUS, the

radius of the larger bubble will be twice the radius of the smaller bubble and the larger bubble will have more than twice the area of the smaller bubble.

Not supported by: Java, ActiveX

BSIZE=multiplier

specifies an overall scaling factor for the bubbles so that you can increase or decrease the size of all bubbles by this factor. By default, BSIZE=5. If you specify BSIZE=0, then the default size is used instead.

In Web output, the Java applets and the ActiveX Control override the default value. To prevent this override, specify a value for the BSIZE= option, rather than relying on the default value.

Featured in: Example 2 on page 1122 and Example 2 on page 1122.

Not supported by: Java (partial), ActiveX (partial)

CAUTOHREF=reference-line-color

specifies colors for reference lines drawn at major tick marks on the horizontal axis, as specified by the AUTOHREF option. The default color is either the value of the CAXIS= option or the first color in the color list. To specify line types for these reference lines, use the LAUTOHREF= option. To specify labels for these reference lines, use the HAXIS= option.

CAUTOVREF=reference-line-color

specifies colors for reference lines drawn at major tick marks on the vertical axis, as specified by the AUTOVREF option. The default color is either the value of the CAXIS= option or the first color in the color list. To specify line types for these reference lines, use the LAUTOVREF= option. To specify labels for these reference lines, use the VAXIS= option.

CAXIS=axis-color

CA=axis-color

specifies the color for the axis line and all major and minor tick marks. By default, the procedure uses the first color in the colors list.

If you use the CAXIS= option, it may be overridden by

- 1 the COLOR= option in an AXIS definition, which in turn is overridden by
- 2 the COLOR= suboption of the MAJOR= or MINOR= option in an AXIS definition.

Featured in: Example 2 on page 1122 and Example 3 on page 1124.

CFRAME=background-color

CFR=background-color

fills the axis area with the specified color. If the FRAME option is also in effect, the procedure determines the color of the frame according to the precedence list given for the FRAME option description. If the IFRAME= option is in effect, the specified image fills the axis area instead of the specified color.

CHREF=reference-line-color | (reference-line-color)

CH=reference-line-color | (reference-line-color)

specifies the color of reference lines drawn perpendicular to the horizontal axis. This option affects reference lines drawn with the AUTOHREF, HREF, and GRID options. Specifying a single color without parentheses applies that color to all reference lines. The CAUTOHREF= option overrides the CHREF= option for lines drawn with the AUTOHREF option. Specifying a single color in parentheses applies that color only to the first reference line drawn with the HREF= option. Specifying a color list applies colors sequentially to successive reference lines drawn with the HREF= option. The syntax of the color list is of the form (*color1 color2... colorN*) or (*color1, color2..., colorN*). Default colors for reference lines are determined by the CAXIS= option or by

the first color in the color list. To specify line types for these reference lines, use the LHREF= option. To specify labels for these reference lines, use the HAXIS= option.

CTEXT=*text-color*

C=*text-color*

specifies the color for all text on the axes, including tick mark values, axis labels, and bubble labels.

If you omit the CTEXT= option, a color specification is searched for in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the default, the first color in the colors list.

If you use the CTEXT= option, it overrides the color specification for the axis label and the tick mark values in the COLOR= option in an AXIS definition that is assigned to the axis.

If you use CTEXT=, the color specification is overridden in this situation: if you also use the COLOR= suboption of a LABEL= or VALUE= option in an AXIS definition that is assigned to the axis, that suboption determines the color of the axis label or the color of the tick mark values, respectively.

CVREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*

CV=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*

specifies the color of reference lines drawn perpendicular to the vertical axis. This option affects reference lines drawn with the AUTOVREF, VREF, and GRID options. Specifying a single color without parentheses applies that color to all reference lines. The CAUTOVREF= option overrides the CVREF= option for lines drawn with the AUTOVREF option. Specifying a single color in parentheses applies that color only to the first reference line drawn with the VREF= option. Specifying a color list applies colors sequentially to successive reference lines drawn with the VREF= option. The syntax of the color list is of the form (*color1 color2... colorN*) or (*color1, color2..., colorN*). Default colors for reference lines are determined by the CAXIS= option or by the first color in the color list. To specify line types for these reference lines, use the LVREF= option. To specify labels for these reference lines, use the VAXIS= option.

DESCRIPTION='*entry-description*'

DES='*entry-description*'

specifies the description of the catalog entry for the plot. The maximum length for *entry-description* is 256 characters. The description does not appear on the plot. By default, the procedure assigns a description of the form BUBBLE OF *variable*variable=variable*.

The *entry-description* can include the #BYLINE, #BYVAL, and #BYVAR substitution options, which work as they do when used on TITLE, FOOTNOTE, and NOTE statements. For more information, refer to the description of the option on page 222, and the discussion of "Substituting BY Line Values in a Text String" on page 226. The 256-character limit applies before the substitution takes place for these options; thus, if in the SAS program the entry-description text exceeds 256 characters, it is truncated to 256 characters, and then the substitution is performed.

The descriptive text is shown in the "description" portion of each of the following:

- in the Results window
- among the catalog-entry properties that you can view from the Explorer window
- in the Table of Contents that is generated when you use CONTENTS= option on an ODS statement, assuming that the GPLOT output is generated while the contents page is open. See "Linking to Output through a Table of Contents" on page 495.
- in the Description field of the PROC GREPLAY window

FRAME | NOFRAME**FR | NOFR**

specifies whether a frame is drawn around the axis area. The default is FRAME; however, if the V6COMP option is in effect on the GOPTIONS statement, the default is NOFRAME. If you also use a BUBBLE2 or PLOT2 statement and your plotting statements have conflicting frame specifications, FRAME is used.

For the frame color, a specification is searched for in this order:

- 1 the CAXIS= option
- 2 the COLOR= option in the AXIS definition assigned to the vertical axis
- 3 the COLOR= option in the AXIS definition assigned to the horizontal axis
- 4 the default, the first color in the colors list.

To fill the axis area with a background color, use the CFRAME= option.

To fill the axis area with a background image, use the IFRAME= option.

GRID

draws reference lines at all major tick marks on both axes. You get the same result when you use all of these options in a BUBBLE statement: AUTOHREF, AUTOVREF, FRAME, LVREF=34, and LHREF=34. The line type for GRID is 34.

The line color is the color of the axis.

HAXIS=*value-list* | AXIS<1 . . . 99>

specifies major tick mark values for the horizontal axis or assigns an axis definition. For a description of *value-list*, see the HAXIS= on page 1108 option for the PLOT statement. To assign labels to horizontal reference lines, specify an axis definition that contains the REFLABEL= option. Labels will be applied in sequence to all reference lines drawn with the AUTOHREF and HREF= options.

If you assign an axis definition that does not currently exist, the option is ignored. By default, the procedure scales the axis and provides an appropriate number of tick marks.

If data values fall outside of the range that is specified by the HAXIS= option, then by default the outlying data values are not used in interpolation calculations.

For Web output that is generated with a Java or ActiveX device driver, certain options of the AXIS statement are not supported. For details, see “AXIS Statement” on page 124.

See also: “About the Input Data Set” on page 1086 for more information on values out of range.

Featured in: Example 2 on page 1122.

Not supported by: Java (partial), ActiveX (partial)

HMINOR=*number-of-minor-ticks***HM=*number-of-minor-ticks***

specifies the number of minor tick marks that are drawn between each major tick mark on the horizontal axis. Minor tick marks are not labeled. The HMINOR= option overrides the NUMBER= suboption of the MINOR= option in an AXIS definition. You must specify a positive number.

Featured in: Example 2 on page 1122.

HREF=*value-list*

draws one or more reference lines perpendicular to the horizontal axis at points that are specified by *value-list*. For a description of *value-list* see the HAXIS= on page 1108 option for the PLOT statement. To specify colors for these reference lines, use the CHREF= option. To specify line types for these reference lines, use the LHREF= option. To specify labels for these reference lines, use the HAXIS= option.

HREVERSE

specifies that the order of the values on the horizontal axis be reversed. For Web output that is generated with a Java device driver, the horizontal axis data must be numeric.

Not supported by: Java (partial)

HZERO

specifies that tick marks on the horizontal axis begin in the first position with a value of zero. The HZERO request is ignored if negative values are present for the horizontal variable or if the horizontal axis has been specified with the HAXIS= option.

IFRAME=*fileref* | '*external-file*'

identifies the image file you wish to apply to the backplane of the plot. See also the IMAGESTYLE= option and “Placing a Backplane Image on Graphs with Frames” on page 115. The IFRAME= option is overridden by the NOIMAGEPRINT option “IMAGEPRINT” on page 318.

Not supported by: Java

IMAGESTYLE= TILE | FIT

specifies whether to tile the image to fill the backplane or to stretch the image to fit the backplane. The TILE value is the default. See also the IFRAME= option.

LAUTOHREF=*reference-line-type*

specifies a line type for reference lines drawn at major tick marks on the horizontal axis, as specified by the AUTOHREF option. The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. The default value 1 draws a solid line. To specify colors for these reference lines, use the CAUTOHREF= option. To specify labels for these reference lines, use the HAXIS= option.

LAUTOVREF=*reference-line-type*

specifies a line type for reference lines drawn at major tick marks on the vertical axis, as specified by the AUTOVREF option. The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. The default value 1 draws a solid line. To specify colors for these reference lines, use the CAUTOVREF= option. To specify labels for these reference lines, use the VAXIS= option.

LHREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list***LH=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list***

specifies line types for reference lines drawn perpendicular to the horizontal axis. The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. This option affects reference lines drawn with the AUTOHREF, HREF, and GRID options. Specifying a single line type without parentheses applies that line type to all reference lines. The LAUTOHREF= option overrides the LHREF= option for lines drawn with the AUTOHREF option. Specifying a single line type in parentheses applies that line type only to the first line drawn with the HREF= option. Specifying a line-type list applies line types in sequence to successive reference lines drawn with the HREF= option. The syntax of the line type list is of the form (*type1 type2... typeN*). The default value 1 draws a solid line. To specify colors for these reference lines, use the CHREF= option. To specify labels for these reference lines, use the HAXIS= option.

LVREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list***LV=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list***

specifies line types for reference lines drawn perpendicular to the vertical axis. The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a

solid line; values 2 through 46 specify dashed lines. This option affects reference lines drawn with the AUTOVREF, VREF, and GRID options. Specifying a single line type without parentheses applies that line type to all reference lines. The LAUTOVREF= option overrides the LVREF= option for lines drawn with the AUTOVREF option. Specifying a single line type in parentheses applies that line type only to the first line drawn by the VREF= option. Specifying a line-type list applies line types in sequence to successive reference lines drawn with the VREF= option. The syntax of the line type list is of the form (*type1 type2... typeN*). The default value 1 draws a solid line. To specify colors for these reference lines, use the CVREF= option. To specify labels for these reference lines, use the VAXIS= option.

NAME='entry-name'

specifies the name of the catalog entry for the graph. The maximum length for *entry-name* is eight characters. The default name is GPLOT. If you specify DEVICE=ACTIXIMG or DEVICE=JAVAIMG, then the name that you specify will be used for the Java or ActiveX device driver image output even if the file exists. For all other devices, if the name duplicates an existing entry name, SAS/GRAPH adds a number to the duplicate name to create a unique entry, for example, GPLOT1.

NOAXIS

NOAXES

suppresses the axes, including axis lines, axis labels, all major and minor tick marks, and tick mark values.

VAXIS=value-list | AXIS<1...99>

specifies the major tick mark values for the vertical axis or assigns an axis definition. For a description of the *value-list*, see the HAXIS= option on page 1108 of the PLOT statement. To assign labels to reference lines, specify an axis definition that contains the REFLABEL= option. Labels will be applied in sequence to all reference lines defined with the AUTOVREF and VREF= options.

For Web output that is generated with a Java or ActiveX device driver, certain options of the AXIS statement are not supported. For details, see "AXIS Statement" on page 124.

Featured in: Example 2 on page 1122 and Example 3 on page 1124.

Not supported by: Java (partial), ActiveX (partial)

VMINOR=number-of-minor-ticks

VM=number-of-minor-ticks

specifies the number of minor tick marks that are drawn between each major tick mark on the vertical axis. Minor tick marks are not labeled. VMINOR= overrides the NUMBER= suboption of the MINOR= option in an AXIS definition. You must specify a positive number.

Featured in: Example 2 on page 1122.

VREF=value-list

draws one or more reference lines perpendicular to the vertical axis at points that are specified by *value-list*. For a description of the *value-list*, see the HAXIS= option on page 1108 of the PLOT statement. To specify colors for reference lines, use the CVREF= option. To specify line types for these reference lines, use the LVREF= option. To specify labels for these reference lines, use the VAXIS= option.

VREVERSE

specifies that the order of the values on the vertical axis should be reversed.

VZERO

specifies that tick marks on the vertical axis begin in the first position with a zero. The VZERO request is ignored if the vertical variable either contains negative values

or has been ordered with the VAXIS= option or the ORDER= option in an AXIS statement.

Controlling the Display of Bubbles

The BUBBLE statement draws circles only for values that are located within the axes. Observations with values that lie outside of the axis area are not plotted. If a bubble size value causes a bubble to overlap the axis, the bubble is clipped against the axis line. The bubbles for the highest axis value and lowest axis value may be clipped unless you modify the axes in either of the following ways:

- by offsetting the first and last values
- by adding values to the range that is represented by the axis.

Specify the range of values on an axis with the HAXIS= or VAXIS= option, or with AXIS definitions.

To add a right vertical axis, use a BUBBLE2 statement.

BUBBLE2 Statement

Creates a second vertical axis on the right side of a graph produced by an accompanying BUBBLE or PLOT statement. A second variable can be plotted against this axis.

Requirements: You cannot use the BUBBLE2 statement alone. You can use it only with a BUBBLE or PLOT statement. At least one plot request is required.

Global statements: AXIS“AXIS Statement” on page 124, FOOTNOTE“TITLE, FOOTNOTE, and NOTE Statements” on page 210, TITLE“TITLE, FOOTNOTE, and NOTE Statements” on page 210

Description

The BUBBLE2 statement specifies one or more plot requests that name the horizontal and right vertical axis variables and the variable that controls the size of the bubbles. This statement automatically

- scales the axes to include the maximum and minimum data values
- labels each axis with the name of its variable or an associated label
- displays each major tick mark value
- draws circles for values that are located within the axes.

You can use statement options to control right vertical axis scaling, draw reference lines on the right vertical axis, control the display of the bubbles, display a background color or image, and specify annotation.

In addition, you can use global statements to modify the axes (AXIS statement), and add text to the graph (TITLE, NOTE, and FOOTNOTE statements). You can also use the Annotate data set to enhance the plot.

Syntax

BUBBLE2 *plot-request(s)* </option(s)>;

option(s) can be one or more options from any or all of the following categories:

- bubble appearance options:

- BCOLOR=*bubble-color*
- BFONT=*font*
- BLABEL
- BSCALE=AREA | RADIUS
- BSIZE=*multiplier*
- plot appearance options:
 - ANNOTATE=*Annotate-data-set*
 - CAXIS=*axis-color*
 - CFRAME=*background-color*
 - CTEXT=*text-color*
 - FRAME | NOFRAME
 - GRID
 - NOAXIS | NOAXES
- vertical axis options:
 - AUTOVREF
 - CAUTOVREF=*reference-line-color*
 - CVREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*
 - LAUTOVREF=*reference-line-type*
 - LVREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*
 - VAXIS=*value-list* | AXIS<1...99>
 - VMINOR=*number-of-minor ticks*
 - VREF=*value-list*
 - VREVERSE
 - VZERO

Required Arguments

plot-request(s)

each specifies the variables to plot and produces a separate graph. All variables must be in the input data set. Multiple plot requests are separated with blanks. A plot request must have this form:

*y-variable***x-variable*=*bubble-size*

plots the values of two variables and draws a circle (bubble) at each data point. The value of the third variable determines the size of the bubble. All variables must be in the input data set.

y-variable

variable plotted on the *right* vertical axis; typically it is different from *y-variable* in the accompanying BUBBLE or PLOT statement.

x-variable

variable plotted on the horizontal axis; it is the same as *x-variable* in the accompanying BUBBLE or PLOT statement.

bubble-size

variable that dictates the size of the bubbles. *Bubble-size* must be numeric. If the value of *bubble-size* is positive, bubbles are drawn with a solid line; if it is negative, bubbles are drawn with a dashed line.

Options

Options for the BUBBLE2 statement are identical to those for the BUBBLE statement except for these options, which are ignored if specified:

AUTOHREF
 CAUTOHREF=
 CHREF=
 DESCRIPTION=
 HAXIS=
 HMINOR=
 HREF=
 HZERO=
 IFRAME=
 IMAGESTYLE =
 LAUTOHREF=
 LHREF=
 NAME=

See “BUBBLE Statement” on page 1090 for complete descriptions of options used with the BUBBLE2 statement.

Coordinating BUBBLE and BUBBLE2 Plot Requests

The BUBBLE2 statement draws circles only for values that are located within the axes. Bubbles are not drawn for values that lie outside of the axis range. If a bubble size value causes a bubble to overlap the axis, the bubble is clipped against the axis line.

In the BUBBLE2 statement, either *y-variable* or *bubble-size* may differ from the variables in the BUBBLE statement. Here are some possible combinations of plot requests for BUBBLE and BUBBLE2 statement pairs and how they affect the plot:

- The vertical axis variables Y and Y2 are different, but the bubble size variable, S, is the same in both:

```
bubble y*x=s;
bubble2 y2*x=s;
```

These plot requests generate a plot in which both sets of bubbles have the same value (size) but different locations on the graph.

- The vertical axis variables are the same, Y, but the bubble size variables, S and S2, are different:

```
bubble y*x=s;
bubble2 y*x=s2;
```

The resulting plot has two identical vertical axes and two sets of concentric bubbles of different sizes.

- Both the vertical axis variables, Y and Y2, and the bubble size variables, S and S2, are different:

```
bubble y*x=s;
bubble2 y2*x=s2;
```

These plot requests produce the equivalent of an overlay plot in which two different sets of bubbles plotted against different vertical axes are displayed on the same graph.

The plot requests on the BUBBLE and BUBBLE2 statements must be evenly matched, for example:

```
bubble y*x=s b*a=c;
      bubble2 y2*x=s b2*a=c2;
```

These statements produce two graphs each with two vertical axes. The first pair of plot requests ($Y*X=S$ and $Y2*X=S$) produce one graph in which the variable X is plotted on the horizontal axis, the variable Y is plotted on the left axis, and the variable Y2 is plotted on the right axis. In this pair, the value of S is the same for both requests. The second pair of plot requests ($B*A=C$ and $B2*A=C2$) produce another graph in which the variable A is plotted on the horizontal axis, the variable B is plotted on the left axis, and the variable B2 is plotted on the right axis.

Any modifications to horizontal axes specifications must be identical for both statements; if they are different, the BUBBLE2 axis specification is ignored.

If the scale of values for the left and right vertical axes is the same and you want both axes to represent the same range of values, specify the range with a VAXIS= option in both the BUBBLE and BUBBLE2 statements.

PLOT Statement

Creates plots in which one variable is plotted on the horizontal axis and a second variable is plotted on the left vertical axis.

Requirements: At least one plot request is required.

Global statements: AXIS“AXIS Statement” on page 124, FOOTNOTE“TITLE, FOOTNOTE, and NOTE Statements” on page 210, LEGEND“LEGEND Statement” on page 151, PATTERN“PATTERN Statement” on page 169, SYMBOL“SYMBOL Statement” on page 183, TITLE“TITLE, FOOTNOTE, and NOTE Statements” on page 210

Supports: Drill-down functionality

Description

The PLOT statement specifies one or more plot requests that name the horizontal and left vertical axis variables, and optionally a third classification variable. This statement automatically

- scales the axes to include the maximum and minimum data values
- plots data points within the axes
- labels each axis with the name of its variable and displays each major tick mark value.

You can use statement options to manipulate the axes, modify the appearance of your graph, and describe catalog entries. You can use SYMBOL definitions to modify plot symbols for the data points, join data points, draw regression lines, plot confidence limits, or specify other types of interpolations. For more information on the SYMBOL statement, see “About SYMBOL Definitions” on page 1114.

In addition, you can use global statements to modify the axes; add titles, footnotes, and notes to the plot; or modify the legend if one is generated by the plot. You can also use an Annotate data set to enhance the plot.

Syntax

```
PLOT plot-request(s) </option(s)>;
```

option(s) can be one or more options from any or all of the following categories:

- plot options:
 - AREAS=*n*
 - GRID
 - LEGEND | LEGEND=LEGEND<1...99>
 - NOLEGEND
 - OVERLAY
 - REGEQN
 - SKIPMISS
- appearance options:
 - ANNOTATE=*Annotate-data-set*
 - CAXIS=*axis-color*
 - CFRAME=*background-color*
 - COUTLINE=*outline-color*
 - CTEXT=*text-color*
 - FRAME | NOFRAME
 - HREVERSE
 - IFRAME= *fileref* | '*external-file*'
 - IMAGESTYLE = TILE | FIT
 - NOAXIS | NOAXES
- horizontal axis options:
 - AUTOHREF
 - CAUTOHREF=*reference-line-color*
 - CHREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*
 - HAXIS=*value-list* | AXIS<1...99>
 - HMINOR=*number-of-minor-ticks*
 - HREF=*value-list*
 - HZERO
 - LAUTOHREF=*reference-line-type*
 - LHREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*
- vertical axis options:
 - AUTOVREF
 - CAUTOVREF=*reference-line-color*
 - CVREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*
 - LAUTOVREF=*reference-line-type*
 - LVREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*
 - VAXIS=*value-list* | AXIS<1...99>
 - VMINOR=*number-of-minor-ticks*
 - VREF=*value-list*
 - VREVERSE
 - VZERO
- catalog entry description options:
 - DESCRIPTION='entry-description'
 - NAME='entry-name'
- ODS options:

HTML=*variable*

HTML_LEGEND=*variable*

Required Arguments

plot-request(s)

each specifies the variables to plot and produces a separate graph, unless you specify OVERLAY. All variables must be in the input data set. Multiple plot requests are separated with blanks. You can plot character or numeric variables. A plot request can be any of these:

*y-variable***x-variable*<=*n*>

plots the values of two variables and, optionally, assigns a SYMBOL definition to the plot.

y-variable

variable plotted on the left vertical axis.

x-variable

variable plotted on the horizontal axis.

n

number of the *n*th generated SYMBOL definition.

Note: The *n*th generated SYMBOL definition is not necessarily the same as the *n*th SYMBOL statement. Plot requests of the form *y-variable***x-variable*=*n* assign the SYMBOL definition that is designated by *n* to the plot that is produced by *y-variable***x-variable*. For more information, see “About Plot Requests that Assign a SYMBOL Definition” on page 1114. Δ

*(y-variable(s))*x-variable(s)*

plots the values of two or more variables and produces a separate graph for each combination of Y and X variables. That is, each Y*X pair is plotted on a separate set of axes, unless you specify OVERLAY.

y-variable(s)

variables plotted on the left vertical axes.

x-variable(s)

variables plotted on the horizontal axes.

If you use only one *y-variable* or only one *x-variable*, omit the parentheses for that variable, for example,

```
plot (temp rain)*month;
```

This plot request produces two plots, one of TEMP and MONTH and one of RAIN and MONTH.

*y-variable***x-variable*=*third-variable*

plots the values of two variables against a third classification variable

y-variable

variable plotted on the left vertical axis.

x-variable

variable plotted on the horizontal axis.

third-variable

classification variable against which *y-variable* and *x-variable* are plotted. *Third-variable* can be character or numeric, but numeric variables should contain discrete rather than continuous values, or should be formatted to provide discrete values.

A separate plot (set of data points) is produced for each unique value of *third-variable*; all plots are drawn on the same set of axes, and a legend is automatically generated to show the plot symbol and color for each value of the classification variable.

Note: If a BY statement is used to produce multiple plots, you can make the legend the same across graphs by specifying the UNIFORM option in the PROC GPLOT statement. \triangle

The following plot request produces a graph with a plot line for each department and a legend that shows the plot symbol for each department:

```
plot sales*weekday=dept;
```

For an example of a plot that specifies a *third-variable*, see Example 8 on page 1135.

You can use more than one type of plot request in a single PLOT statement (provided that you do not specify OVERLAY), for example

```
plot temp*month rain*month=2;
```

Options

Options in a PLOT statement affect all graphs that are produced by that statement. You can specify as many options as you want and list them in any order.

ANNOTATE=Annotate-data-set

ANNO=Annotate-data-set

specifies a data set to annotate plots that are produced by the PLOT statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587.

AREAS=*n*

fills all the areas below plot line *n* with a pattern. The value of *n* specifies which areas to fill:

- AREAS=1 fills the first area.
- AREAS=2 fills both the first and second areas, and so forth.

If you specify a value for AREAS= that is greater than the number of bounded areas in the plot, the area between the top plot line and the axis frame is filled.

Before an area can be filled, the data points that border the area must be joined by a line. Use a SYMBOL statement with one of these interpolation methods to join the data points:

```
INTERPOL=JOIN
```

```
INTERPOL=STEP
```

```
INTERPOL=Rseries
```

```
INTERPOL=SPLINE | SM | L
```

See “SYMBOL Statement” on page 183 for details on interpolation methods.

By default, the AREAS= option fills areas by rotating a solid pattern through the colors list, starting with the first color in the list. If it needs more patterns, it rotates hatch patterns, beginning with the M2N0 pattern. See “PATTERN Statement” on page 169 for more information on map/plot patterns. However, if the V6COMP graphics option is in effect, or if color is limited to a single color with the

CPATTERN= or COLORS= graphic options, the solid pattern is skipped and the first default pattern is M2N0. If the COLORS= graphic option specifies a single color, use as many SYMBOL statements as you have areas to fill in the plot because the INTERPOL= setting does not automatically apply to multiple symbol definitions.

Note: If your device's default colors list is in effect and the first color in the list is black, color rotation begins with the second color in the list (no solid black patterns), unless the V6COMP graphics option is in effect. See "How Default Patterns and Outlines Are Generated" on page 178 for more information. Δ

You can alter the default pattern behavior by specifying patterns and colors on PATTERN statements that specify map and plot patterns. A separate PATTERN definition is needed for each specified area.

If you specify PATTERN statements, AREAS= uses the lowest numbered PATTERN statement first. If it runs out of patterns, it uses the default behavior for map and plot patterns. See "PATTERN Statement" on page 169 for details.

Pattern definitions are assigned to the areas below the plot lines in the order the plots are drawn. The first area is that between the horizontal axis and the plot line that is drawn first. The second area is that above the first plot line and below the plot line that is drawn second, and so forth. If the line that is drawn second lies below the line that is drawn first, the second area is hidden when the first is filled. The plots with the lower line values must be drawn first to prevent one area fill from overlaying another. If the lines cross, only the part of an area that is above the previous line is visible.

Therefore, if you produce multiple plots by submitting multiple plot requests and using the OVERLAY option, the plot requests must be ordered in the PLOT statement so that the plot request that produces the lowest line values is the first (leftmost) plot request, the plot request that produces the next lowest line values is the second plot request, and so on.

If you produce multiple plots with a *y-variable*x-variable=third-variable* plot request, the lines are plotted in order of increasing third variable values. Therefore, the data must be recoded so that the lowest value of the third variable produces the lowest plot line, the next lowest value produces the next lowest plot line, and so on.

AREAS= works only if all plot lines are generated by the same PLOT or PLOT2 statement.

If you use the VALUE= option in the SYMBOL statement, some symbols may be hidden. If reference lines are also specified with AREAS=, they are drawn behind the pattern fill.

Featured in: Example 7 on page 1134.

AUTOHREF

draws reference lines at all major tick marks on the horizontal axis. If the AREAS= option is also used, the filled areas cover the reference lines. To draw lines on top of the filled areas, use the ANNOTATE= option in either the PROC GPLOT statement or the PLOT statement. To specify line types for these reference lines, use the LAUTOHREF= option. To specify colors for these reference lines, use the CAUTOHREF= option. To specify labels for these reference lines, use the HAXIS= option.

AUTOVREF

draws reference lines at all of the major tick marks on the vertical axis. If you also use the AREAS= option, the filled areas cover the reference lines. To draw lines on top of the filled areas, use the ANNOTATE= option in either the PROC GPLOT statement or the PLOT statement. To specify line types for these reference lines, use the LAUTOVREF= option. To specify colors for these reference lines, use the CAUTOVREF= option. To specify labels for these reference lines, use the VAXIS= option.

CAUTOHREF=reference-line-color

specifies colors for reference lines drawn at major tick marks on the horizontal axis, as specified by the AUTOHREF option. The default color is either the value of the CAXIS= option or the first color in the color list. To specify line types for these reference lines, use the LAUTOHREF= option. To specify labels for these reference lines, use the HAXIS= option.

CAUTOVREF=reference-line-color

specifies colors for reference lines drawn at major tick marks on the vertical axis, as specified by the AUTOVREF option. The default color is either the value of the CAXIS= option or the first color in the color list. To specify line types for these reference lines, use the LAUTOVREF= option. To specify labels for these reference lines, use the VAXIS= option.

CAXIS=axis-color**CA=axis-color**

specifies the color for the axis line and all major and minor tick marks. By default, the procedure uses the first color in the colors list.

If you use the CAXIS= option, it may be overridden by

- the COLOR= option in an AXIS definition, which in turn is overridden by
- the COLOR= suboption of the MAJOR= or MINOR= option in an AXIS definition for major and minor tick marks.

Featured in: Example 5 on page 1129.

CFRAME=background-color**CFR=background-color**

fills the axis area with the specified color. If the FRAME option is also in effect, the procedure determines the color of the frame according to the precedence list given later in the FRAME option description. If the IFRAME= option is in effect, an image will appear in the background instead of the color.

CHREF=reference-line-color | (reference-line-color) | reference-line-color-list**CH=reference-line-color | (reference-line-color) | reference-line-color-list**

specifies the color of reference lines drawn perpendicular to the horizontal axis. This option affects reference lines drawn with the AUTOHREF, HREF, and GRID options. Specifying a single color without parentheses applies that color to all reference lines. The CAUTOHREF= option overrides the CHREF= option for reference lines drawn with the AUTOHREF option. Specifying a single color in parentheses applies that color only to the first reference line drawn with the HREF= option. Specifying a color list applies colors sequentially to successive reference lines drawn with the HREF= option. The syntax of the color list is of the form (*color1 color2 ...colorN*) or (*color1, color2, ...colorN*). The default color for reference lines is determined by the CAXIS= option or by the first color in the color list. To specify line types for these reference lines, use the LHREF= option. To specify labels for these reference lines, use the HAXIS= option.

COUTLINE=outline-color

specifies the color of the outline that is drawn around filled areas. The filled areas are generated when the SYMBOL statement or GOPTIONS statement specifies INTERPOL on page 191=*map/plot-pattern*. The default outline color is black for ActiveX devices. Otherwise, the default color is the first color in the colors list. The COUTLINE= option cannot be used with the PATTERN statement. The COUTLINE= option overrides the SYMBOL statement option CO=.

Not supported by: Java

CTEXT=text-color**C=text-color**

specifies the color for all text on the axes, including tick mark values and axis labels. If the PLOT request generates a legend, the CTEXT= option also colors the legend label and the value descriptions.

If you omit the CTEXT= option, a color specification is searched for in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the default, the first color in the colors list.

If you use the CTEXT= option, it overrides the color specification for the axis label and the tick mark values in the COLOR= option in an AXIS definition that is assigned to the axis.

If you use the CTEXT= option, the color specification is overridden in one or more of these situations:

- If you also use the COLOR= suboption of a LABEL= or VALUE= option in a AXIS definition that is assigned to the axis, that suboption determines the color of the axis label or the color of the tick mark values, respectively.
- If you also use the COLOR= suboption of a LABEL= or VALUE= option in a LEGEND definition that is assigned to the legend, it determines the color of the legend label or the color of the legend value descriptions, respectively.

Featured in: Example 5 on page 1129

CVREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*

CV=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*

specifies the color of reference lines drawn perpendicular to the vertical axis.

Specifying a single color without parentheses applies that color to reference lines drawn with the AUTOVREF and VREF= options. The CAUTOVREF= option overrides the CVREF= option for reference lines drawn with the AUTOVREF option. Specifying a single color in parentheses applies that color only to the first reference line drawn with the VREF= option. Specifying a color list applies colors sequentially to successive reference lines drawn with the VREF= option. The syntax of the color list is of the form (*color1 color2 ...colorN*) or (*color1, color2, ...colorN*). The default color for reference lines is determined by the CAXIS= option or by the first color in the color list. To specify line types for these reference lines, use the LVREF= option. To specify labels for these reference lines, use the VAXIS= option.

For needle plots that are generated with a Java or ActiveX device driver, the value of the CVREF= option is not applied to the default reference line that is drawn at zero when the minimum value of the vertical axis is less than zero. The color of this line is the first color in the color list, which is black by default.

Featured in: Example 5 on page 1129

Not supported by: Java (partial), ActiveX (partial)

DESCRIPTION=*'entry-description'*

DES=*'entry-description'*

specifies the description of the catalog entry for the plot. The maximum length for *entry-description* is 256 characters. The description does not appear on the plot. By default, the procedure assigns a description of the form PLOT OF *y-variable***x-variable*, where *y-variable* and *x-variable* are the names of the plot variables.

The *entry-description* can include the #BYLINE, #BYVAL, and #BYVAR substitution options, which work as they do when used on TITLE, FOOTNOTE, and NOTE statements. For more information refer to the description of the text-string on page 222 option and the section that discusses “Substituting BY Line Values in a Text String” on page 226. The 256-character limit applies before the substitution takes place for these options; thus, if in the SAS program the entry-description text exceeds 256 characters, it is truncated to 256 characters, and then the substitution is performed.

The descriptive text is shown in the "description" portion of each of the following:

- in the Results window
- among the catalog-entry properties that you can view from the Explorer window
- in the Table of Contents that is generated when you use the CONTENTS= option on an ODS statement. This assumes that the GPLOT output is generated while the contents page is open. See “Linking to Output through a Table of Contents” on page 495.
- in the Description field of the PROC GREPLAY window

FRAME | NOFRAME

FR | NOFR

specifies whether a frame is drawn around the axis area. The default is FRAME; however, if the V6COMP option is in effect on the GOPTIONS statement, the default is NOFRAME. If you also use a BUBBLE2 or PLOT2 statement and your plotting statements have conflicting frame specifications, FRAME is used.

For the frame color, a specification is searched for in this order:

- 1 the CAXIS= option
- 2 the COLOR= option in the AXIS definition assigned to the vertical axis
- 3 the COLOR= option in the AXIS definition assigned to the horizontal axis
- 4 the default, the first color in the colors list.

To fill the axis area with a background color, use the CFRAME= option.

To fill the axis area with a background image, use the IFRAME= option.

GRID

draws reference lines at all major tick marks on both axes. You get the same result when you use all of these options in a PLOT statement: AUTOHREF, AUTOVREF, FRAME, LVREF=34, and LHREF=34. The line type for GRID is 34. The line color is the color of the axis. When specified in a PLOT2 statement, the reference lines are drawn on the vertical axis on the right side of the plot.

HAXIS=*value-list* | AXIS<1 . . . 99>

specifies major tick mark values for the horizontal axis or assigns an axis definition. By default, the procedure scales the axis and provides an appropriate number of tick marks. To assign labels to reference lines, use an axis definition that contains the REFLABEL= option. The labels will be applied in sequence to all reference lines defined with the AUTOHREF and HREF= options.

The way you specify *value-list* depends on the type of variable:

- For numeric variables, *value-list* is either an explicit list of values, or a starting and an ending value with an interval increment, or a combination of both forms:

n <...*n*>

n TO *n* <BY *increment*>

n <...*n*> TO *n* <BY *increment* > <*n* <...*n*> >

If a numeric variable has an associated format, the specified values must be the *unformatted* values.

- For date-time values, *value-list* includes any SAS date, time, or datetime value described for the SAS functions INTCK and INTNX, shown here as *SAS-value*:

'*SAS-value*'*i* < ...'*SAS-value*'*i*>

'*SAS-value*'*i* TO '*SAS-value*' *i*<BY *interval*>

- For character variables, *value-list* is a list of unique character values enclosed in quotation marks and separated by blanks:

'*value-1*' < ...'*value-n*'>

If a character variable has an associated format, the specified values must be the *formatted* values.

For a complete description of *value-list*, see the ORDER= on page 130 option in the AXIS statement.

If data values fall outside of the range that is specified by the HAXIS= option, then by default the outlying data values are not used in interpolation calculations. See “About the Input Data Set” on page 1086 for more information on values out of range.

For Web output that is generated with a Java or ActiveX device driver, certain options of the AXIS statement are not supported. For details, see “AXIS Statement” on page 124.

Featured in: Example 4 on page 1126, Example 5 on page 1129, and Example 9 on page 1138.

Not supported by: Java (partial), ActiveX (partial)

HMINOR=*number-of-minor-ticks*

HM=*number-of-minor-ticks*

specifies the number of minor tick marks drawn between each major tick mark on the horizontal axis. Minor tick marks are not labeled. The HMINOR= option overrides the NUMBER= suboption of the MINOR= option in an AXIS definition. You must specify a positive number.

Featured in: Example 4 on page 1126, Example 5 on page 1129, and Example 9 on page 1138.

HREF=*value-list*

draws one or more reference lines perpendicular to the horizontal axis at points specified by *value-list*. See the HAXIS= option for a description of *value-list*. If the AREAS= option is also used, the filled areas cover the reference lines. To draw lines on top of the filled areas, use the ANNOTATE= option on either the PROC GPLOT or the PLOT statement. To specify colors for these reference lines, use the CHREF= option. To specify line types for these reference lines, use the LHREF= option. To specify labels for these reference lines, use the HAXIS= option.

HREVERSE

specifies that the order of the values on the horizontal axis be reversed. For Web output that is generated with a Java device driver, the horizontal axis data must be numeric.

Not supported by: Java (partial)

HTML=*variable*

identifies the variable in the input data set whose values create links in the HTML output file that is generated by ODS. These links are associated with the plot points, or if AREA= is used, with the areas between plot lines. The links point to the data or graph that you wish to display when the user drills down on the plot point or area. The maximum length for the value of this variable is 1024 characters.

Note that the HTML= option is functional only when a single PLOT or PLOT2 statement appears in the PROC GPLOT procedure.

Not supported by: Java (partial), ActiveX (partial)

HTML_LEGEND=*variable*

identifies the variable in the input data set whose values are used to create links in the HTML output file that is generated by ODS. When the HTML output file is displayed in a Web browser, clicking on an element in the legend displays the URL that was specified for that legend element, based on the value of the variable that is named as the value of the HTML_LEGEND option. The maximum length for the

value of this variable is 1024 characters. To see an example that generates a drill-down graph using ODS, see Example 10 on page 1141.

Not supported by: Java, ActiveX

HZERO

specifies that tick marks on the horizontal axis begin in the first position with a value of zero. The HZERO request is ignored if negative values are present for the horizontal variable or if the horizontal axis has been specified with the HAXIS= option.

IFRAME=*fileref* | '*external-file*'

identifies the image file you wish to apply to the backplane frame of the plot. See also the IMAGESTYLE= option and “Placing a Backplane Image on Graphs with Frames” on page 115. The IFRAME= option is overridden by the NOIMAGEPRINT option “IMAGEPRINT” on page 318.

For Web output that is generated with the ACTIVEX or ACTXIMG device drivers,

Not supported by: Java

IMAGESTYLE= TILE | FIT

specifies whether to tile multiple instances of the image to fill the backplane frame (TILE) or to stretch a single instance of the image to fill the backplane frame (FIT). The TILE value is the default. See also the IFRAME= option.

Not supported by: Java

LAUTOHREF=*reference-line-type*

specifies a line type for reference lines drawn at major tick marks on the horizontal axis, as specified by the AUTOHREF option. The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. The default value 1 draws a solid line. To specify colors for these reference lines, use the CAUTOHREF= option. To specify labels for these reference lines, use the HAXIS= option.

LAUTOVREF=*reference-line-type*

specifies a line type for reference lines drawn at major tick marks on the vertical axis, as specified by the AUTOVREF option. The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. The default value 1 draws a solid line. To specify colors for these reference lines, use the CAUTOVREF= option. To specify labels for these reference lines, use the VAXIS= option.

LEGEND | LEGEND=LEGEND<1...99>

generates a legend or specifies the legend to use for the plot.

- a PLOT statement that includes the OVERLAY option does not automatically generate a legend. In these plot types, use LEGEND to produce a default legend, or LEGEND=LEGEND n to assign a defined LEGEND statement to the plot. The default legend is centered below the axis frame and identifies which colors and plot symbols represent the *y-variables* that you specify for the plots.
- a plot request of the form *y-variable***x-variable*=*third-variable* automatically generates a default legend that identifies which colors and plot symbols represent each value of the classification variable. In these plot types, override the default by using LEGEND=LEGEND n to assign a defined LEGEND statement to the plot.

If you use the SHAPE= option in a LEGEND statement, the value SYMBOL is valid. If you use the PLOT statement’s AREAS= option, SHAPE=BAR is also valid.

See also: “LEGEND Statement” on page 151.

Featured in: Example 6 on page 1131.

LHREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*

LH=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*

specifies line types for reference lines drawn perpendicular to the horizontal axis. The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. This option affects reference lines drawn with the AUTOHREF, HREF, and GRID options. Specifying a single line type without parentheses applies that line type to all reference lines. The LAUTOHREF= option overrides the LHREF= option for lines drawn with the AUTOHREF option. Specifying a single line type in parentheses applies that line type only to the first reference lines drawn with the HREF= option. Specifying a line-type list applies line types in sequence to successive reference lines drawn with the HREF= option. The syntax of the line type list is of the form (*type1 type2... typeN*). The default value 1 draws a solid line. To specify colors for these reference lines, use the CHREF= option. To specify labels for these reference lines, use the HAXIS= option.

LVREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*

LV=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*

specifies line types for reference lines drawn perpendicular to the vertical axis. The *reference-line-type* value can be a whole number from 1 to 46. A value of 1 specifies a solid line; values 2 through 46 specify dashed lines. This option affects reference lines drawn with the AUTOVREF, VREF, and GRID options. Specifying a single line type without parentheses applies that line type to all reference lines. The LAUTOVREF= option overrides the LVREF= option for lines drawn with the AUTOVREF option. Specifying a single line type in parentheses applies that line type only to the first line drawn with the VREF= option. Specifying a line-type list applies line types in sequence to successive reference lines drawn with the VREF= option. The syntax of the line type list is of the form (*type1 type2... typeN*). The default value 1 draws a solid line. To specify colors for these reference lines, use the CVREF= option. To specify labels for these reference lines, use the VAXIS= option.

For needle plots that are generated with a Java or ActiveX device driver, the value of the LVREF= option is not applied to the default reference line that is drawn at zero when the minimum value of the vertical axis is less than zero. This line is solid (not dashed).

Featured in: Example 5 on page 1129.

Not supported by: Java (partial), ActiveX (partial)

NAME= '*entry-name*'

specifies the name of the catalog entry for the graph. The maximum length for *entry-name* is eight characters. The default name is GPLOT. If you specify DEVICE=ACTIXIMG or DEVICE=JAVAIMG, then the name that you specify will be used for the Java or ActiveX device driver image output even if the file exists. For all other devices, if the name duplicates an existing entry name, SAS/GRAPH adds a number to the duplicate name to create a unique entry, for example, GPLOT1.

NOAXIS

NOAXES

suppresses the axes, including axis lines, axis labels, all major and minor tick marks, and tick mark values.

NOLEGEND

suppresses the legend that is generated by a plot request of the type *y-variable*x-variable=third-variable*.

OVERLAY

places all the plots that are generated by the PLOT statement on one set of axes. The axes are scaled to include the minimum and maximum values of all of the

variables, and the variable names or labels associated with the first pair of variables label the axes.

The OVERLAY option produces a legend if you include the LEGEND or the LEGEND=*n* option in the PLOT statement.

OVERLAY is not enabled with plot requests of the form *y-variable*x-variable=third-variable*. However, you can achieve an overlay effect by using a PLOT and PLOT2 statement.

When generating output for the Web with the JAVA, JAVAMETA, or JAVAIMG device drivers, the OVERLAY option cannot be used in the PLOT statement in combination with the global statement SYMBOL. This applies only when the SYMBOL statement uses the INTERPOL= option, and when the INTERPOL= option has the values BOX, HILO, or STD. For Java output using the PLOT2 statement, INTERPOL=BOX|HILO|STD cannot be used in a SYMBOL statement, with or without the OVERLAY option.

Featured in: Example 6 on page 1131 and Example 7 on page 1134.

Not supported by: Java (partial)

REGEQN

displays the regression equation that is specified in the INTERPOL= option of the SYMBOL statement in the lower left hand corner of the plot. You cannot modify the format that is used for the equation.

The GPLOT regression equation is computed from the screen coordinates of the markers. Therefore, a graph might not display if the chart area for the plot becomes so small that markers cannot be drawn because there are no coordinates from which to build the regression equation. In such cases, the regression equation is no longer meaningful.

Featured in: Example 4 on page 1126.

Not supported by: Java, ActiveX

SKIPMISS

breaks a plot line or an area fill at occurrences of missing values of the Y variable. By default, plot lines and area fills are not broken at missing values. SKIPMISS is available only with JOIN interpolation. If SKIPMISS is used, observations should be sorted by the independent (horizontal axis) variable. If the plot request is *y-variable*x-variable=third-variable*, observations should also be sorted by the values of the third variable.

See also: “About the Input Data Set” on page 1086.

VAXIS=*value-list* | AXIS<1...99>

specifies the major tick mark values for the vertical axis or assigns an axis definition. See the HAXIS= option for a description of the *value-list*. To assign labels to reference lines, use an axis definition that contains the REFLABEL= option. The labels will be applied in sequence to all reference lines defined with the AUTOVREF and VREF= options.

For Web output that is generated with a Java or ActiveX device driver, certain options of the AXIS statement are not supported. For details, see “AXIS Statement” on page 124.

Featured in: Example 4 on page 1126 and Example 5 on page 1129.

Not supported by: Java (partial), ActiveX (partial)

VMINOR=*number-of-minor-ticks*

VM=*number-of-minor-ticks*

specifies the number of minor tick marks that are drawn between each major tick mark on the vertical axis. Minor tick marks are not labeled. The VMINOR= option overrides the NUMBER= suboption of the MINOR= option in an AXIS definition. You must specify a positive number.

Featured in: Example 5 on page 1129.

VREF=*value-list*

draws one or more reference lines perpendicular to the vertical axis at points that are specified by the *value-list*. See the HAXIS= option for a description of the *value-list*. If the AREAS= option is also used, the filled areas cover the reference lines. To draw lines on top of the filled areas, use the ANNOTATE= option in either the PROC GPLOT statement or the PLOT statement. To specify colors for these reference lines, use the CVREF= option. To specify line types for these reference lines, use the LVREF= option. To specify labels for these reference lines, use the VAXIS= option.

Featured in: Example 5 on page 1129.

VREVERSE

specifies that the order of the values on the vertical axis be reversed.

VZERO

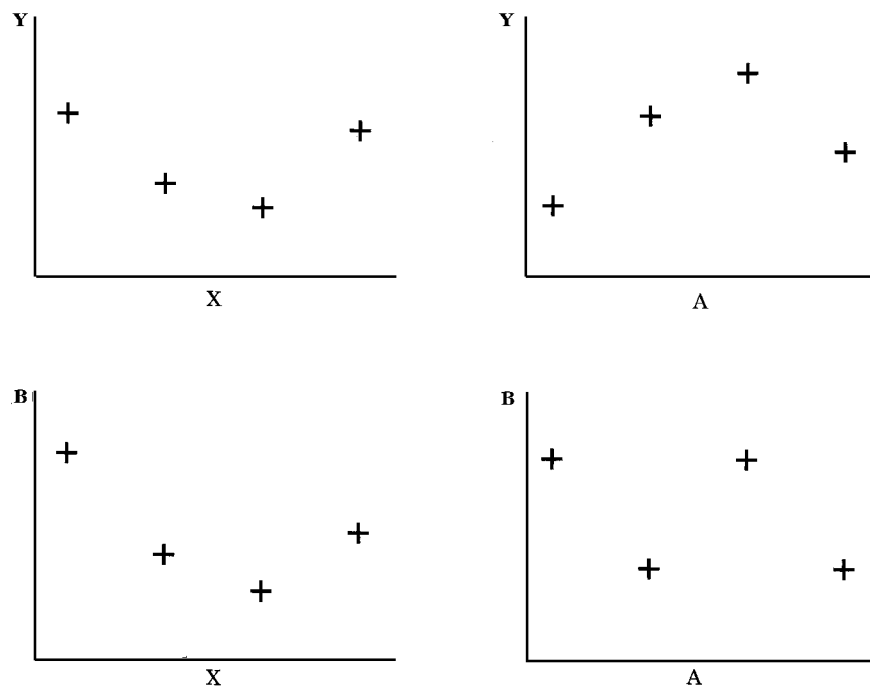
specifies that tick marks on the vertical axis begin in the first position with a zero. The VZERO request is ignored if the vertical variable either contains negative values or has been ordered with the VAXIS= option or the ORDER= option in an AXIS statement.

Plot Requests with Multiple Variables

Plot requests with multiple variables produce a separate plot for every Y*X pair, unless you specify OVERLAY. For example, this statement produces four plots (the actual plots are produced on separate pages). See Figure 37.7 on page 1113

```
plot (y b)*(x a);
```

Figure 37.7 Graphs Generated by Multiple Plot Requests



About SYMBOL Definitions

SYMBOL statements control the appearance of plot symbols and lines, and define interpolation methods. They can specify

- the shape, size, and color of the plot symbols that mark the data points
- plot line style, color, and width
- an interpolation method for plotting data
- how missing values are treated in interpolation calculations.

SYMBOL definitions are assigned either by default by the GPLOT procedure or explicitly with a plot request.

If no SYMBOL definition is currently in effect, the GPLOT procedure produces a scatter plot of the data points using the default plot symbol, the plus sign (+). If you need more than one SYMBOL definition, the procedure rotates through the current colors list to produce symbols of different colors. If the current colors list contains only one color, or if all the colors are used, additional plot symbols are used.

If SYMBOL definitions have been defined but not explicitly assigned by a plot request of the form *y-variable*x-variable=n*, the procedure assigns them in the order in which they are generated. For example, this statement creates three plots:

```
plot y*x b*a s*r;
```

The procedure assigns the first generated SYMBOL definition to Y*X, the second generated SYMBOL definition to B*A, and the third to S*R.

If more SYMBOL definitions are needed than have been defined, the procedure uses the default definitions for the plots that remain.

See “SYMBOL Statement” on page 183.

About Plot Requests that Assign a SYMBOL Definition

Plot requests of the form *y-variable*x-variable=n* are useful when you use the OVERLAY option to produce multiple plots on one graph and you want to assign a particular SYMBOL definition to each plot.

With plot requests of this type it is important to remember that a single SYMBOL statement can generate multiple SYMBOL definitions, so that the SYMBOL definition that is designated by *n* may not be the same as the SYMBOL statement of the same number. That is, the third SYMBOL definition is not necessarily the same as the SYMBOL3 statement. See “SYMBOL Statement” on page 183 for more information on the SYMBOL statement.

PLOT2 Statement

Produces one or more plots with the vertical axis on the right side of the graph against which a second variable can be plotted.

Requirements: You cannot use the PLOT2 statement alone. It can be used only with a PLOT or BUBBLE statement. At least one plot request is required.

Global statements: AXIS“AXIS Statement” on page 124, FOOTNOTE“TITLE, FOOTNOTE, and NOTE Statements” on page 210, LEGEND“LEGEND Statement” on page 151, PATTERN“PATTERN Statement” on page 169, SYMBOL“SYMBOL Statement” on page 183, TITLE“TITLE, FOOTNOTE, and NOTE Statements” on page 210

Description

The PLOT2 statement specifies one or more plot requests that name the horizontal and right vertical axis variables. This statement automatically

- plots data points within the axes
- scales the axes to include the maximum and minimum data values
- labels each axis with the name of its variable and displays each major tick mark value.

You can use statement options to manipulate the axes and modify the appearance of your graph. You can use SYMBOL definitions to modify plot symbols for the data points, join data points, draw regression lines, plot confidence limits, or specify other types of interpolation. For more information on the SYMBOL statement see “About SYMBOL Definitions” on page 1114.

Note: When using PLOT2 to generate output with the JAVA or ACTIVEX device drivers, and when the global statement SYMBOL is used, the value of the SYMBOL statement option INTERPOL= cannot be BOX, STD, or HILO. △

In addition, you can use global statements to modify the axes; add titles, footnotes, and notes to the plot; or modify the legend if one is generated by the plot. You can also use an Annotate data set to enhance the plot.

Syntax

PLOT2 *plot-request(s)* </option(s)>;

option(s) can be one or more options from any or all of the following categories:

- plot options:
 - AREAS=*n*
 - GRID
 - LEGEND | LEGEND=LEGEND<1...99>
 - NOLEGEND
 - OVERLAY
 - REGEQN
 - SKIPMISS
- appearance options:
 - ANNOTATE=*Annotate-data-set*

CAXIS=*axis-color*
 CFRAME=*background-color*
 COUTLINE=*outline-color*
 CTEXT=*text-color*
 FRAME | NOFRAME
 NOAXIS | NOAXES

- vertical axis options:

AUTOVREF
 CAUTOVREF=*reference-line-color*
 CVREF=*reference-line-color* | (*reference-line-color*) | *reference-line-color-list*
 HREVERSE
 LAUTOVREF=*reference-line-type*
 LVREF=*reference-line-type* | (*reference-line-type*) | *reference-line-type-list*
 VAXIS=*value-list* | AXIS<1...99>
 VMINOR=*n*
 VREF=*value-list*
 VREVERSE
 VZERO

- ODS options:

HTML=*variable*
 HTML_LEGEND=*variable*

Required Arguments

plot-request(s)

each specifies the variables to plot and produces a separate graph, unless you specify OVERLAY. All variables must be in the input data set. Multiple plot requests are separated with blanks. A plot request can be any of these:

*y-variable***x-variable*<=*n*>

plots the values of two variables and, optionally, assigns a SYMBOL definition to the plot.

y-variable

variable plotted on the *right* vertical axis.

x-variable

variable plotted on the horizontal axis.

n

number of the *n*th generated SYMBOL definition.

(*y-variable(s)*)*(*x-variable(s)*)

plots the values of two or more variable and produces a separate graph for each combination of Y and X variables.

y-variable(s)

variables plotted on the *right* vertical axes.

x-variable(s)

variables plotted on the horizontal axes.

*y-variable***x-variable*=*third-variable*

plots the values of two variables against a third classification variable

y-variable

variable plotted on the *right* vertical axis.

x-variable

variable plotted on the horizontal axis.

third-variable

classification variable against which *y-variable* and *x-variable* are plotted.

Third-variable can be character or numeric, but numeric variables should contain discrete rather than continuous values, or should be formatted to provide discrete values.

For more information about plot requests, see “PLOT Statement” on page 1101.

In a PLOT2 plot request, the X variable for the horizontal axis must be the same as in the accompanying PLOT or BUBBLE statement. Typically, the Y variable for the right vertical axis is different.

Use the same types of plot requests with a PLOT2 statement that you use with a PLOT statement, but a PLOT2 statement always plots the values of *y-variable* on the right vertical axis.

Options

Options for the PLOT2 statement are identical to those for the PLOT statement except for these options, which are ignored if you specify them:

AUTOHREF

CAUTOHREF=

CHREF=

DESCRIPTION=

HAXIS=

HMINOR=

HREF=

HZERO=

IFRAME=

IMAGESTYLE =

LAUTOHREF=

LHREF=

NAME=

See “PLOT Statement” on page 1101 for descriptions of options that you can use with the PLOT2 statement.

Matching Plot Requests

The plot requests in both the PLOT and PLOT2 statements must be evenly matched as in this example:

```
plot y*x b*a;
plot2 y2*x b2*a;
```

These statements produce two graphs, each with two vertical axes. The first pair of plot requests (Y*X and Y2*X) produce one graph in which X is plotted on the horizontal axis, Y is plotted on the left axis, and Y2 is plotted on the right axis. The second pair of

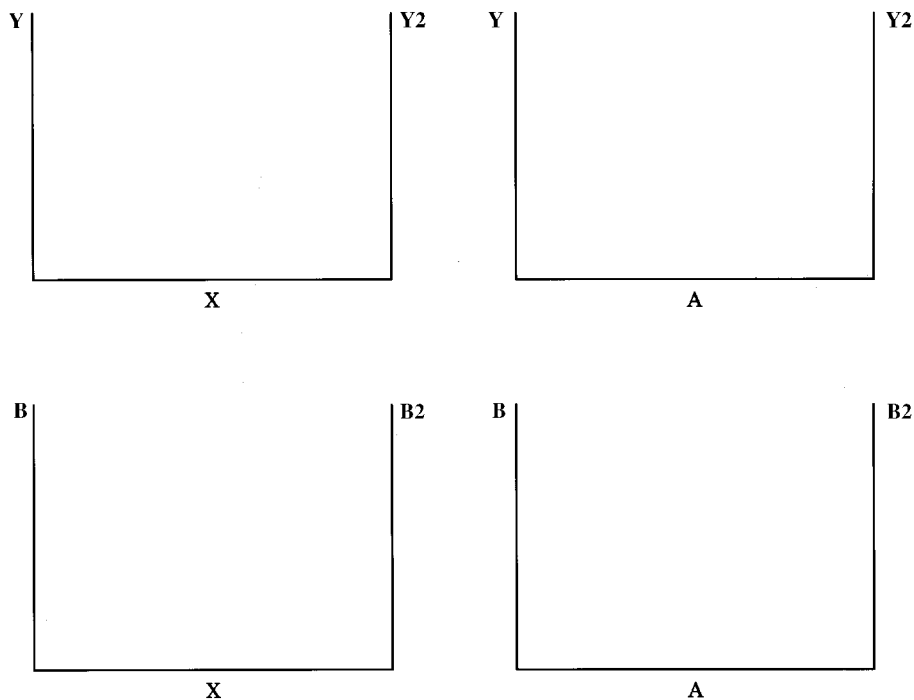
plot requests (B*A and B2*A) produce another graph in which A is plotted on the horizontal axis, B is plotted on the left axis, and B2 is plotted on the right axis.

Using Multiple Plot Requests

Plot requests of the form (y-variable(s))*(x-variable(s)). Both the PLOT and PLOT2 statements generate multiple graphs (the actual plots are produced on separate pages). See Figure 37.8 on page 1118

```
plot (y b)*(x a);
plot2 (y2 b2)*(x a);
```

Figure 37.8 Diagram of Graphs Produced by Multiple Plot Requests in PLOT and PLOT2 Statements

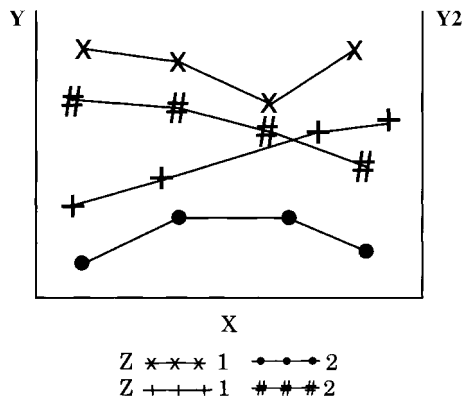


Requesting Plots of Three Variables with a Legend

When both the PLOT and PLOT2 statements use plot requests of the form $y\text{-variable} * x\text{-variable} = \text{third-variable}$, each statement generates a separate legend. If the third variable has two values, these statements produce one graph with four sets of data points. See Figure 37.9 on page 1119. The figure assumes SYMBOL statements are used to specify the plot symbols that are shown and to connect the data points with straight lines.

```
plot y*x=z;
plot2 y2*x=z;
```

Figure 37.9 Diagram of Multiple Plots on One Graph



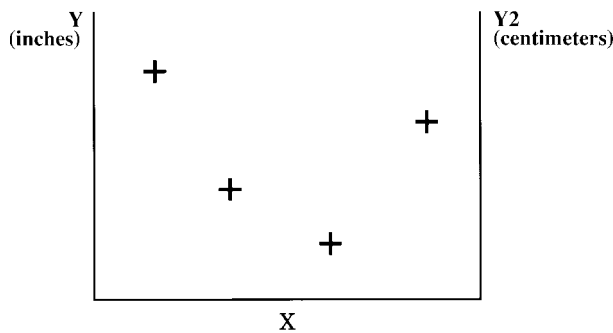
Using a Second Vertical Axis

Displaying the Same Values in a Different Scale

If your data contain the same variable values in two different scales, such as height in inches and height in centimeters, you can display one scale of values on the left axis and the other scale of values on the right axis. If both vertical axes are calibrated so that they represent the same range of values, then for each observation of X the data points for Y and Y2 are the same.

For example, if Y is height in inches and Y2 is height in centimeters and if the Y axis values range from 0 to 84 inches and the Y2 axis values range from 0 to 213.36 centimeters, the plot will be like the diagram shown in Figure 37.10 on page 1119.

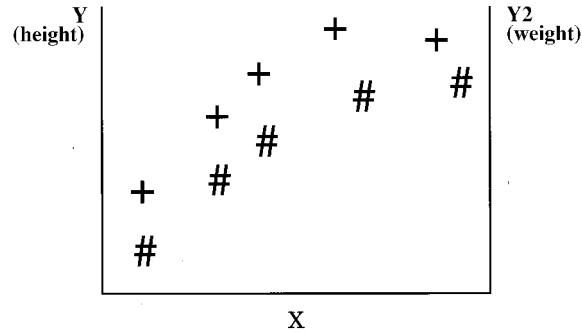
Figure 37.10 Right Axis with Different Scale of Values



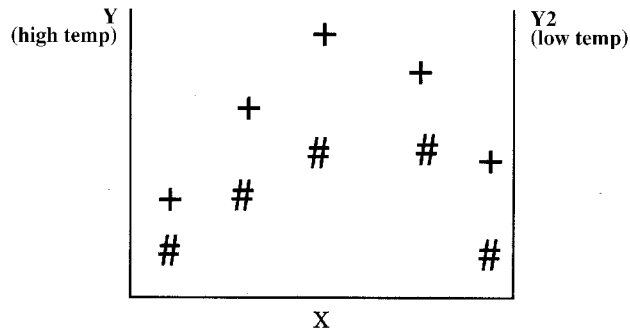
For plots such as these, the PLOT2 statement should use a SYMBOL statement that specifies INTERPOL=NONE and VALUE=NONE.

Displaying Different Values

If your data contain variables with different data values (such as height and weight), you can display one type of data on the left axis and another type of data on the right axis. Because the Y variable and the Y2 variable contain different data, two sets of data points are displayed on the graph. For example, if Y is height and Y2 is weight, the plot will be like the diagram in Figure 37.11 on page 1120.

Figure 37.11 Right Axis with Different Values and Different Scale**Displaying the Same Scale on Both Axes**

If your data contain two sets of values for the same type of data, you can use the PLOT2 statement to generate a right axis that is calibrated the same as the left axis so that the data points on the right of the graph are easier to read. For example, if Y is high temperatures and Y2 is low temperatures, you can create a graph like the diagram in Figure 37.12 on page 1120.

Figure 37.12 Right Axis with Same Scale of Values

To scale both axes the same, specify the same range of values either with the VAXIS= option in both the PLOT and PLOT2 statements, or with AXIS statements.

Using PATTERN and SYMBOL Definitions

The PLOT2 statement uses PATTERN and SYMBOL definitions in the same way the PLOT statement does. These definitions are assigned in order first to the PLOT statement and then to the PLOT2 statement.

For more information, see “About SYMBOL Definitions” on page 1114.

Examples

Example 1: Generating a Simple Bubble Plot

Procedure features:

BUBBLE statement option:

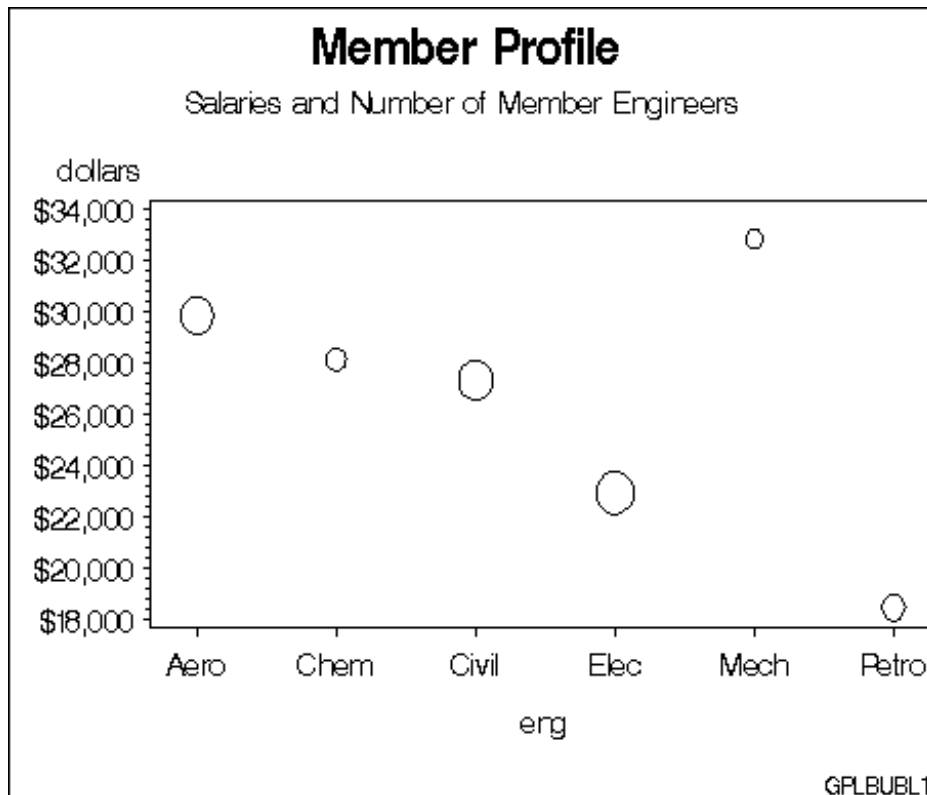
HAXIS=

Other features:

AXIS statement

FORMAT statement

Sample library member: GPLBUBL1



This example shows a bubble plot in which each bubble represents a category of engineer. The plot shows engineers on the horizontal axis and average salaries on the vertical axis. Each bubble's vertical location is determined by the average salary for the category. Each bubble's size is determined by the number of engineers in the category: the more engineers, the larger the bubble.

Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftitle=swissb ftext=swiss htitle=6 htext=4;
```

Create the data set. The data set JOBS contains average salary data for several categories of engineer. It also indicates the number of engineers in each category.

```
data jobs;
        length eng $5;
```

```

        input eng dollars num;
        datalines;
Civil 27308 73273
Aero  29844 70192
Elec  22920 89382
Mech  32816 19601
Chem  28116 25541
Petro 18444 34833
;

```

Define titles and footnote.

```

title1 'Member Profile';
title2 'Salaries and Number of Member Engineers';
footnote h=3 j=r 'GPLBUBL1 ';

```

Define axis characteristics. OFFSET= specifies an offset for the tick marks so that bubbles near an axis are not clipped.

```

axis1 offset=(5,5);

```

Generate bubble plot. HAXIS= assigns the AXIS1 statement to the horizontal axis. The salary averages are assigned a dollar format.

```

proc gplot data=jobs;
    format dollars dollar9.;
    bubble dollars*eng=num / haxis=axis1;
run;
quit;

```

Example 2: Labeling and Sizing Plot Bubbles

Procedure features:

BUBBLE statement options:

```

BCOLOR=
BFONT=
BLABEL
BSIZE=
CAXIS=
HAXIS=
VAXIS=
VMINOR

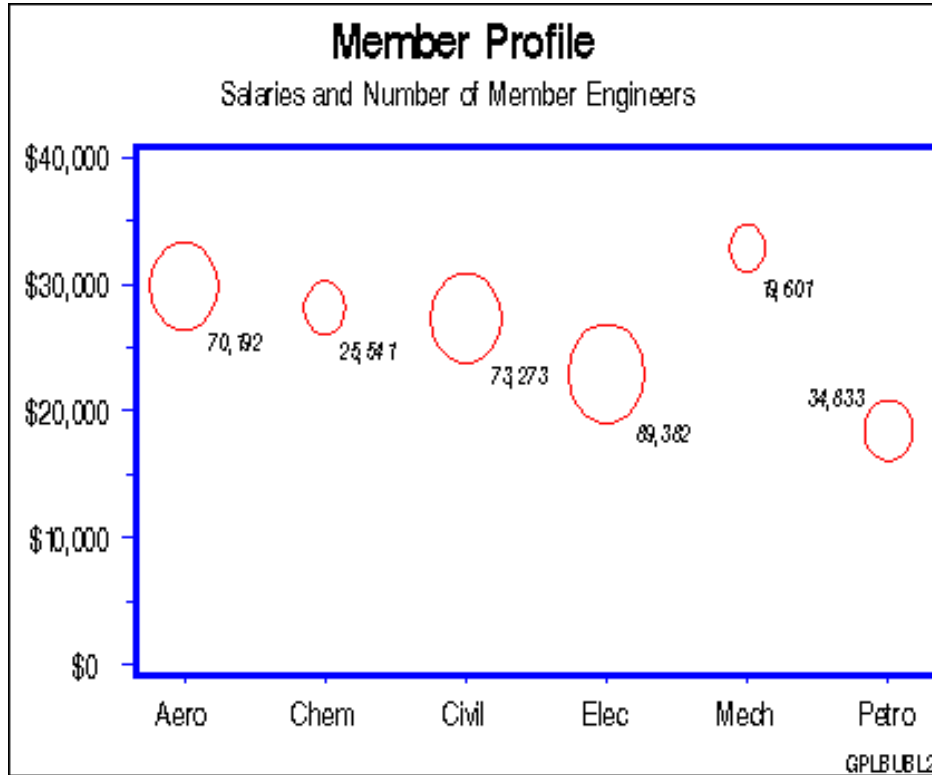
```

Other features:

AXIS statement

Data set: JOBS on page 1121

Sample library member: GPLBUBL2



This example modifies the code in Example 1. It shows how BUBBLE statement options control the appearance of bubbles and their labels. It also shows how AXIS statements can modify the plot axes.

Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftitle=swissb ftext=swiss htitle=6 htext=4;
```

Define titles and footnote.

```
title1 'Member Profile';
title2 h=4 'Salaries and Number of Member Engineers';
footnote1 h=3 j=r 'GPLBUBL2 ';
```

Define axis characteristics. AXIS1 suppresses the horizontal axis label and uses OFFSET= to move the first and last major tick mark values away from the vertical axes so bubbles are not clipped. AXIS2 uses ORDER= to set major tick mark intervals. This could be done with VAXIS= on the BUBBLE statement, but then you could not suppress the axis label and alter other axis characteristics.

```
axis1 label=none
      offset=(5,5)
      width=3
```

```

        value=(height=4);
axis2 order=(0 to 40000 by 10000)
      label=none
      major=(height=1.5)
      minor=(height=1)
      width=3
      value=(height=4);

```

Generate bubble plot. VMINOR= specifies one minor tick mark for the vertical axis. BCOLOR= colors the bubbles. BLABEL labels each bubble with the value of variable NUM, and BFONT= specifies the font for labeling text. BSIZE= increases the bubble sizes by increasing the scaling factor size to 12. CAXIS= colors the axis lines and all major and minor tick marks.

```

proc gplot data=jobs;
  format dollars dollar9. num comma7.0;
  bubble dollars*eng=num / haxis=axis1
                        vaxis=axis2
                        vminor=1
                        bcolor=red
                        blabel
                        bfont=swissi
                        bsize=12
                        caxis=blue;

run;
quit;

```

Example 3: Adding a Right Vertical Axis

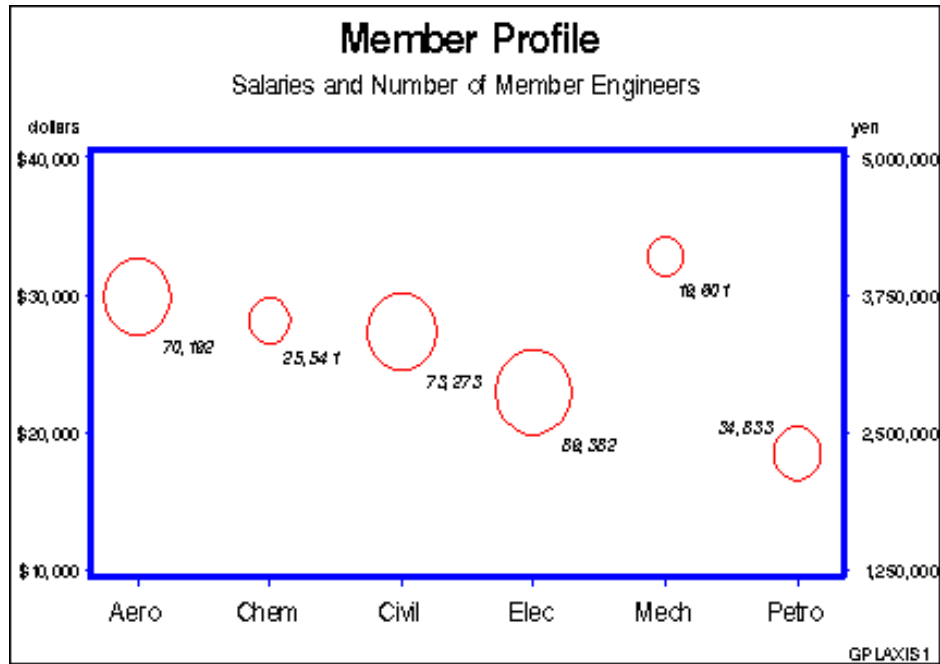
Procedure features:

BUBBLE2 statement options:

BCOLOR=
 BSIZE=
 CAXIS=
 VAXIS=

Data set: JOBS on page 1121

Sample library member: GPLAXIS1



This example modifies Example 2 on page 1122 to show how a BUBBLE2 statement generates a right vertical axis that displays the values of the vertical coordinates in a different scale from the scale that is used for the left vertical axis. Salary values are scaled by dollars on the left vertical axis and by yen on the right vertical axis.

BUBBLE and BUBBLE2 statement options control the size and appearance of the bubbles and their labels. In particular, the VAXIS options calibrate the axes so that the data points are identical and only one set of bubbles appears.

Note: If the data points are not identical, two sets of bubbles are displayed. Δ

Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftitle=swissb ftext=swiss htitle=6 htext=3;
```

Create the data set JOBS2 and calculate variable YEN. The DATA step uses a SET statement to read the JOBS data set.

```
data jobs2;
  set jobs;
  yen=dollars*125;
run;
```

Define titles and footnote.

```
title1 'Member Profile';
```

```

title2 h=4 'Salaries and Number of Member Engineers';
footnote j=r ' GPLAXIS1 ';

```

Define horizontal-axis characteristics.

```

axis1 offset=(5,5)
      label=none
      width=3
      value=(h=4);

```

Generate bubble plot with second vertical axis. In the BUBBLE statement, HAXIS= specifies the AXIS1 definition and VAXIS= scales the left axis. In the BUBBLE2 statement, VAXIS= scales the right axis. Both axes represent the same range of monetary values. The BUBBLE and BUBBLE2 statements ensure that the bubbles generated by each statement are identical by coordinating specifications on BCOLOR=, which colors the bubbles; BSIZE=, which increases the size of the scaling factor to 12; and CAXIS=, which colors the axis lines and all major and minor tick marks. Axis labels and major tick mark values use the default color, which is the first color in the colors list.

```

proc gplot data=jobs2;
  format dollars dollar7. num yen comma9.0;
  bubble dollars*eng=num / haxis=axis1
                    vaxis=10000 to 40000 by 10000
                    hminor=0
                    vminor=1
                    blabel
                    bfont=swissi
                    bcolor=red
                    bsize=12
                    caxis=blue;

  bubble2 yen*eng=num / vaxis=1250000 to 5000000 by 1250000
                    vminor=1
                    bcolor=red
                    bsize=12
                    caxis=blue;

run;
quit;

```

Example 4: Plotting Two Variables

Procedure features:

PLOT statement options:

```

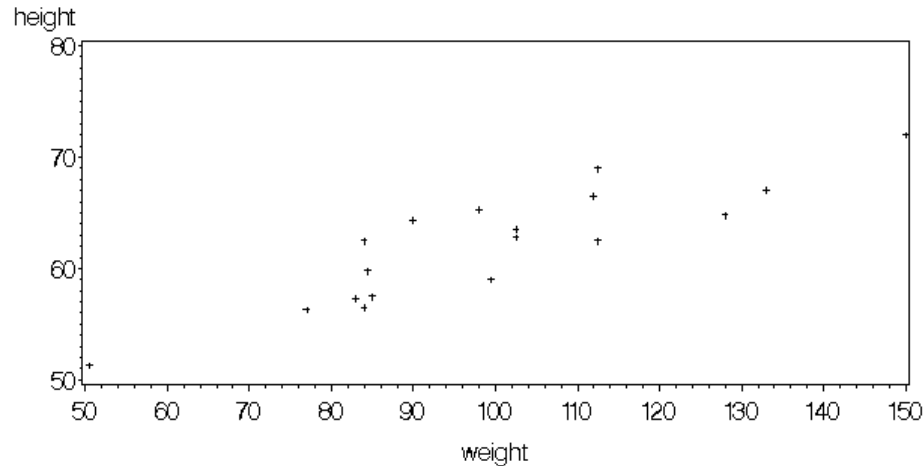
HAXIS=
HMINOR=
REGEQN
VAXIS=

```

Other features:

RUN-group processing
 SYMBOL statement
 Sample library member: GPLVRBL1

Study of Height vs Weight



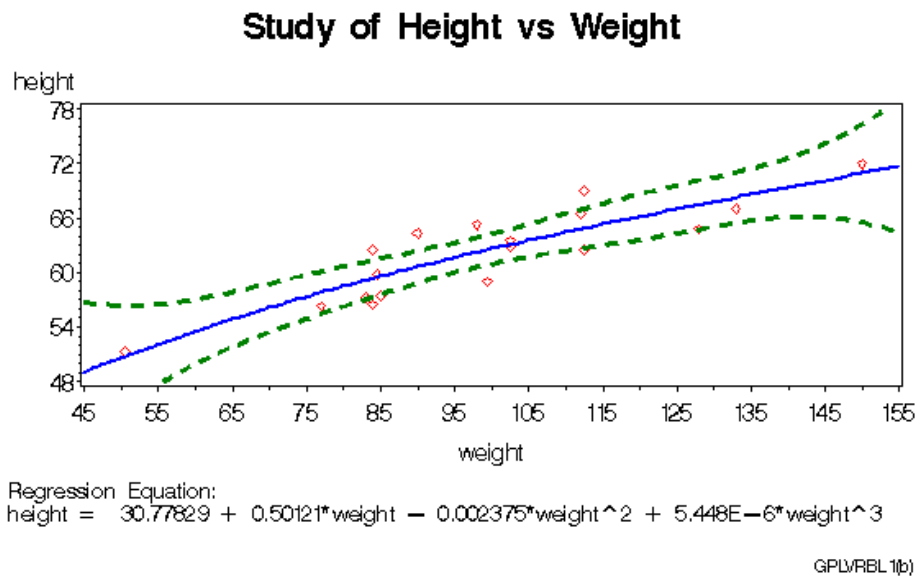
Source: T. Lewis & L. R. Taylor

Introduction to Experimental Ecology

GPLVREL1(a)

In this example, the PLOT statement uses a plot request of the type *y-variable*x-variable* to plot the variable HEIGHT against the variable WEIGHT. The plot shows that weight generally increases with size.

This example then requests the same plot with some modifications. As shown by the following output, the second plot request specifies a regression analysis with confidence limits, and scales the range of values along the vertical and horizontal axes. It also displays the regression equation specified for the SYMBOL statement. Because the procedure supports RUN-group processing, you do not have to repeat the PROC GPLOT statement to generate the second plot.

**Set the graphics environment.**

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftitle=swissb ftext=swiss htitle=6 htext=4;
```

Create the data set. STATS contains the heights and weights of numerous individuals.

```
data stats;
  input height weight;
  datalines;
69.0 112.5
56.5 84.0
...more data lines...
67.0 133.0
57.5 85.0
;
```

Define title and footnotes.

```
title 'Study of Height vs Weight';
footnote1 h=3 j=1 ' Source: T. Lewis & L. R. Taylor';
footnote2
  h=3 j=1 ' Introduction to Experimental Ecology'
  j=r 'GPLVRBL1(a) ';
```

Generate a default scatter plot.

```
proc gplot data=stats;
  plot height*weight;
```

```
run;
```

Redefine footnotes to make room for the regression equation.

```
footnote1; /* this clears footnote1 */
footnote2 h=3 j=r 'GPLVRBL1(b) ';
```

Define symbol characteristics. INTERPOL= specifies a cubic regression analysis with confidence limits for mean predicted values. VALUE=, HEIGHT=, and CV= specify a plot symbol, size, and color. CI=, CO=, and WIDTH= specify colors and a thickness for the interpolation and confidence-limits lines.

```
symbol1 interpol=rcclm95
      value=diamond
      height=3
      cv=red
      ci=blue
      co=green
      width=2;
```

Generate scatter plot with regression line. HAXIS= and VAXIS= define the range of axes values. HMINOR= specifies one minor tick mark between major tick marks. REGEQN displays the regression equation specified on the SYMBOL1 statement.

```
plot height*weight / haxis=45 to 155 by 10
                    vaxis=48 to 78 by 6
                    hminor=1
                    regeqn;

run;
quit;
```

Example 5: Connecting Plot Data Points

Procedure features:

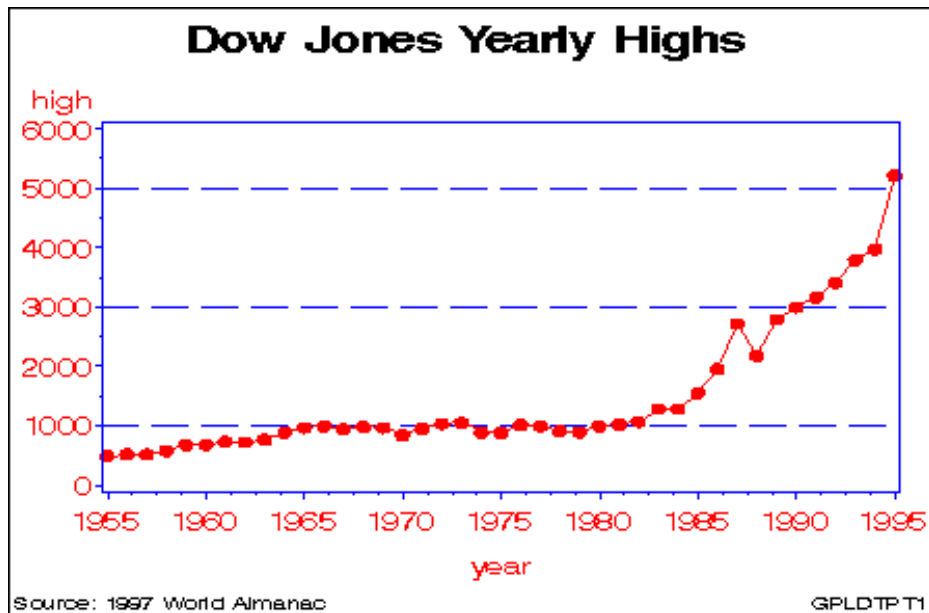
PLOT statement option:

```
CAXIS=
CTEXT
CVREF
HAXIS
HMINOR=
LVREF=
VAXIS=
VMINOR=
VREF
```

Other features:

SYMBOL statement

Sample library member: GPLDTPT1



In this example, the PLOT statement uses a plot request of the type *y-variable*x-variable* to plot the variable HIGH against the variable YEAR to show the annual highs of the Dow Jones Industrial Average over several decades.

This example uses a SYMBOL statement to specify a plot symbol and connect data points with a straight line. In addition, the example shows how PLOT statement options can add reference lines and modify the axes (AXIS statements are not used).

Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftitle=swissb ftext=swiss htitle=6 htext=4;
```

Create the data set. STOCKS contains yearly highs and lows for the Dow Jones Industrial Average, and the dates of the high and low values each year.

```
data stocks;
  input year @7 hdate date9. @15 high
        @24 ldate date9. @32 low;
  format hdate ldate date9.;
  datalines;
1955 30DEC55 488.40 17JAN55 388.20
1956 06APR56 521.05 23JAN56 462.35
...more data lines...
```



```

1994 31JAN94 3978.36 04APR94 3593.35
1995 13DEC95 5216.47 30JAN95 3832.08
;

```

Define title and footnote.

```

title1 'Dow Jones Yearly Highs';
footnotel h=3 j=1 ' Source: 1997 World Almanac'
          j=r ' GPLDTPT1 ';

```

Define symbol characteristics. SYMBOL1 defines the symbol that marks the data points and specifies its height and color. INTERPOL=JOIN joins the data points with straight lines.

```

symbol1 color=red
        interpol=join
        value=dot
        height=3;

```

Generate the plot and modify the axis values. HAXIS= sets major tick marks for the horizontal axis. VAXIS= sets major tick marks for the vertical axis. HMINOR= and VMINOR= specify the number of tick marks between major tick marks.

```

proc gplot data=stocks;
  plot high*year / haxis=1955 to 1995 by 5
                  vaxis=0 to 6000 by 1000
                  hminor=3
                  vminor=1

```

Add reference lines and specify colors. VREF= draws reference lines on the vertical axis at three marks. LVREF= specifies the line style (dashed) for the lines; CVREF= specifies blue as the line color. CAXIS= colors the axis lines and all major and minor tick marks. CTEXT= specifies red for all plot text, including axis labels and major tick mark values.

```

          vref=1000 3000 5000
          lvref=2
          cvref=blue
          caxis=blue
          ctext=red;

run;
quit;

```

Example 6: Generating an Overlay Plot

Procedure features:

PLOT statement options:

LEGEND=

OVERLAY

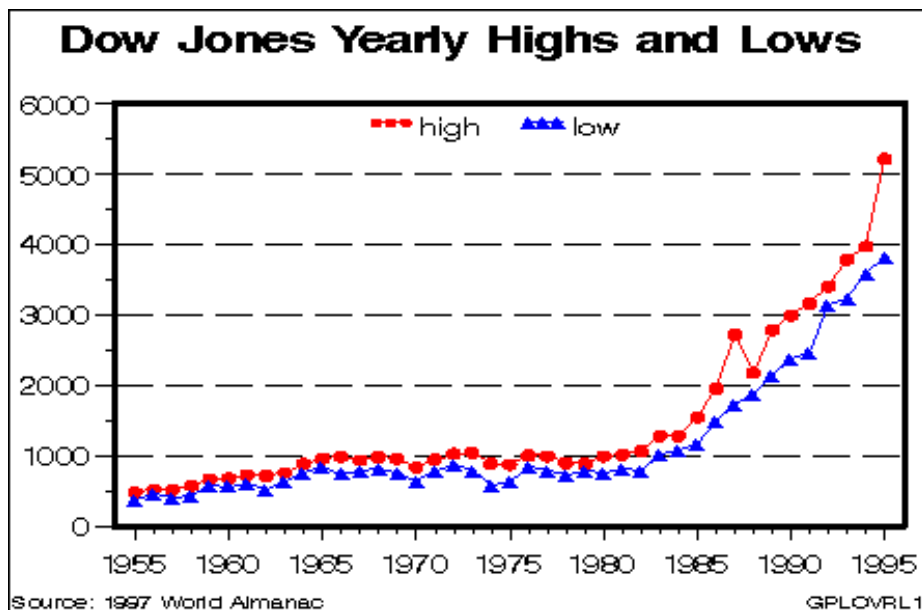
Other features:

LEGEND statement

SYMBOL statement

Data set: STOCKS on page 1130

Sample library member: GPLOVRL1



In this example, one PLOT statement plots both the HIGH and LOW variables against the variable YEAR using two plot requests. The OVERLAY option on the PLOT statement determines that both plot lines appear on the same graph. The other PLOT options scale the vertical axis, add a reference line to the plot, and specify the number of minor tick marks on the axes. The SYMBOL, AXIS, and LEGEND statements modify the plot symbols, axes, and legend.

Note: If the OVERLAY option were not specified, each plot request would generate a separate graph. \triangle

Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
colors=(black blue green red)
ftitle=swissb ftext=swiss htitle=6 htext=4;
```

Define title and footnote.

```
title1 'Dow Jones Yearly Highs and Lows';
footnote1 h=3 j=1 ' Source: 1997 World Almanac'
j=r 'GPLOVRL1 ';
```

Example 7: Filling Areas in an Overlay Plot

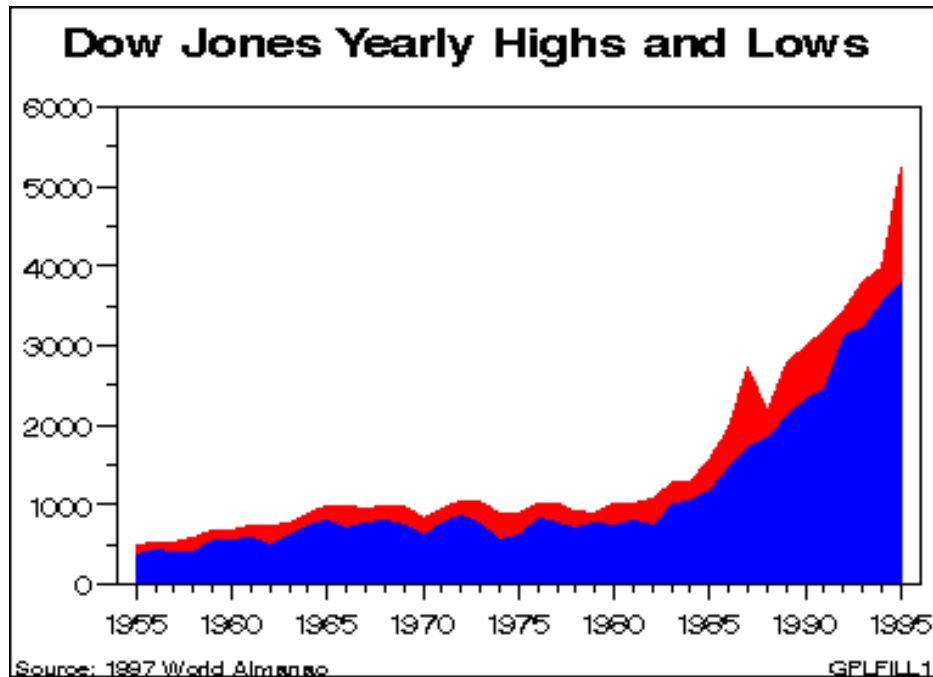
Procedure features:

PLOT statement options:

AREAS=
OVERLAY**Other features:**

GOPTIONS statement

SYMBOL statement

Data set: STOCKS on page 1130**Sample library member:** GPLFILL1

This example uses the AREAS= option in the PLOT statement to fill the areas that are under the plot lines. As in the previous example, two plots are overlaid on the same graph.

Set the graphics environment. COLORS= sets the area colors. CTEXT= sets the color for all text.

```
goptions reset=global gunit=pct border cback=white
        colors=(blue red) ctext=black
        ftitle=swissb ftext=swiss htitle=6 htext=4;
```

Define title and footnote.

```

title1 'Dow Jones Yearly Highs and Lows';
footnote1 h=3 j=1 ' Source: 1997 World Almanac'
          j=r 'GPLFILL1 ';

```

Define symbol characteristics. INTERPOL= specifies a line to connect data points. The line creates the fill boundary.

```

symbol1 interpol=join;

```

Define axis characteristics.

```

axis1 order=(1955 to 1995 by 5) offset=(2,2)
      label=none
      major=(height=2)
      minor=(height=1);
axis2 order=(0 to 6000 by 1000) offset=(0,0)
      label=none
      major=(height=2)
      minor=(height=1);

```

Generate a plot with filled areas. The plot requests are ordered to draw the lowest plot first. Area 1 occupies the space between the lowest (first) plot line and the horizontal axis, and area 2 is below the highest (second) plot line. This arrangement prevents the pattern for area 1 from overlaying the pattern for area 2. AREAS=2 fills all the areas below the second plot line.

```

proc gplot data=stocks;
  plot low*year high*year / overlay
                                haxis=axis1
                                hminor=4
                                vaxis=axis2
                                vminor=1
                                caxis=black
                                areas=2;

run;
quit;

```

Example 8: Plotting Three Variables

Procedure features:

PLOT classification variable

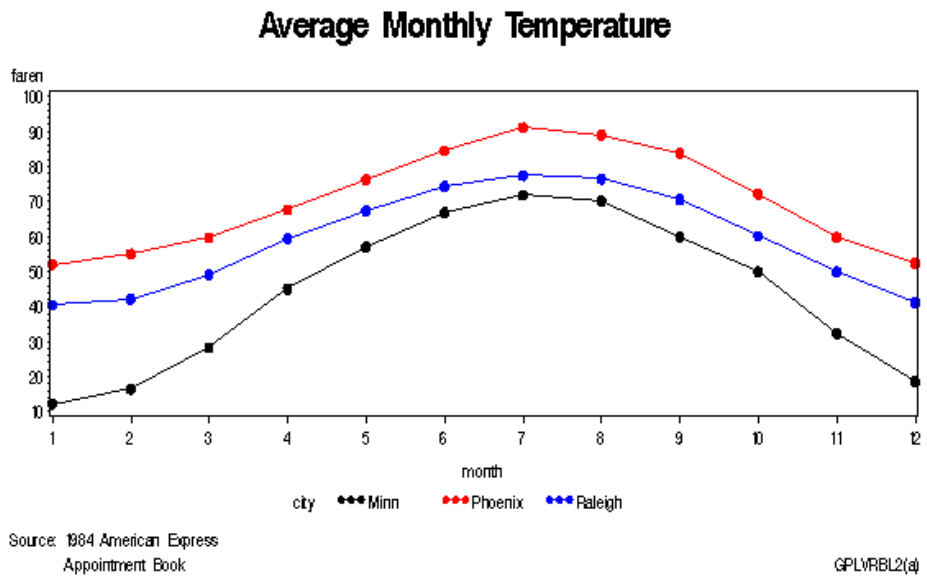
Other features:

AXIS statement

SYMBOL statement

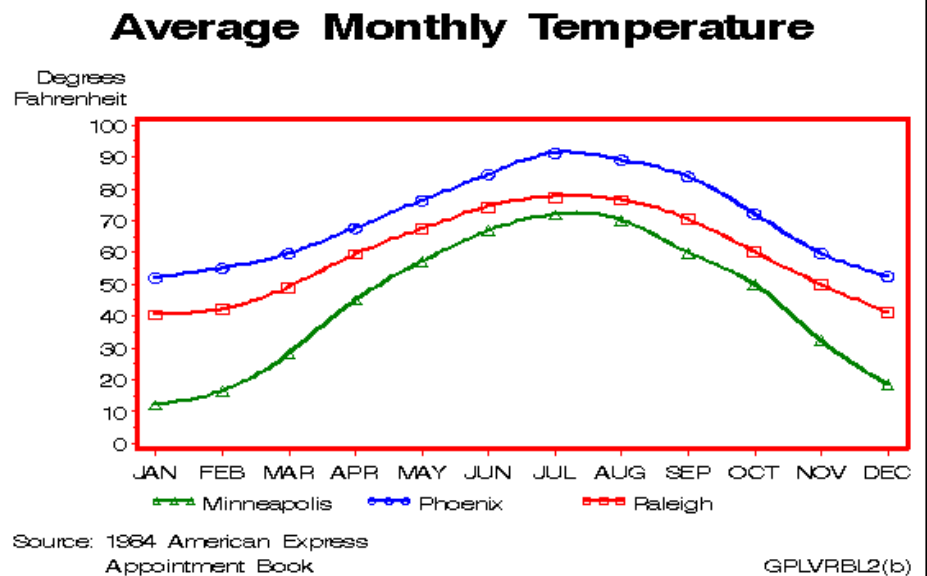
RUN-group processing

Sample library member: GPLVRBL2



This example shows that when your data contain a classification variable that groups the data, you can use a plot request of the form *y-variable*x-variable=third-variable* to generate a separate plot for every formatted value of the classification variable, which in this case is CITY. With this type of request, all plots are drawn on the same graph and a legend is automatically produced and explains the values of *third-variable*. The default legend uses the variable name CITY for the legend label and the variable values for the legend value descriptions. Because no LEGEND definition is used in this example, the font and height of the legend label and the legend value descriptions are set by the graphics options FTEXT= and HTEXT=. Height specifications in the SYMBOL statement do not affect the size of the symbols in the legend values.

This example then modifies the plot request. As shown in the following output, the plot is enhanced by using different symbol definitions for each plot line, changing axes labels, and scaling the vertical axes differently.



Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftitle=swissb ftext=swiss htitle=6 htext=3;
```

Create the data set. CITYTEMP contains the average monthly temperatures of three cities: Raleigh, Minneapolis, and Phoenix.

```
data citytemp;
  input  month faren city $;
  datalines;
  1      40.5   Raleigh
  1      12.2   Minn
  1      52.1   Phoenix
  ...more data lines...
  12     41.2   Raleigh
  12     18.6   Minn
  12     52.5   Phoenix
;
```

Define title and footnote.

```
title1 'Average Monthly Temperature';
footnote1 j=1 ' Source: 1984 American Express';
footnote2 j=1 '           Appointment Book'
           j=r 'GPLVRBL2(a) ';
```

Define symbol characteristics. This statement specifies that a straight line connect data points, and that the data points be represented by a 3-unit-high dot. Because no color is specified, the default color behavior is used and each line is a different color.

```
symbol1 interpol=join
        value=dot
        height=3;
```

Generate a plot of three variables. The plot request draws one plot on the graph for each value of CITY and produces a legend that defines CITY values.

```
proc gplot data=citytemp;
  plot faren*month=city / hminor=0;
run;
```

Modify FOOTNOTE2 to reference new output.

```
footnote2 j=1 '           Appointment Book'
           j=r 'GPLVRBL2(b) ';
```

Define new symbol characteristics. SYMBOL statements are assigned to the values of CITY in alphabetical order. For example, the value **Minn** is assigned SYMBOL1.

```
symbol1 color=green interpol=spline
        width=2 value=triangle
        height=3;
symbol2 color=blue interpol=spline
        width=2 value=circle
        height=3;
symbol3 color=red interpol=spline
        width=2 value=square
        height=3;
```

Define new axis characteristics. AXIS1 suppresses the axis label and specifies month abbreviations for the major tick mark labels. AXIS2 specifies a two-line axis label and scales the axis to show major tick marks at every 10 degrees from 0 to 100 degrees.

```
axis1 label=none
      value=('JAN' 'FEB' 'MAR' 'APR' 'MAY' 'JUN'
            'JUL' 'AUG' 'SEP' 'OCT' 'NOV' 'DEC')
      order = 1 to 12 by 1
      offset=(2)
      width=3;
axis2 label=('Degrees' justify=right 'Fahrenheit')
      order=(0 to 100 by 10)
      width=3;
```

Enhance the legend.

```
legend1 label=none value=(tick=1 'Minneapolis');
```

Generate the enhanced plot. Because the procedure supports RUN-group processing, you do not have to repeat the PROC GPLOT statement to generate the second plot.

```
plot faren*month=city / haxis=axis1 hminor=0
                        vaxis=axis2 vminor=1
                        caxis=red legend=legend1;

run;
quit;
```

Example 9: Plotting with Different Scales of Values

Procedure features:

PLOT statement options:

HAXIS=

HMINOR=

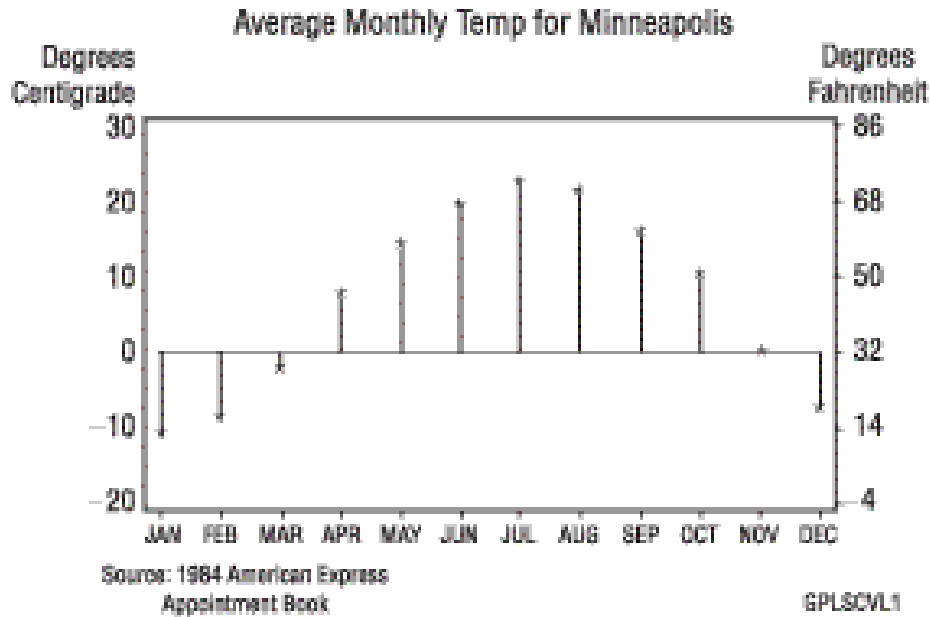
PLOT and PLOT2 statement options:

CAXIS=
VAXIS=
VMINOR=

Other features:

AXIS statement
SYMBOL statement

Sample library member: GPLSCVL1



This example shows how a PLOT2 statement generates a right axis that displays the values of the vertical coordinates in a different scale from the scale that is used for the left axis.

In this plot of the average monthly temperature for Minneapolis, temperature variables that represent degrees centigrade (displayed on the left axis) and degrees Fahrenheit (displayed on the right axis) are plotted against the variable MONTH. Although the procedure produces two sets of data points, it calibrates the axes so that the data points are identical and it displays only one plot.

This example uses SYMBOL statements to define symbol definitions. By default, the SYMBOL1 statement is assigned to the plot that is generated by the PLOT statement, and SYMBOL2 is assigned to the plot generated by the PLOT2 statement.

Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftitle=swissb ftext=swiss htitle=6 htext=3;
```

Create the data set and calculate centigrade temperatures. MINNTEMP contains average monthly temperatures for Minneapolis.

```

data minntemp;
  input @10 month
        @23 f2;
  c2=(f2-32)/1.8;
  output;
  datalines;
01JAN83 1 1 40.5 12.2 52.1
01FEB83 2 1 42.2 16.5 55.1
  ...more data lines...
01NOV83 11 4 50.0 32.4 59.8
01DEC83 12 1 41.2 18.6 52.5
;

```

Define title and footnote.

```

title1 'Average Monthly Temperature for Minneapolis';
footnote1 j=1 ' Source: 1984 American Express';
footnote2 j=1 '           Appointment Book'
           j=r 'GPLSCVL1 ' ;

```

Define symbol characteristics. INTERPOL=NEEDLE generates a horizontal reference line at zero on the left axis and draws vertical lines from the data points to the reference line. CI= specifies the color of the interpolation line and CV= specifies the color of the plot symbol.

```

symbol1 interpol=needle
        ci=blue
        cv=red
        width=3
        value=star
        height=3;

```

Define symbol characteristics for PLOT2. SYMBOL2 suppresses interpolation lines and plotting symbols; otherwise, they would overlay the lines or symbols displayed by SYMBOL1.

```

symbol2 interpol=none
        value=none;

```

Define axis characteristics. In the AXIS2 and AXIS3 statements, ORDER= controls the scaling of the axes. Both axes represent exactly the same range of temperature, and the distance between the major tick marks on both axes represent an equivalent quantity of degrees (10 for centigrade and 18 for Fahrenheit).

```

axis1 label=none
      value=('JAN' 'FEB' 'MAR' 'APR' 'MAY' 'JUN'
            'JUL' 'AUG' 'SEP' 'OCT' 'NOV' 'DEC')
      offset=(2)
      width=3;
axis2 label=('Degrees' justify=right ' Centigrade')
      order=(-20 to 30 by 10)
      width=3;

```

```
axis3 label=(h=3 'Degrees' justify=left 'Fahrenheit')
      order=(-4 to 86 by 18)
      width=3;
```

Generate a plot with a second vertical axis. HAXIS= specifies the AXIS1 definition. VAXIS= specifies AXIS2 and AXIS3 definitions in the PLOT and PLOT2 statements. CAXIS= colors the axis lines and all major and minor tick marks. Axis labels and major tick mark values use the default color. VMINOR= specifies the number of minor tick marks for each axis.

```
proc gplot data= minntemp;
  plot c2*month / caxis=red
                haxis=axis1 hminor=0
                vaxis=axis2 vminor=1;
  plot2 f2*month / caxis=red
                vaxis=axis3
                vminor=1;
run;
quit;
```

Example 10: Creating Plots with Drill-down for the Web

Procedure features:

PLOT statement options:

```
HTML=
HTML_LEGEND=
```

ODS features:

ODS HTML statement:

```
BODY=
NOGTITLE
PATH=
```

Other features:

```
BY statement
GOPTIONS statement
```

Sample library member: GPLDRIL1

This example shows how to create a plot with simple drill-down functionality for the Web. If you display the plot in a Web browser, you can select any plot point or legend symbol to display a report on monthly temperatures for the selected city.

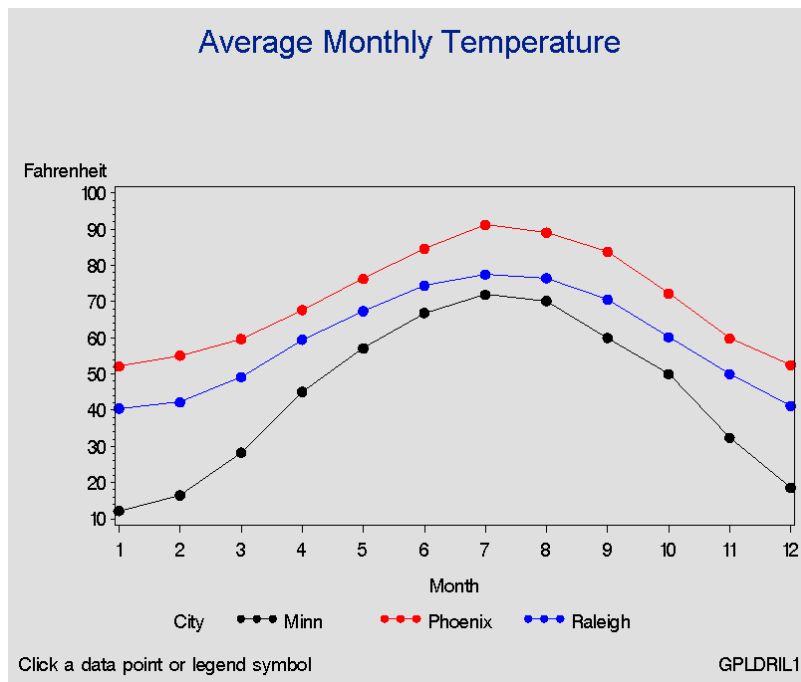
The example explains how to use an ODS statement such as ODS HTML to generate a graph with drill-down links. It shows how to:

- explicitly name the HTML files and direct the different types of output to different files
- use BY-group processing with ODS, and determine the anchor names for the different pieces of output
- use the PATH= option to specify the destination for the HTML and GIF files created by the ODS statement

- add an HTML HREF string to a data set to define a link target
- assign link targets with the HTML= and HTML_LEGEND= procedure options
- suppress the titles in the GIF files and display them in the HTML file.

For more information on drill-down graphs, see “Adding Drill-Down Links to Web Presentations” on page 571.

This program modifies the code from sample GPLVRBL2, which shows how to generate separate plots for the formatted values of a classification variable. In this example, the code implements drill-down capability for the plot, enabling you to select any plot point or legend symbol to drill down to a report on the yearly temperatures for the corresponding city. The following figure shows the drill-down plot as it is viewed in a Browser.



The following figure shows the report that appears when you select any plot point or legend symbol that corresponds to the data for Raleigh.

Monthly Temperatures in Raleigh

Month	Fahrenheit
1	40.5
2	42.2
3	49.2
4	59.5
5	67.4
6	74.4
7	77.5
8	76.5
9	70.6
10	60.2
11	50
12	41.2

Assign the fileref to the Web-server path. FILENAME assigns the fileref ODSOUT, which specifies a destination for the HTML and GIF files produced by the example program. ODSOUT must point to a Web-server location if procedure output is to be viewed on the Web. Later in the program, PATH=ODSOUT is specified on the ODS HTML statement, which directs program output to that location.

```
filename odsout 'path to Web server space';
```

Close the ODS listing destination for output. To conserve system resources, use ODS LISTING to close the Listing destination for procedure output. Thus, the graphics output is not displayed in the GRAPH window, although it is written to the catalog.

```
ods listing close;
```

Assign graphics options for producing the ODS output. DEVICE=GIF causes ODS to generate the graphics output as GIF files. TRANSPARENCY causes the graphics output to use the Web-page background as the background of the graph. NOBORDER suppresses the border around the graphics output area, which makes the border treatment the same as that for the non-graphics output that is generated by the example.

```
goptions reset=global gunit=pct
        colors=(black red blue green)
        ftext=swiss ftitle=swissb htitle=6 htext=3
        device=gif transparency noborder;
```

Open the HTML destination. PATH= specifies the ODSOUT fileref as the HTML destination for all the HTML and GIF files produced by the program. BODY= names the HTML file for storing the drill-down plot. NOGTITLE suppresses the graph title from the SAS/GRAPH output and displays it through the HTML page. ODS automatically assigns anchor names to each piece of output that is generated while the HTML destination is open.

```
ods html path=odsout
        body='city_plots.html'
        nogtitle;
```

Create the data set CITYTEMP. CITYTEMP contains the average monthly temperatures for three cities.

```
data citytemp;
  input Month Fahrenheit City $;
  datalines;
  1      40.5    Raleigh
  1      12.2    Minn
  1      52.1    Phoenix
  2      42.2    Raleigh
  2      16.5    Minn
  2      55.1    Phoenix
  3      49.2    Raleigh
  3      28.3    Minn
  3      59.7    Phoenix
  4      59.5    Raleigh
  4      45.1    Minn
  4      67.7    Phoenix
  5      67.4    Raleigh
  5      57.1    Minn
  5      76.3    Phoenix
  6      74.4    Raleigh
  6      66.9    Minn
  6      84.6    Phoenix
  7      77.5    Raleigh
  7      71.9    Minn
  7      91.2    Phoenix
  8      76.5    Raleigh
  8      70.2    Minn
  8      89.1    Phoenix
  9      70.6    Raleigh
  9      60.0    Minn
  9      83.8    Phoenix
  10     60.2    Raleigh
  10     50.0    Minn
  10     72.2    Phoenix
  11     50.0    Raleigh
  11     32.4    Minn
  11     59.8    Phoenix
  12     41.2    Raleigh
  12     18.6    Minn
  12     52.5    Phoenix
  ;
```

Add the HTML variable to CITYTEMP and create the NEWTEMP data set. The HTML variable CITYDRILL contains the target locations to associate with the different values of the variable CITY. Each location for CITYDRILL references the file city_reports.html, which this program will create. Each location ends with the default anchor name (IDX1, IDX2, and IDX3) that ODS assigns to the target output when it creates that output in file city_reports.html.

```
data newtemp;
  set citytemp;
  length citydrill $ 40;
  if city='Minn' then
    citydrill='HREF="city_reports.html#IDX1"';
  else if city='Phoenix' then
    citydrill='HREF="city_reports.html#IDX2"';
  else if city='Raleigh' then
    citydrill='HREF="city_reports.html#IDX3"';
```

Define titles and footnotes and a symbol definition for the plots.

```
title1 'Average Monthly Temperature';
footnote1 j=1 h=3 ' Click a data point or legend symbol'
          j=r 'GPLDRILL1 ';

symbol1 interpol=join
          value=dot
          height=3;
```

Generate the plot. Both HTML= and HTML_LEGEND= specify CITYDRILL as the variable that contains the targets for the drill-down links. HTML= determines that each plot point will be a hot zone that links to target output, and HTML_LEGEND= determines that the legend symbols will be hot zones that link to target output. This GPLOT procedure generates the first piece of output in this program; thus, the plot receives the first default anchor name, which is IDX.

```
proc gplot data=newtemp;
  plot fahrenheit*month=city / hminor=0
      html=citydrill
      html_legend=citydrill;
run;
quit;
```

Change the HTML file. BODY= opens a new HTML file for storing the reports for city temperatures. The new file is assigned the name city_reports.html, which is the file name assigned above to variable CITYDRILL as part of its target-link locations. The reports that are generated later in this program will all be written to this one HTML file.

```
ods html path=odsout
      body='city_reports.html';
```

Sort data set NEWTEMP in order by city.

```
proc sort data=newtemp;
  by city month;
run;
```

Clear the footnotes, and suppress the default BY-line.

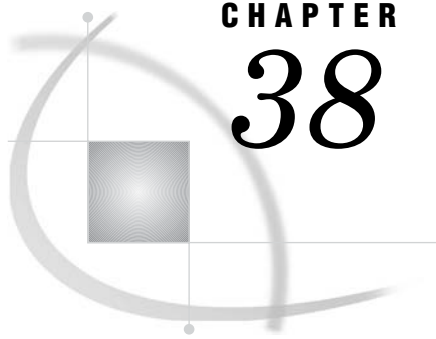
```
goptions reset=footnote;  
option nobyline;
```

Print a report of monthly temperatures for each city. The BY statement determines that a separate report is generated for each city. Thus, the REPORT procedure generates three pieces of output. To assign anchor locations to this new output, ODS increments the last anchor name that was used (IDX), and therefore assigns the anchor names IDX1, IDX2, and IDX3 to the output. These are the anchor locations that were specified above as the anchor locations for variable CITYDRILL.

```
title1 'Monthly Temperatures in #byval(city)';  
proc report data=newtemp nowindows;  
  by city;  
  column city month fahrenheit;  
  define city      / noprint group;  
  define month     / display group;  
  define Fahrenheit / display group;  
run;
```

Close the HTML destination, and open the LISTING destination.

```
ods html close;  
ods listing;
```

CHAPTER 38

The GPRINT Procedure

<i>Overview</i>	1147
<i>Concepts</i>	1148
<i>About External Text Files</i>	1148
<i>Procedure Syntax</i>	1148
<i>PROC GPRINT Statement</i>	1148
<i>Examples</i>	1153
<i>Example 1: Specifying Color Text</i>	1153
<i>Example 2: Adjusting the Size of Characters</i>	1156

Overview

The GPRINT procedure converts a text file into graphics output that can be displayed or printed on a graphics output device. You can enhance the output with TITLE, NOTE, and FOOTNOTE statements, or include Annotate graphics, or both. Like output from any other SAS/GRAPH procedure, output from the GPRINT procedure can be stored in catalogs and replayed with the GREPLAY procedure.

You can use the GPRINT procedure when you want to create graphics output from tabular material, reports, or any external text file produced by the SAS System or other software application. To display text and graphics generated by SAS/GRAPH software, use the GSLIDE procedure.

Figure 38.1 on page 1147 shows a graphics output generated by the GPRINT procedure from SAS output generated by the MEANS procedure. Titles and footnotes have been added, and the Swiss font has been assigned to the procedure output text.

Figure 38.1 Graph Generated with the GPRINT Procedure

Regional Sales Report	
Region	Staff
NE	4
NW	3
SE	2
SW	4

Concepts

About External Text Files

External text files are files that you have stored outside of SAS. They can be created in several different ways. Four common methods are as follows:

- save the contents of the OUTPUT or LOG window to an external file with the FILE command
- direct the output from SAS procedures to an external file using the PRINTTO procedure and a FILENAME statement
- direct the output from a SAS data step to an external file using the FILE and PUT statements
- create a text file from another software application such as a text editor or a spreadsheet program.

Note: Depending on the operating environment and the method used to generate the file, external text files may contain carriage-control characters. For more information on carriage-control characters, see the NOCC option on page 1150. \triangle

You can use a FILENAME statement or host command to specify a fileref that points to the location of the external text file that you want to print. This external file serves as the input file for the GPRINT procedure.

Procedure Syntax

Global statements: FOOTNOTE, GOPTIONS, TITLE

Reminder: The procedure can include the NOTE statement.

Supports: Output Delivery System (ODS)

```
PROC GPRINT FILEREF=fileref
      <option(s)>;
```

PROC GPRINT Statement

The PROC GPRINT statement identifies the external file to be converted to graphics output. Optionally, specifies the text color, a destination catalog for graphics output, and an Annotate data set.

Syntax

```
PROC GPRINT FILEREF=fileref
      <option(s)>;
```

option(s) can be one or more of the following:

```

ANNOTATE=Annotate-data-set
CTEXT=text-color
DESCRIPTION='entry-description'
GOUT=<libref.>output-catalog
NAME='entry-name'
NOCC
O

```

Required Arguments

FILEREF=*fileref*

specifies the fileref that is associated with the external file that will be used as input to the GPRINT procedure. *Fileref* must have been previously defined in a FILENAME statement or host command.

See also: “FILENAME Statement” on page 28.

Featured in: Example 2 on page 1156.

Options

Options in the PROC GPRINT statement affect all graphs that the statement produces. You can specify as many options as you want and list them in any order.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate the output that the GPRINT procedure produces.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587.

CTEXT=*text-color*

specifies the color in which the procedure displays the text from the input file.

If you do not use the CTEXT= option, a color specification is searched for in the following order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the default, the first color in the colors list.

The CTEXT= option in the PROC GPRINT statement does not affect titles and footnotes generated by TITLE and FOOTNOTE definitions.

Featured in: Example 1 on page 1153.

DESCRIPTION=*'entry-description'*

DES=*'entry-description'*

specifies the description of the catalog entry for the chart. The maximum length for *entry-description* is 256 characters. The description does not appear on the chart. By default, the GPRINT procedure assigns the description OUTPUT FROM PROC GPRINT.

GOUT=<*libref.*>*output-catalog*

specifies the SAS catalog in which to save the graphics output produced by the GPRINT procedure. If you omit the libref, SAS/GRAPH looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist.

See also: “Storing Graphics Output in SAS Catalogs” on page 53.

NAME=*'entry-name'*

specifies the name of the catalog entry for the graph. The maximum length for *entry-name* is 8 characters. The default name is GPRINT. If the specified name duplicates the name of an existing entry, SAS/GRAPH software adds a number to the duplicate name to create a unique entry, for example, GPRINT1.

NOCC

tells the procedure that the external text file does not contain carriage-control characters. If you include the NOCC option, the procedure assumes that the first character on each line of the input file is a text character and not a carriage-control character. If you omit the NOCC option, the characters in column one are read as carriage-control characters. If they are valid carriage-control characters, the GPRINT procedure recognizes and executes them. If they are not valid carriage-control characters, the GPRINT procedure issues an error message.

O

causes a 0 (numeric zero) to be converted to the letter O in the output. This option circumvents the use of a numeric zero with an interior slash that is present on some devices.

Adjusting SAS Output and Graphics Output

The size of SAS output (or other text) in columns and rows and the size of graphics output are independently controlled. Depending on the result you want, you can do either of the following:

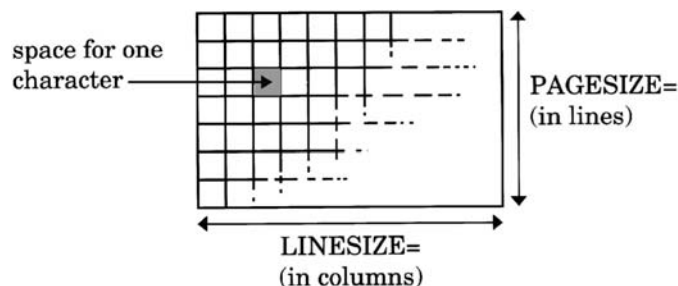
- Adjust the size of your SAS output (or other text) to fit the available space on your graph.
- Adjust the dimensions of the graphics output area and the size of the cells within the graphics output area to control the size of the characters that are displayed as graphics output by the GPRINT procedure.

You can adjust the size (columns and rows) of any other external text file that you use as input to the GPRINT procedure. Although the following sections explain how to adjust the size of SAS output, the general process can be applied to any text file.

SAS Output Size

SAS output prints in pages. The length (in number of rows) and the width (in number of columns) of the page are determined by the PAGESIZE= and LINESIZE= options, respectively. Each character of SAS output occupies one column of space in a row (one cell), as shown in Figure 38.2 on page 1150.

Figure 38.2 SAS Output Size

**Graphics Output Size**

Graphics output is drawn in the graphics output area, which is also divided into cells. The overall dimensions of the graphics output area (width and height) are determined by the values of the device parameters XMAX and YMAX. These values, which determine the aspect ratio of the graphics output area, can be temporarily reduced with the HSIZE= and VSIZE= graphics options.

The number of columns and rows that fill the area is determined by the values of the LCOLS or PCOLS and LROWS or PROWS device parameters. These values, which determine the size and aspect ratio of a cell, can be temporarily altered with the HPOS= and VPOS= graphics options. The more columns and rows there are in a given area, the smaller the cells are. Therefore, using HPOS= and VPOS= to change the number of columns and rows also changes the size of the cells and may change the size of the characters. However, it does not affect the overall dimensions of the graph. For details, see “Maintaining the aspect ratio of cells” on page 1152.

See “Procedure Output and the Graphics Output Area” on page 34 for a complete description of the graphics output area and Chapter 2, “SAS/GRAPH Programs,” on page 25 for information on device parameters and graphic options.

Matching Sizes

When you use the GPRINT procedure to convert SAS output to graphics output, you may need to manipulate the dimensions of either or both to get the proper size characters in the graphics output and to avoid truncating lines. Adjustment may be necessary in the following situations:

- If the number of rows per page in the SAS output (PAGESIZE=) exceeds the number of rows in the graphics output area (LROWS or PROWS), then the GPRINT procedure produces additional pages of graphics output.
- If the number of rows per page in the SAS output (PAGESIZE=) is much less than the number of rows in the graphics output area (LROWS or PROWS), then the output does not fill the graphics output area.
- If the width of a line of SAS output (LINESIZE=) exceeds the number of columns in the graphics output area (LCOLS or PCOLS), then the GPRINT procedure truncates the line.
- If the width of SAS output (LINESIZE=) is much less than the number of columns in the graphics output area (LCOLS or PCOLS), then the output does not fill the graphics output area.

You can adjust the size of the SAS output or the size of the graphics output, or both.

Adjusting the size of the SAS output

The following steps show you how to use the PAGESIZE= and LINESIZE= options to adjust the page size of the SAS output to fit the size of the graphics output area.

- 1 Use the GDEVICE procedure to determine the number of rows (LROWS or PROWS) and the number of columns (LCOLS or PCOLS) on the graphics device that you intend to use. See Chapter 31, “The GDEVICE Procedure,” on page 915 for details.
- 2 Determine the number of columns and rows that you are going to use for SAS/GRAPH titles and footnotes. (If you specify height in units of CELLS, each unit of height equals one row.)
- 3 Use the OPTIONS statement to set the PAGESIZE= option equal to the number of rows on the device minus the number of positions to be used by TITLE and FOOTNOTE definitions. Set the LINESIZE= option equal to the number of columns on the device minus the number of positions used by titles and footnotes if the titles and footnotes are positioned vertically.
- 4 Produce the SAS output.

Adjusting the size of the graphics output

The following steps show you how to use the HPOS= and VPOS= graphics options to adjust the number of columns and rows in the graphics output area on the output device so that it can accommodate the page size of your SAS output.

- 1 Determine the number of columns (LINESIZE=) and rows (PAGESIZE=) in the SAS output.
- 2 Use the GOPTIONS statement to set the VPOS= graphics option equal to the number of rows in the SAS output plus the number of rows to be used by TITLE and FOOTNOTE definitions. Set the HPOS= graphics option equal to the number of columns in the SAS output plus the number of columns to be used by titles and footnotes if the titles and footnotes are positioned vertically.
- 3 Produce the GPRINT output.

Similarly, adjusting the overall dimensions of the graphics output area with the HSIZE= and VSIZE= graphics options may affect the size and possibly the aspect ratio of the cells.

Note: Changing the values of the HPOS= and VPOS= graphics options changes the size of the cells and consequently of characters in the output. On devices with nonscalable hardware fonts, changing the aspect ratio with HPOS= and VPOS= causes the Simulate font to be used instead of hardware characters. However, if you specify software fonts, the change in aspect ratio may be ignored. See “Using Fonts” on page 1152 and “Using Hardware Fonts” on page 78 for more information . \triangle

Maintaining the aspect ratio of cells

If you change the values of the HPOS= and VPOS= graphics options to control the size of characters or to match the rows and columns of the external text file, you should try to maintain the same ratio of columns to rows as the original values of the device parameters. For example, suppose you have SAS output with 50 columns and 10 rows, and a graphics device that has 80 columns and 32 rows. The aspect ratio of the device is 5:2. If you print 10 rows of output on a device with 32 rows, you will have 22 blank lines. You can reduce the number of blank lines and increase the size of the characters by reducing the number of rows in the graphics output area with VPOS=. If, in addition to the 10 rows of output, you allow four lines of space for titles and two lines of space for a footnote, you need a total of 16 rows. Therefore, assigning a value of 20 to VPOS= should produce readable text and plenty of space. If VPOS=20, setting HPOS= to 50 retains the original aspect ratio of the device (80:32 or 5:2).

Note that this method allows space for titles and footnotes in terms of rows; the actual size of the titles and footnotes depends on the height specification you use. Using the unit CELLS to define the height of titles and footnotes makes it easier to calculate precisely how much space is available.

Using Fonts

By default, the GPRINT procedure uses the default hardware font with a height of 1 cell to display the text from the external file. However, if you specify a nonscalable hardware font, SAS/GRAPH may use the Simulate font instead. See Chapter 5, “SAS/GRAPH Fonts,” on page 75 for details.

Font and height specifications for titles and footnotes are determined by the TITLE and FOOTNOTE definitions. See “TITLE, FOOTNOTE, and NOTE Statements” on page 210 for details.

To specify a font and height for the text, use the FTEXT= and HTEXT= graphics options. If you specify a software font, it is best to use a uniform font such as Swiss Uniform so that your text will be evenly spaced.

CAUTION:

Changes in the aspect ratio of cells made with the HPOS= and VPOS= graphics options are ignored if you specify software fonts. Change the aspect ratio in the device entry if you want the software characters proportioned to fit the new aspect ratio. \triangle

If you specify a software font and change the aspect ratio of the cells with the HPOS= and VPOS= graphics options, the change in aspect ratio is ignored and the procedure continues to draw the font in the original proportions. As a result, your text may not fit the graphics output area.

However, if you want the software characters to reflect a change in aspect ratio or if you want the characters to fit the new aspect ratio even if they are distorted, use the LCOLS or PCOLS and LROWS or PROWS device parameters in the device entry to change the aspect ratio of the cells. Using the device entry to specify a change in the aspect ratio enables you to distort the characters. See Chapter 31, “The GDEVICE Procedure,” on page 915 for more information on changing device parameters.

Examples

Example 1: Specifying Color Text

Procedure features:

GPRINT procedure options:

CTEXT=

Other features:

GOPTIONS statement

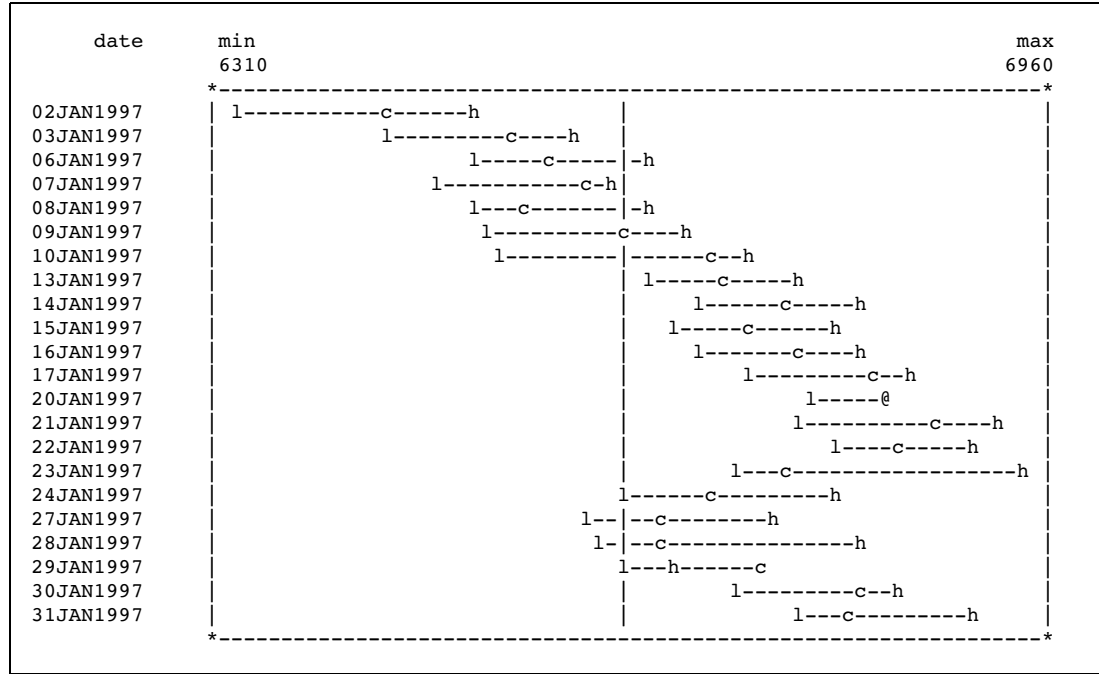
TIMEPLOT procedure

Sample library member: GPRCOLOR

This example creates the REFLIB.DOWHLC data set and generates a graph with color text from output that is produced by the TIMEPLOT procedure. The TIMEPLOT procedure is not a graphics procedure and produces text output only. (See *Base SAS Procedures Guide* for details on the TIMEPLOT procedure.)

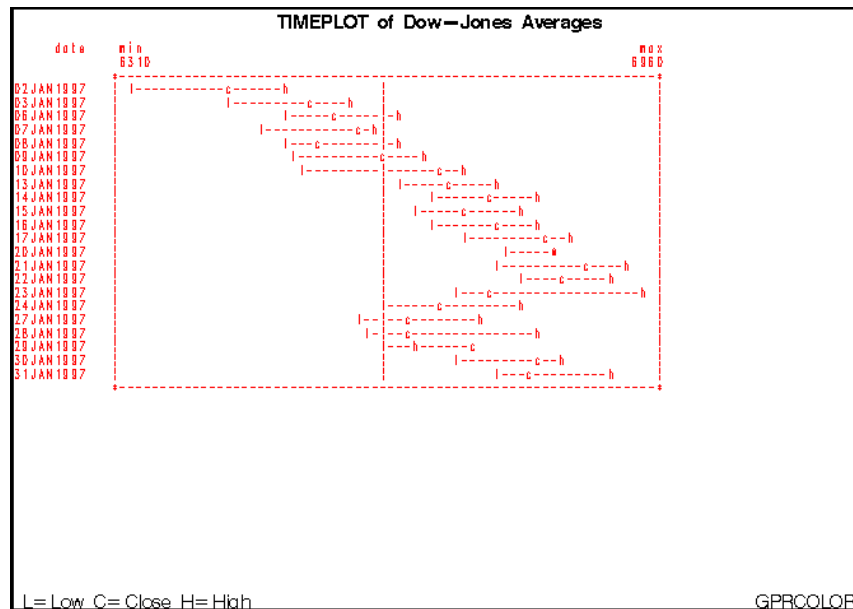
The first part of this example uses the TIMEPLOT procedure with the newly created REFLIB.DOWHLC data set as input to produce Output 38.1:

Output 38.1 SAS Output from the TIMEPLOT Procedure



The second part of this example takes the output generated by the TIMEPLOT procedure and converts it to a graph by using the GPRINT procedure. Figure 38.3 on page 1154 shows the graph with color text, a title, and a footnote:

Figure 38.3 GPRINT Procedure Output with Enhanced Text (GPRCOLOR)



Assign the libref and set the graphics environment. HTEXT= assigns the height for the text in the default unit, cells.

```
libname reflib 'SAS-data-library';
options reset=global border cback=white
        colors=(black blue green red)
        ftitle=swissb htitle=3pct
        htext=.8 ftext=none
        hsize=7in vsize=5in;
```

Assign the fileref OUT to the external file.

```
*filename out 'external-file';
```

Create the data set REFLIB.DOWHLC.

```
data reflib.dowhlc;
  input date date9. high low close;
  format date date9.;
  datalines;
02JAN1997   6511.38   6318.96   6442.49
03JAN1997   6586.42   6437.10   6544.09
...more data lines...
30JAN1997   6845.03   6719.96   6823.86
31JAN1997   6912.37   6769.99   6813.09
;
```

Suppress the date line and page numbers and set the linesize and pagesize.

```
options nodate nonumber linesize=80 pagesize=60;
```

Specify the destination for all subsequent procedure output.

```
proc printto print=out new;
run;
```

Generate TIMEPLOT graph output. It is sent to external file.

```
proc timeplot data=reflib.dowhlc;
  plot low close high / overlay hiloc ref=mean(low)
                                npp axis=6310 to 6960 by 10;
  id date;
run;
```

Reset destination for printed output to default.

```
proc printto;
run;
```

Define title and footnote.

```

title 'TIMEPLOT of Dow-Jones Averages';
footnote h=3 pct f=swiss
        j=l ' L=Low' ' C=Close' ' H=High'
        j=r 'GPRCOLOR ';

```

Generate graph from the external file and specify text color. CTEXT= assigns a color to the text produced by the GPRINT procedure.

```

proc gprint fileref=out ctext=red;
run;

```

Example 2: Adjusting the Size of Characters

Procedure features:

GPRINT statement options:

FILEREF=

Other features:

FILENAME statement

GOPTIONS statement

PRINT procedure

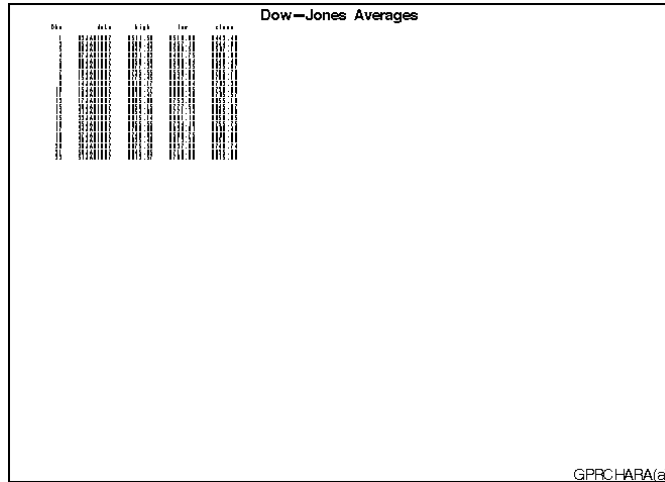
PRINTTO procedure

Data set: REFLIB.DOWHLC (see Example 1 on page 1153)

Sample library member: GPRCHARA

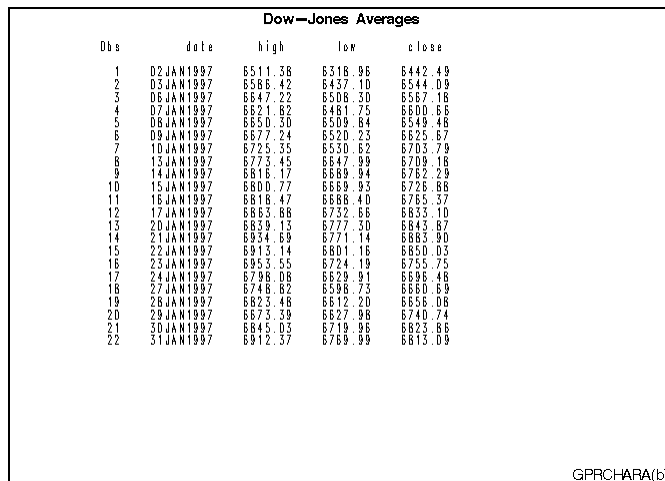
This example creates a graph from a text file and increases the size of the text. The first part of this example uses the PRINT procedure to create an external file that contains SAS output. The GPRINT procedure is used to import the text file into a graph. Because the LINESIZE= option (columns) is set to 76 and the PAGESIZE= option (rows) is set to 24, the output is small and occupies only a portion of the page, as shown in Figure 38.4 on page 1157:

Figure 38.4 GPRINT Procedure Output with No Adjustments (GPRCHARA(a))



In the second part of this example, the number of columns and rows in the graphics output area is reduced with the HPOS= and VPOS= graphic options. Thus, the size of the characters in the graph increase, as shown in Figure 38.5 on page 1157:

Figure 38.5 GPRINT Procedure Output with Adjusted Sizing (GPRCHARA(b))



Assign the libref and set the graphics environment. FTEXT= in the GOPTIONS statement specifies the default hardware font. (This is the default setting.)

```
libname reflib 'SAS-data-library';
goptions reset=global border cback=white
         colors=(black blue green red)
         ftitle=swissb ftext=none
         hsize=7in vsize=5in
         hpos=142 vpos=68;
```

Assign the fileref DOW to the external file. The fileref DOW is associated with the external file where the output from PROC PRINT is stored.

```
filename dow 'external-file';
```

Suppress the date line and page numbers. Set the line and page size.

```
options nodate nonumber linesize=76 pagesize=24;
```

Specify the destination for all subsequent procedure output. The PRINTTO procedure directs the SAS output to the external file that the GPRINT procedure subsequently uses as input. PRINT= directs all printed procedure output to the file referenced by the fileref DOW. NEW causes the output file to be replaced each time the program is run.

```
proc printto print=dow new;
run;
```

Send the output to the destination file. The PRINT procedure generates the text and sends it to the external file specified by PROC PRINTTO.

```
proc print data=reflib.dowhlc;
run;
```

Reset destination for printed output to the default. The destination for printed output is reset to the default by resubmitting PROC PRINTTO with no options.

```
proc printto;
run;
```

Define title and footnote.

```
title 'Dow-Jones Averages';
footnote h=3 pct f=swiss j=r 'GPRCHARA(a) ';
```

Generate graph from the external file. FILEREF= specifies the external file that is used as input. NOCC is omitted because the input text file contains carriage-control characters.

```
proc gprint fileref=dow;
run;
```

Reduce HPOS= and VPOS= to increase cell size.

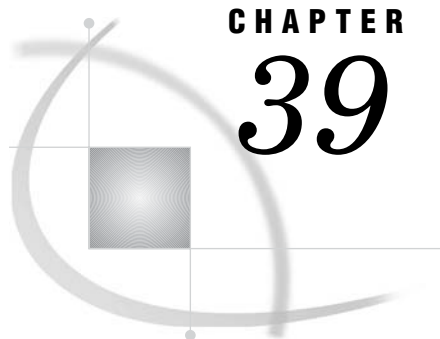
```
goptions hpos=75 vpos=30;
```

Define the footnote.

```
footnote h=3 pct f=swiss j=r 'GPRCHARA(b) ';
```

Generate adjusted graph.

```
proc gprint fileref=dow;  
run;
```

CHAPTER 39

The GPROJECT Procedure

<i>Overview</i>	1161
<i>Concepts</i>	1163
<i>About the Input Map Data Set</i>	1163
<i>Input Map Data Sets that Contain Only Unprojected Values</i>	1163
<i>Input Map Data Sets that Contain Both Projected and Unprojected Values</i>	1164
<i>About Coordinate Values</i>	1164
<i>About Types of Map Projections</i>	1165
<i>Albers' Equal-Area Projection</i>	1165
<i>Lambert's Conformal Projection</i>	1166
<i>Gnomonic Projection</i>	1167
<i>Procedure Syntax</i>	1167
<i>PROC GPROJECT Statement</i>	1168
<i>ID Statement</i>	1171
<i>Using the GPROJECT Procedure</i>	1172
<i>Selecting Projections</i>	1172
<i>Controlling Projection Criteria</i>	1172
<i>Clipping Map Data Sets</i>	1173
<i>Examples</i>	1173
<i>Example 1: Using Default Projection Specifications</i>	1174
<i>Example 2: Emphasizing Map Areas</i>	1177
<i>Example 3: Clipping an Area from the Map</i>	1178
<i>Example 4: Projecting an Annotate Data Set</i>	1180
<i>References</i>	1182

Overview

The GPROJECT procedure processes traditional map data sets by converting spherical coordinates (longitude and latitude) into Cartesian coordinates for use by the GMAP procedure. The process of converting coordinates from spherical to Cartesian is called *projecting*. In many of the traditional map data sets available with SAS/GRAPH software, the observation values are stored as longitude and latitude coordinates on a sphere (which means the map is unprojected). When these observations are plotted by the GMAP procedure, which is designed to plot points on a two-dimensional plane, the resulting map is often reversed and elongated as a result of forcing the curved map surface onto a flat plane.

The GPROJECT procedure enables you to use one of several map projection techniques to project the coordinates in a traditional map data set into a two-dimensional plane while attempting to minimize the distortion of area, distance, direction, and shape properties of the original sphere. (The earth is not precisely spherical and the GPROJECT procedure does not attempt to correct this small

distortion.) The output map data set that is produced by the procedure contains Cartesian coordinates that can be displayed correctly using the GMAP procedure.

The GPROJECT procedure also can create a rectangular subset of the input map data set by excluding all points with longitude and latitude values that fall outside of a specified range. This provides a handy way to reduce the size of the map data set if you need only a portion of a larger map.

The GPROJECT procedure does not produce any graphics output. Instead, it produces an output map data set, which typically becomes the input map data set for the GMAP procedure (see Chapter 35, “The GMAP Procedure,” on page 995).

Figure 39.1 on page 1162 and Figure 39.2 on page 1163 illustrate the effect of using GPROJECT defaults (Albers projection with standard parallels that are calculated by the procedure) to project a typical map data set with coordinates that are stored as longitude and latitude.

The program for the following maps can be seen in Example 1 on page 1174.

Figure 39.1 Map before Projection (GPJDEFLT(a))

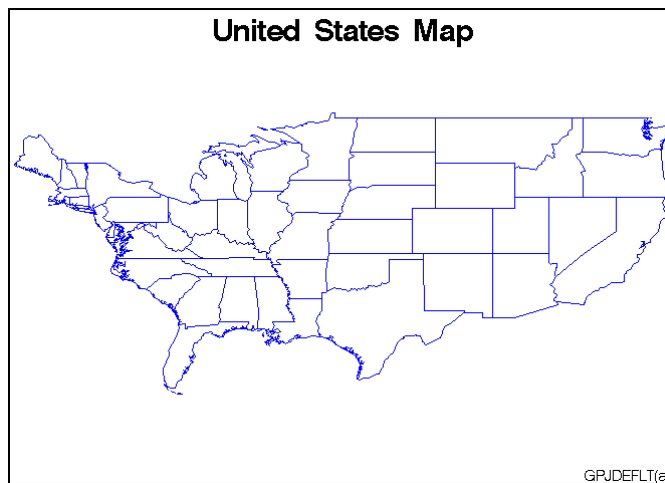
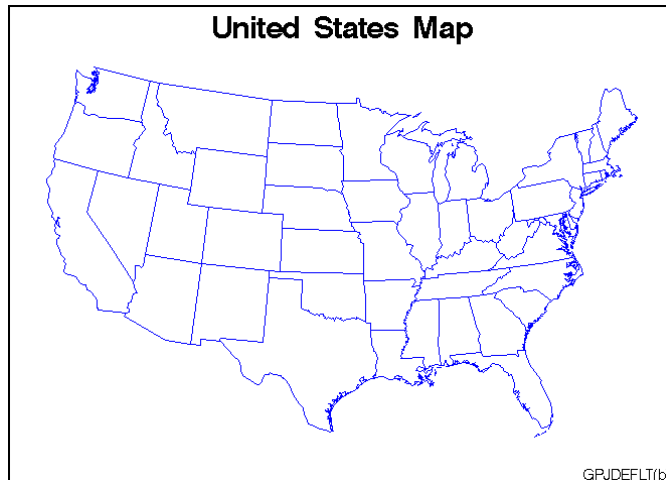


Figure 39.2 Map after Projection (GPJDEFLT(b))

Concepts

About the Input Map Data Set

The input map data set must be in traditional map data set format (see “About Traditional Data Sets” on page 999), and it must contain these variables:

- a numeric variable named X that contains the longitude coordinates of the map boundary points.
- a numeric variable named Y that contains the latitude coordinates of the map boundary points.
- one or more *identification variables* that uniquely identify the unit areas in the map. These variables are listed in the ID statement.

The X and Y variables contain the values that are to be projected.

In addition, the input map data set also can contain

- a numeric variable named SEGMENT that distinguishes nonconterminous segments of the unit areas.
- a numeric variable named DENSITY that can be used to affect the output from PROC GPROJECT. See “Clipping Map Data Sets” on page 1173 for more information.

Other variables in the input map data set do not affect the GPROJECT procedure.

Input Map Data Sets that Contain Only Unprojected Values

Note: Projection is appropriate for map data sets in which the X and Y variable values represent longitude and latitude. Some of the map data sets that are supplied with SAS/GRAPH have already been projected; such data set should not be projected again. △

The following is a list of all of the Institute-supplied data sets that contain X and Y variables whose values are unprojected:

CANADA3
 CANADA4
 COUNTIES
 COUNTY
 STATES

See Example 1 on page 1174 for an illustration of this type of input map data set and the variables it contains.

Input Map Data Sets that Contain Both Projected and Unprojected Values

Most traditional map data sets contain both sets of variables (X, Y and LONG, LAT) for projected and unprojected maps. In these cases, the X and Y variables will produce a projected map so you do not need to use the GPROJECT procedure. However, you may want to use the LONG and LAT variables to reproject the map using a different projection type. To do this you must first rename the LONG and LAT variables. It is necessary to rename the LONG and LAT variables because the GPROJECT procedure looks for variables that are named X and Y by default. You can create a new map data set using the OUT= option, drop the current X and Y variables, and rename the LONG and LAT variables. Your new data set will then contain unprojected values in X and Y. The following statements illustrate how to do this:

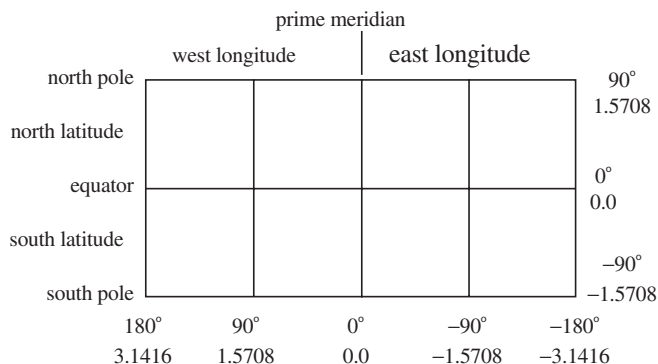
```
proc gproject data=map.austral
              (drop=x y rename=(long=x lat=y))
              out=newaust;
  id id;
run;
```

For additional information on the supplied SAS/GRAPH map data sets, see “About Map Data Sets” on page 999 and the METAMAPS data set in your maps data set directory.

About Coordinate Values

Figure 39.3 on page 1164 shows the standard coordinate system for map data sets with coordinates in longitude and latitude. For the longitude and latitude values (below and to the right of the figure, respectively) the upper value is expressed in degrees and the lower value is expressed in radians. A radian is approximately 57.3 degrees.

Figure 39.3 Longitude and Latitude Coordinates



By default, the GPROJECT procedure assumes that the units for the input coordinate values are radians and that values for the horizontal coordinate increase from east to west across the map. If your map coordinates are stored as degrees of arc, use the DEGREE option in the PROC GPROJECT statement. If the horizontal coordinate values in the map increase west-to-east rather than east-to-west, use the EASTLONG option in the PROC GPROJECT statement. See “Options” on page 1168 for details of DEGREE and EASTLONG.

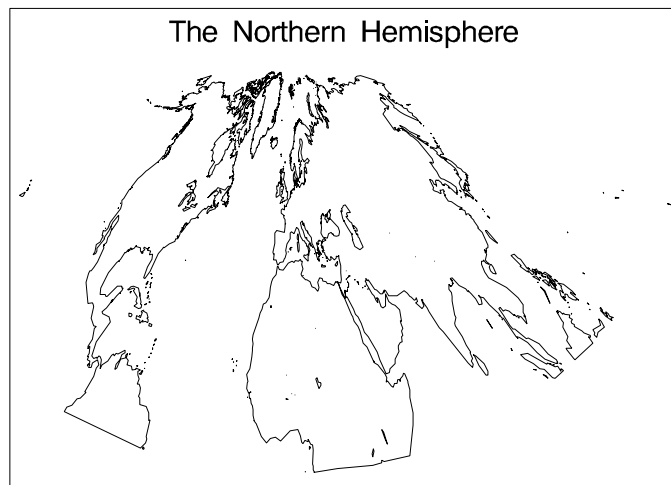
The unprojected map data sets that are provided with SAS/GRAPH can be projected if you use the default procedure characteristics: coordinate units in the data sets are radians, and horizontal values increase east-to-west.

About Types of Map Projections

The GPROJECT procedure performs three different types of projection: Albers’ equal-area projection with two standard parallels (the default method), Lambert’s conformal projection with two standard parallels, or the gnomonic projection (an azimuthal equidistant projection).

These sections describe the basic theory of each projection method. For comparison, Figure 39.4 on page 1165 shows an unprojected map of the northern hemisphere.

Figure 39.4 Unprojected Map



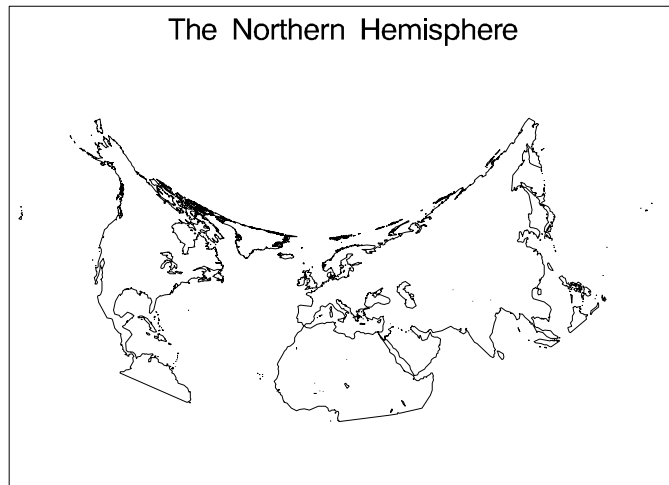
Albers’ Equal-Area Projection

The *Albers’ projection* is a conic projection from the surface of the sphere to a cone secant to the sphere, cutting it at two standard parallels of latitude. The axis of the cone coincides with an extension of the polar axis of the sphere. Each section of the resulting map bears a constant ratio to the area of the sphere. In general, distortion in shape tends to increase toward the poles in latitudes outside of the two standard parallels.

The Albers’ projection is well suited to portray areas of large and small east-to-west extent and produces satisfactory results in most cases. However, both standard parallels must lie on the same side of the equator, so this method may not be suitable for map data sets of large north-to-south extent that span the equator. For those map data sets, use the gnomonic projection method.

Figure 39.5 on page 1166 illustrates an Albers’ equal-area projection of the northern hemisphere.

Figure 39.5 Albers' Projection



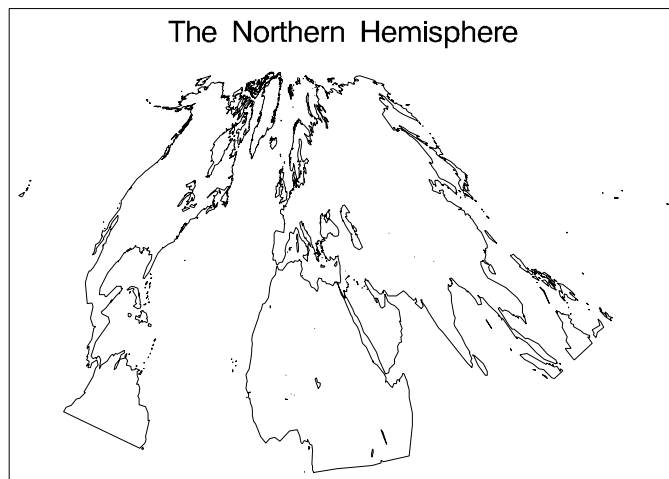
Lambert's Conformal Projection

The *Lambert's projection* is obtained from a secant cone in the same manner as Albers' projection. In the Lambert's projection, meridians of longitude are straight lines that radiate from the apex of the cone, while parallels of latitude are concentric circles. The Lambert's projection is somewhat better than the Albers' projection at representing the original shape of projected unit areas, while the Albers' projection is somewhat better at representing relative sizes of projected unit areas.

The Lambert's projection is ideal for navigational charts and maps of relatively small east-to-west extent. However, as in the Albers' projection, both standard parallels must lie on the same side of the equator, so this method may not be suitable for map data sets that span the equator. For those map data sets, use the gnomonic projection method.

Figure 39.6 on page 1166 illustrates a Lambert's conformal projection of the northern hemisphere.

Figure 39.6 Lambert's Projection



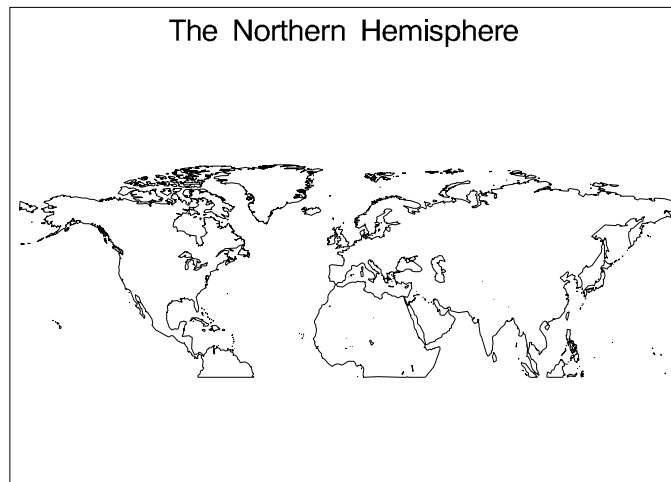
Gnomonic Projection

The *gnomonic projection* is a planar projection from the surface of the sphere directly onto an imaginary plane tangent to the sphere at the map projection pole. By default, the projection pole is placed at the center of the map data set that is to be projected, but you can specify the projection pole to be anywhere on the surface of the sphere. (See the POLELAT= and POLELONG option on page 1170.)

In the gnomonic projection, distortion increases as the distance from the map pole increases. Because of this distortion, the PROC GPROJECT procedure deletes all of the observations that lie more than 85 degrees from the map pole. The gnomonic projection is best suited for mapping areas of small east-to-west extent.

Figure 39.7 on page 1167 illustrates a gnomonic projection of the northern hemisphere.

Figure 39.7 Gnomonic Projection



Procedure Syntax

Requirements: Exactly one ID statement is required.

```
PROC GPROJECT <option(s)>;
  ID id-variable(s);
```

PROC GPROJECT Statement

Identifies the input and output map data sets. Optionally specifies the type of projection, and criteria for clipping and projection.

Requirements: An input map data set is required.

Syntax

PROC GPROJECT <option(s)>;

option(s) can be one or more options from any or all of the following categories:

□ data set options:

DATA=*input-map-data-set*

OUT=*output-map-data-set*

□ projection options:

PARADIV=*n*

PARALEL1=*latitude*

PARALEL2=*latitude*

POLELAT=*latitude*

POLELONG=*longitude*

PROJECT=ALBERS | GNOMON | LAMBERT | NONE

□ coordinate options:

ASIS | DUPOK

DEGREE

EASTLONG

□ clipping options:

LATMIN=*min-latitude*

LATMAX=*max-latitude*

LONGMIN=*min-longitude*

LONGMAX=*max-longitude*

Options

ASIS

DUPOK

specify that observations for which the projected values for the X and Y variables are identical to those in the previous observation should be retained. By default, successive identical observations are deleted.

DATA=*input-map-data-set*

identifies the map data set to be processed. By default, the procedure uses the most recently created SAS data set.

See also: “About the Input Map Data Set” on page 1163 and “SAS Data Sets” on page 29.

Featured in: Example 4 on page 1180.

DEGREE**DEG**

specifies that the units for the longitude (X variable) and latitude (Y variable) coordinates are degrees of arc. By default, coordinate units are considered to be radians.

EASTLONG**EAST**

specifies that the longitude (X variable) values in the input map data set increase to the east. By default, longitude values increase to the west.

LATMAX=*max-latitude*

specify the maximum latitude that will be included in the projection. Any unit areas that cross the selected latitude are clipped and closed along the specified parallels. The LATMAX= and LATMIN= options do not have to be paired; you can specify a maximum latitude without specifying a minimum.

When PROJECT=ALBERS, LAMBERT, or GNOMON, PROC GPROJECT treats the value of *max-latitude* as degrees. When PROJECT=NONE, the procedure treats the value as a Cartesian coordinate.

Featured in: Example 3 on page 1178.

LATMIN=*min-latitude*

specify the minimum latitude that will be included in the projection. Any unit areas that cross the selected latitude are clipped and closed along the specified parallels. The LATMAX= and LATMIN= options do not have to be paired; you can specify a minimum latitude without specifying a maximum.

When PROJECT=ALBERS, LAMBERT, or GNOMON, PROC GPROJECT treats the value of *min-latitude* as degrees. When PROJECT=NONE, the procedure treats the value as a Cartesian coordinate.

Featured in: Example 3 on page 1178.

LONGMAX=*max-longitude*

specify the maximum longitude to be included in the projection. Any unit areas that cross the selected longitude are clipped and closed along the specified meridians. The LATMAX= and LATMIN= options do not have to be paired; you can specify a maximum longitude without specifying a minimum.

When PROJECT=ALBERS, LAMBERT, or GNOMON, PROC GPROJECT treats the value of *max-longitude* as degrees. When PROJECT=NONE, the procedure treats the value as a Cartesian coordinate.

Featured in: Example 3 on page 1178.

LONGMIN=*min-longitude*

specify the minimum longitude to be included in the projection. Any unit areas that cross the selected longitude are clipped and closed along the specified meridians. The LATMAX= and LATMIN= options do not have to be paired; you can specify a minimum longitude without specifying a maximum.

When PROJECT=ALBERS, LAMBERT, or GNOMON, the GPROJECT procedure treats the value of *min-longitude* as degrees. When PROJECT=NONE, the procedure treats the value as a Cartesian coordinate.

Featured in: Example 3 on page 1178.

OUT=*output-map-data-set*

names the new map data set, which contains the coordinates of the new unit areas that are created by the GPROJECT procedure.

By default, the GPROJECT procedure names the new data set that uses the DATA*n* naming convention. That is, the procedure uses the name WORK.DATA*n*,

where n is the next unused number in sequence. Thus, the first automatically named data set is DATA1, the second is DATA2, and so on.

Featured in: Example 4 on page 1180.

PARADIV= n

specifies the divisor that computes the values used for standard parallels for the Albers' or Lambert's projections when explicit values are not provided. By default PARADIV=4, which causes standard parallels to be set at 1/4 and 3/4 of the range of latitude values in the input map data set.

See also: PARALEL1= and PARALEL2= option

PARALEL1=*latitude*

PARALEL2=*latitude*

specify values for the standard parallels that are used in the Albers' or Lambert's projection. *Latitude* must be in degrees. Positive values indicate north of the equator, and negative values indicate south of the equator. These options are ignored for the gnomonic projection.

By default, the GPROJECT procedure calculates values for the standard parallels. The defaults are chosen to minimize the distortion inherent in the projection process. The algorithm used is

$$\text{PARALEL1} = \text{minlat} + R / P_D$$

$$\text{PARALEL2} = \text{maxlat} - R / P_D$$

where:

R

is the range of latitude values in the input map data set.

P_D

is the PARADIV= value (see the discussion of the PARADIV= option).

minlat

is the minimum latitude value in the input map data set.

maxlat

is the maximum latitude value in the input map data set.

If you do not use PARALEL1= or PARALEL2=, or you omit either option, the GPROJECT procedure uses the calculated value for the missing parameter.

The standard parallels, whether explicitly specified or supplied by the procedure, must lie on the same side of the equator. If they do not, PROC GPROJECT prints an error message and stops (the procedure may calculate standard parallels that lie on opposite sides of the equator). When projecting a map data set that contains unit areas that cross the equator, you may have to explicitly specify standard parallels that both lie on the same side of the equator. If this causes excessive distortion of the map, you may be able to use the gnomonic projection instead of the Albers' or Lambert's projection because the gnomonic technique has no such limitations at the equator.

POLELAT=*latitude*

POLELONG=*longitude*

specify a projection pole to use for the gnomonic projection. The projection pole is the point at which the surface of the sphere touches the surface of the imaginary plane onto which the map is projected. POLELAT= specifies the latitude of the projection point.

Units for *latitude* are degrees; positive values indicate north of the equator, and negative values indicate south of the equator. POLELONG= gives the longitude for the projection point. Units for *longitude* are degrees; positive values indicate west of the prime meridian, and negative values indicate east of the prime meridian (unless EASTLONG also has been used in the PROC GPROJECT statement).

If you do not use POLELAT= or POLELONG=, or you omit either option, PROC GPROJECT uses values for the position of the center of the unit areas that are defined by the DATA= data set for the missing parameter.

Note: The map that is defined by the input map data set should not contain points more than 85 degrees (1.48353 radians) from the projection pole; all points that exceed this value are deleted from the output map data set. △

Featured in: Example 2 on page 1177.

PROJECT=ALBERS | LAMBERT | GNOMON | NONE

specifies the projection method to apply to the map data set. Values for PROJECT= are as follows:

ALBERS

specifies Albers' equal-area projection with two standard parallels.

LAMBERT

specifies Lambert's conformal projection with two standard parallels.

GNOMON

specifies the gnomonic projection, which is an azimuthal projection.

NONE

specifies that no projection should be performed. Use this option in conjunction with the LATMIN=, LATMAX=, LONGMIN=, and LONGMAX= options to perform clipping without projection (for example, on map data sets that have already been projected).

By default, PROJECT=ALBERS.

Note: There are several additional projections available. These projections are experimental and are not supported by SAS Institute Technical Support. They are: ADAMS, AITOFF, APIANUS, ARAGO, BEHRMANN, BRAUN, CYLINDRI, ECKERT1, ECKERT3, ECKERT5, EQUIRECT or MARINUS, GALL, KVRISKY7, MILLER1, MILLER2, ORTHO, PARABOLI, PETERS, PUTNINS4, ROBINSON, STEREO, WINKEL2.

You must specify the EASTLONG option to use any of these experimental projections. △

See also: "About Types of Map Projections" on page 1165.

Featured in: Example 2 on page 1177.

ID Statement

Identifies the variable or variables that define the hierarchy of the current unit areas in the input map data set.

Requirements: At least one *id-variable* is required.

Featured in: Example 1 on page 1174.

Syntax

ID *id-variable(s)*;

Required Arguments

id-variable(s)

specifies one or more variables in the input map data set that identify unit areas. *Id-variable* can be either numeric or character.

Each group of observations with a different ID variable value is evaluated as a separate unit area.

Using the GPROJECT Procedure

The GPROJECT procedure uses a default projection method and default projection criteria to project your map data set. If you do not want to use these defaults, you can use PROC GPROJECT statement options to

- select the map projection method
- specify the map projection criteria
- create a rectangular subset of the input map data set.

The following sections describe how you can use PROC GPROJECT statement options to select your own projection method and projection criteria.

Selecting Projections

Except when projecting map data sets that cover large areas, all three types of projections (Albers', Lambert's, and gnomonic) produce relatively similar results when you use default projection criteria, so you usually do not need to be concerned about which projection method to use when you produce maps solely for graphics output.

However, the default projection criteria may be unsuitable in some circumstances. In particular, the default specifications fail when the map that is being projected extends on both sides of the equator. On other occasions, you may want to select a projection method to achieve a particular effect.

For the Albers' and Lambert's projections, the two standard parallels must both lie on the same side of the equator. PROC GPROJECT stops with an error message if this condition is not met, regardless of whether you explicitly specify parallel values or let the procedure calculate default values. See the descriptions of the PARALEL1= and PARALEL2= option on page 1170 for more information on how to specify the two standard parallels.

Controlling Projection Criteria

For both the Albers' and Lambert's projections, PROC GPROJECT calculates appropriate standard parallels. You can override either or both of these selections if you explicitly specify values for the PARALEL1= or PARALEL2= option. You can influence the selection of default parallels if you use the PARADIV= option. See "Options" on page 1168 for more information on these options.

For the gnomonic projection, PROC GPROJECT determines the longitude and latitude of the approximate center of the input map data set area. You can override either or both of these selections if you explicitly specify values for the POLELAT= or POLELONG= option. See "Options" on page 1168 for more information.

The clipping options, discussed in "Clipping Map Data Sets" on page 1173, can also influence the calculations of the default standard parallels by changing the minimum and maximum coordinate values.

Clipping Map Data Sets

The GPROJECT procedure can create rectangular subsets of the input map data set. This capability provides a way to extract a portion of a larger map if you do not need all the original unit areas for your graph. The procedure enables you to clip unit area boundaries at specified parallels of latitude or meridians of longitude or both. Unit areas that fall completely outside of the specified clipping limits are excluded from the output map data set. Unit areas bisected by the clipping limits are closed along the clipping parallels and meridians, and all points outside of the clipping limits are excluded.

If the input map data set contains the DENSITY variable, any new vertex points and corners that are created by PROC GPROJECT are assigned a DENSITY value of 0 in the output map data set. This enables you to use a subset of the clipped map without using PROC GREduce to assign new DENSITY values. (See Chapter 41, “The GREduce Procedure,” on page 1213 for information on how to reduce the number of points that you need to draw a map.)

You can specify the minimum latitude to be retained in the output map data set with LATMIN= and the maximum latitude with LATMAX=. Minimum and maximum longitude values are specified with LONGMIN= and LONGMAX=, respectively. See “Options” on page 1168 for more details on these options.

This is how the PROC GPROJECT interprets the clipping longitude and latitude values:

- If you specify PROJECT=NONE in the PROC GPROJECT statement, the procedure assumes that the input map data set is already projected and the clipping longitude and latitude values are Cartesian coordinates. In this case, LATMAX= and LATMIN= specify the top and bottom edges, respectively, of the area that you want to extract, and LONGMAX= and LONGMIN= specify right and left edges, respectively.

You must be familiar with the range of values in the X and Y variables in order to select appropriate clipping limits. Use the MEANS or SUMMARY procedure in base SAS to determine the range of values in X and Y. See the *Base SAS Procedures Guide* for more information.

- If PROJECT=ALBERS, LAMBERT, or GNOMON, the clipping values are treated as degrees.

Depending on the size and position of the clipped area and the type of projection that is performed, the resulting map may not be exactly rectangular. PROC GPROJECT performs clipping before projection, so the clipped area may be distorted by the projection process.

To produce a clipped area with a rectangular shape, use PROC GPROJECT in two steps:

- 1 Project the map using the appropriate projection method and projection criteria.
- 2 Project the map using PROJECT=NONE, and use LATMIN=, LATMAX=, LONGMIN=, and LONGMAX= to clip the map.

See Example 3 on page 1178, for an example of clipping an area from an unprojected map data set.

Examples

The following examples illustrate major features of the GPROJECT procedure. Because these examples use map data sets that are supplied with SAS/GRAPH, you may need to replace *SAS-data-library* in the LIBNAME statement with the actual

location of the SAS data library that contains the Institute-supplied map data sets on your system. Contact your SAS Software Consultant for the location of the map data sets at your site. If your site automatically assigns the libref MAPS to the SAS data library that contains the Institute-supplied map data sets, delete the LIBNAME statement in these examples.

Example 1: Using Default Projection Specifications

Procedure features:

ID statement

Other features:

LIBNAME statement

Sample library member: GPJDEFLT

This example demonstrates the effect of using PROC GPROJECT on an unprojected map data set without specifying any options. Because PROJECT= is not used in the PROC GPROJECT statement, the Albers' equal-area projection method is used by default. PROC GPROJECT supplies defaults for the standard parallels that minimize distortion of the projected map areas.

Figure 39.8 Map before Projection (GPJDEFLT(a))

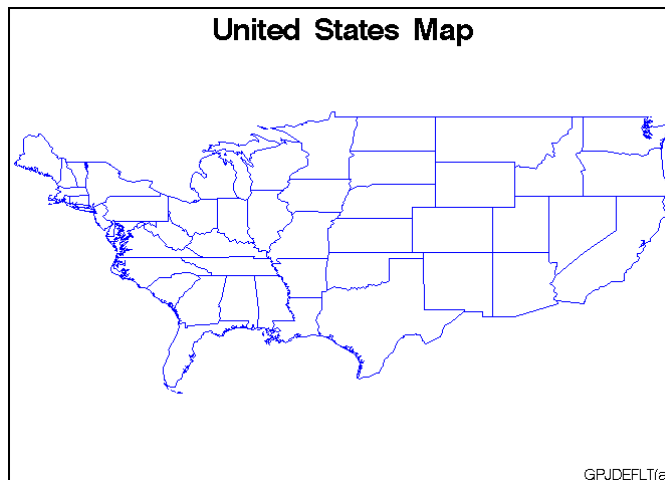


Figure 39.8 on page 1174 illustrates the output produced the US48 map data set, which contains only unprojected values, X and Y. Output 39.1 shows the variables in the data set.

Output 39.1 The US48 Data Set

US48 Data Set					
OBS	STATE	SEGMENT	DENSITY	X	Y
1	1	1	3	1.48221	0.56286
2	1	1	3	1.48226	0.56234
3	1	1	3	1.48304	0.56231
.					
.					
.					

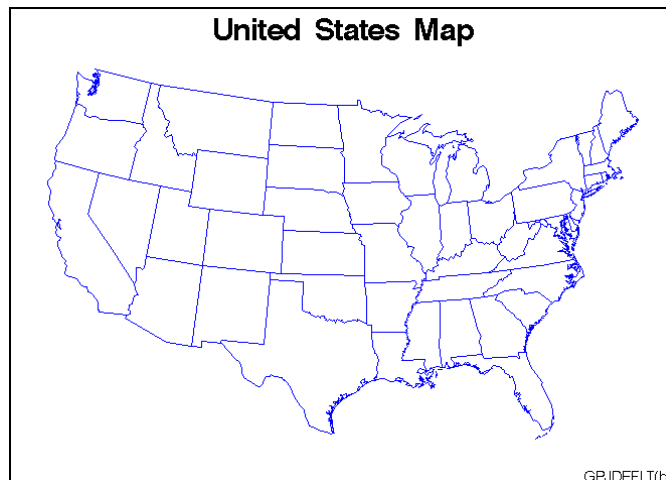
The GPROJECT procedure is used with the US48 map data set as input to create the projected map data set, US48PROJ. The values for X and Y in this new data set are projected (cartesian). Output 39.2 shows the variables in the data set.

Output 39.2 The US48PROJ Data Set

US48PROJ Data Set					
OBS	X	Y	DENSITY	STATE	SEGMENT
1	0.16068	-0.073470	3	1	1
2	0.16069	-0.073993	3	1	1
3	0.16004	-0.074097	3	1	1
.					
.					
.					

The new projected map data set, US48PROJ, is used to create the projected map, Figure 39.9 on page 1175.

Figure 39.9 Map after Projection (GPJDFLT(b))



Assign the libref and set the graphics environment.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ftext=swiss htitle=6 htext=3;
```

Create reduced continental U.S. map data set and remove Alaska, Hawaii, and Puerto Rico.

```
data us48;
  set maps.states;
  if state ne 2 and state ne 15 and state ne 72;
  if density<4;
run;
```

Define title and footnote for unprojected map.

```
title 'United States Map';
footnote j=r 'GPJDEFLT(a) ';
```

Define pattern characteristics.

```
pattern value=mempty repeat=50 color=blue;
```

Show unprojected map.

```
proc gmap map=us48 data=us48 all;
  id state;
  choro state / nolegend;
run;
```

Project map data set using all default criteria. The ID statement identifies the variable in the input map data set that defines unit areas.

```
proc gproject data=us48
              out=us48proj;
  id state;
run;
```

Define footnote for projected map.

```
footnote j=r 'GPJDEFLT(b) ';
```

Show projected map.

```

proc gmap map=us48proj
      data=us48proj all;
      id state;
      choro state / nolegend;
run;
quit;

```

Example 2: Emphasizing Map Areas

Procedure features:

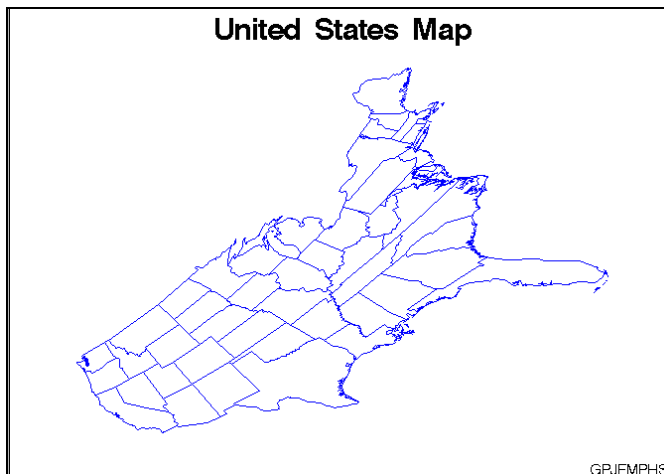
PROC GPROJECT options:

```

POLELAT=
POLELONG=
PROJECT=

```

Sample library member: GPJEMPHS



This example uses the gnomic projection method to create a map in which the east coast of the United States appears disproportionately large compared to the west coast.

Assign the libref and set the graphics environment.

```

libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
      colors=(black blue green red)
      ftext=swiss htitle=6 htext=3;

```

Create reduced continental U.S. map data set and remove Alaska, Hawaii, and Puerto Rico.

```

data us48;
  set maps.states;

```

```

    if state ne 2 and state ne 15 and state ne 72;
    if density<4;
run;

```

Project map onto a plane centered in the Pacific. PROJECT= specifies the projection method for the map data set. POLELONG= and POLELAT= specify a projection pole for the gnomonic projection. In this example, the pole is positioned in the Pacific Ocean.

```

proc gproject data=us48
    out=skew
    project=gnomon
    polelong=160
    polelat=45;
    id state;
run;

```

Define title and footnote for the map.

```

title 'United States Map';
footnote j=r 'GPJEMPHS ';

```

Define pattern characteristics.

```

pattern value=mempty repeat=49 color=blue;

```

Show the projected map.

```

proc gmap map=skew data=skew all;
    id state;
    choro state / nolegend;
run;
quit;

```

Example 3: Clipping an Area from the Map

Procedure features:

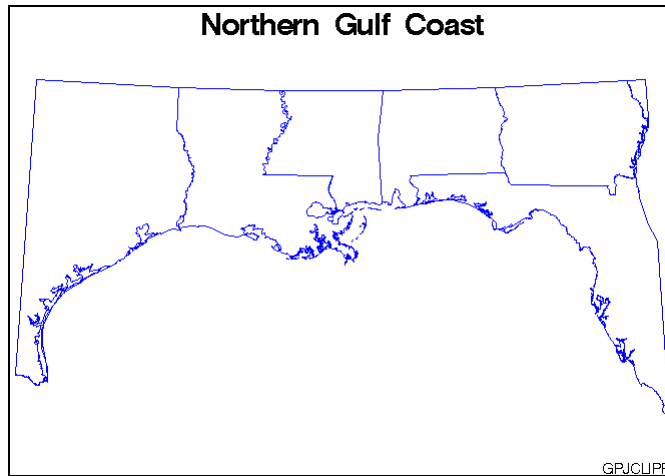
PROC GPROJECT options:

```

LONGMAX=
LONGMIN=
LATMAX=
LATMIN=

```

Sample library member: GPJCLIPP



This example uses the clipping capabilities of PROC GPROJECT to create a map of the states in the United States that border the Gulf of Mexico. Because the PROJECT= option is not used in the GPROJECT procedure, the Albers' equal-area projection method is used by default.

Assign the librefs and set the graphics environment.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ftext=swiss htitle=6 htext=3;
```

Clip and project rectangular subset of the map. LONGMIN= and LONGMAX= specify the minimum and maximum longitudes to be included in the map projection. LATMIN= and LATMAX= specify the minimum and maximum latitudes to be included in the map projection.

```
proc gproject data=maps.states
             out=gulf
             longmin=81
             longmax=98
             latmin=25
             latmax=33;
  where density<5;
  id state;
run;
```

Define title and footnote for the map.

```
title 'Northern Gulf Coast';
footnote j=r 'GPJCLIPP ';
```

Define pattern characteristics.

```
pattern value=empty repeat=7 color=blue;
```

Show the clipped map.

```
proc gmap map=gulf data=gulf all;
  id state;
  choro state / nolegend;
run;
quit;
```

Example 4: Projecting an Annotate Data Set*Procedure features:*

PROC GPROJECT options:

DATA=

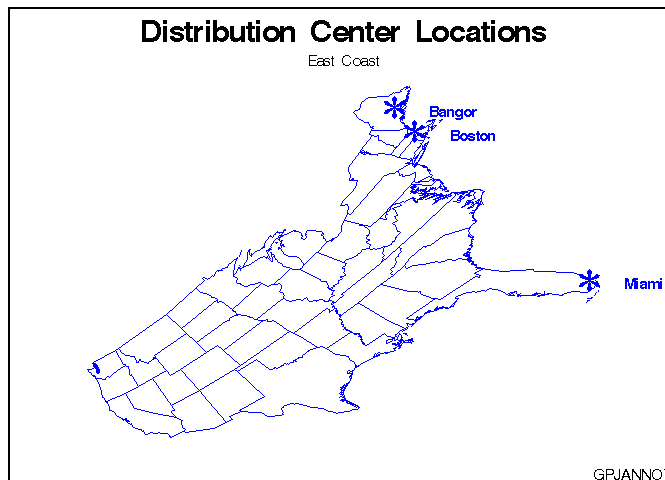
OUT=

ID statement

Other features:

CHORO statement

Annotate data set

Sample library member: GPJANNOT

This example illustrates how to project an Annotate data set for use with a map data set. It labels the locations of Miami, Boston, and Bangor on the map shown in the second example. Because the X and Y variables in the USCITY data set already have been projected to match the US data set, they cannot be used with the map that is produced by the second example. To properly label the projected map, the example uses the same projection method for the city coordinates as the method that is used for the map coordinates. This example illustrates how to use the same projection method for both data sets.

Assign the librefs and set the graphics environment.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(black blue green red)
          ftext=swiss htitle=6 htext=3;
```

Create reduced continental U.S. map data set and remove Alaska, Hawaii, and Puerto Rico.

```
data us48;
  set maps.states;
  if state ne 2 and state ne 15 and state ne 72;
  if density<4;
run;
```

Create Annotate data set CITIES from the MAPS.USCITY data set. The unprojected LONG and LAT variable values are converted to radians and substituted for the projected X and Y variable values. LONG and LAT are converted by multiplying them by the arccosine of -1 and dividing that amount by 180. The cities are each assigned a value for the NEWST variable, sequentially beginning at 100.

```
data cities(drop=state rename=(newst=state));
  set maps.uscity(keep=lat long city state);
  length function style color $ 8
           position $ 1 text $ 20;
  retain function 'label' xsys ysys '2'
           hsys '1' when 'b' newst 100;
  if state=12 and city='Miami' or
     state=25 and city='Boston' or
     state=23 and city='Bangor';
  newst+1; color='blue'; size=10; text='T';
  position='5';
  style='marker'; x=long*acos(-1)/180;
  y=lat*acos(-1)/180; output;
  newst+1; color='blue'; size=4;
  text='      '||city;
  position='6'; style='swissb'; output;
run;
```

Create data set ALL by combining data set US48 and data set CITIES.

```
data all;
  set us48 cities;
run;
```

Project the ALL data set. DATA= specifies the data set to be projected. OUT= specifies the name of the new projected data set that is created. The ID statement identifies the variable in the input map data set that defines map areas.

```
proc gproject data=all
              out=allp
```

```

        project=gnomon
        polelong=160
        polelat=45;
    id state;
run;

```

Separate the projected data set into the CITIESP Annotate data set and the US48P map data set.

```

data citiesp us48p;
    set allp;
    if state>100 then output citiesp;
    else output us48p;
run;

```

Define title and footnote for the map.

```

title1 'Distribution Center Locations';
title2 'East Coast';
footnote j=r 'GPJANNOT ';

```

Define pattern characteristics.

```

pattern value=mempty repeat=49 color=blue;

```

Show the annotated map. The CHORO statement displays the projected map and annotates it using the projected Annotate data set.

```

proc gmap data=us48p map=us48p all;
    id state;
    choro state
        / nolegend
        annotate=citiesp;
run;
quit;

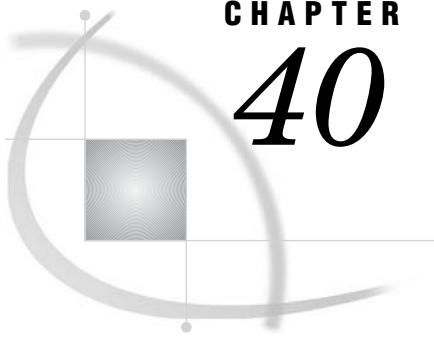
```

References

Pearson, F., II (1977), "Map Projection Equations," Report Number TR-3624, Naval Surface Weapons Center, Dahlgren Laboratory, March, 1977.

Richardus, P. and Adler, R.K. (1972), *Map Projections*, Amsterdam: North-Holland Publishing Company; New York: American Elsevier Publishing Company.

Robinson, A.H. (1978), *Elements of Cartography*, New York: John Wiley & Sons, Inc.



CHAPTER

40

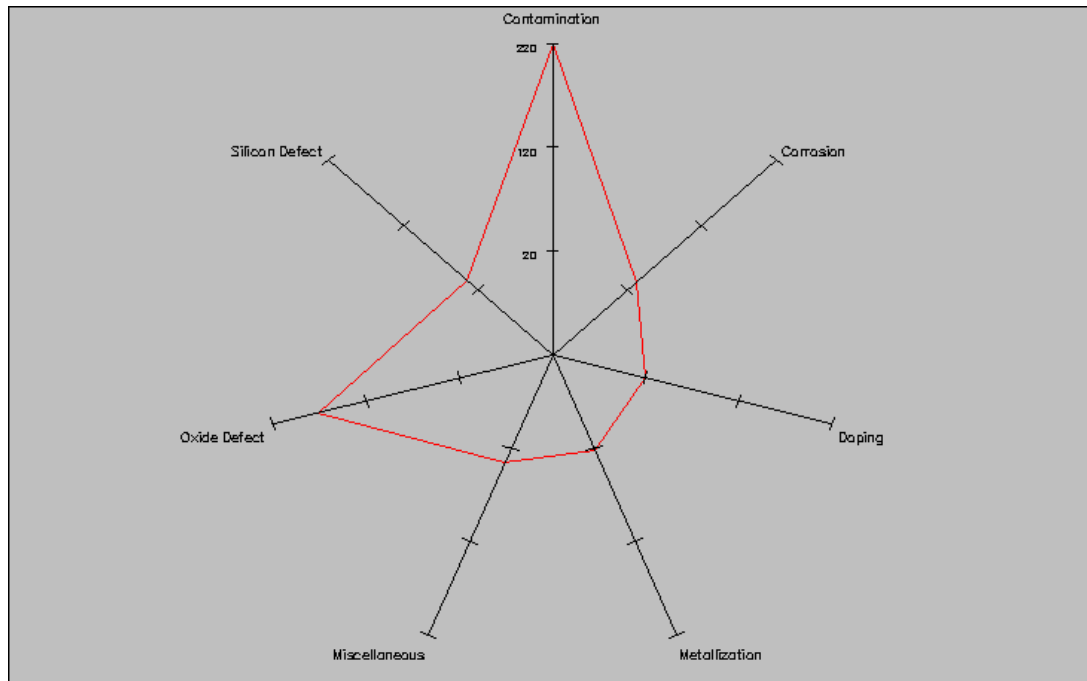
The GRADAR Procedure

<i>Overview</i>	1183
<i>Procedure Syntax</i>	1184
<i>PROC GRADAR Statement</i>	1184
<i>CHART Statement</i>	1185
<i>Examples</i>	1196
<i>Example 1: Generating the Data Set for the GRADAR Examples</i>	1196
<i>Example 2: Producing a Basic Radar Chart</i>	1198
<i>Example 3: Overlaying Radar Charts</i>	1199
<i>Example 4: Tiling Radar Charts</i>	1201
<i>Example 5: Using Multiple Classification Variables in Radar Charts</i>	1202
<i>Example 6: Filling the Stars in Radar Charts</i>	1204
<i>Example 7: Using Images in Radar Charts</i>	1205
<i>Example 8: Changing the Star Type in Radar Charts</i>	1207
<i>Example 9: Using Color and Line Styles in Radar Charts</i>	1208
<i>Example 10: Specifying the Mode for a Radar Chart</i>	1209
<i>Example 11: Assigning Axis Definitions to Axis Spokes</i>	1210

Overview

The GRADAR procedure creates radar (or star) charts that show the relative frequency of data measures in quality control or market research problems. On a radar chart, the chart statistics are displayed along spokes that radiate from the center of the chart (hence the term “star” charts). The charts are often stacked on top of one another with reference circles, thus giving them the look of a radar screen. By default, the chart vertices—the points where the statistical values intersect the spokes—are based on the frequencies associated with the levels of a single numeric variable, but they can be scored with a weight variable. Non-integer values of the chart variable are truncated to integers. The measures can be displayed in decreasing order, the order in which they appear in the input data, increasing order of internal values, or lexicographic order of variable names.

Note: The Java applet does not support client-side rendering of GRADAR charts. Δ



Procedure Syntax

Requirements: At least one CHART statement is required.

Global Statements: AXIS“AXIS Statement” on page 124, FOOTNOTE“TITLE, FOOTNOTE, and NOTE Statements” on page 210, GOPTIONS“GOPTIONS Statement” on page 146, TITLE“TITLE, FOOTNOTE, and NOTE Statements” on page 210

Reminder: The procedure can include the BY, FORMAT, LABEL, and WHERE statements as well as SAS/GRAPH NOTE statement.

Supports: Output Delivery System (ODS)

```
PROC GRADAR <DATA=input-data-set>
  <GOUT=<libref.>output-catalog>
  <ANNOTATE=Annotate-data-set>;
```

```
CHART chart-variable(s) </ option(s)>;
```

PROC GRADAR Statement

Identifies the data set that contains the plot variables. Optionally specifies an output catalog.

Requirements: An input data set is required.

Syntax

```
PROC GRADAR <DATA=input-data-set>
```

<GOUT=<libref.>output-catalog>
<ANNOTATE=Annotate-data-set;>

Options

PROC GRADAR statement options affect all graphs produced by the procedure.

ANNOTATE=Annotate-data-set

ANNO=Annotate-data-set

specifies a data set to annotate all graphs that are produced by the GRADAR procedure. To annotate individual graphs, use ANNOTATE= in the CHART statement.

See also: *SAS/GRAPH Reference, Volumes 1 and 2* for more information on the Annotate data set

DATA=input-data-set

specifies the SAS data set that contains the variable(s) to chart. By default, the procedure uses the most recently created SAS data set.

GOUT=<libref.>output-catalog

specifies the SAS catalog in which to save the graphics output produced by the GRADAR procedure.

CHART Statement

Creates the radar charts in which the length of the vertices along the spines represent the values of the chart statistic for the data categories.

Requirements: At least one chart variable is required.

Global statements: AXIS, TITLE, FOOTNOTE, and NOTE

Syntax

CHART *chart-variable(s)* </option(s)>;

option(s) can be one or more of the following:

ACROSS=*variable*

ANNOTATE=*Annotate-data-set*

CAXIS=*grid-color*

CFRAME=*background-color* | (*variable*)

CFRAMESIDE=*color*

CFRAMETOP=*color*

CSPOKES=*spoke-color*

CSTARCIRCLES=*color* | (*colors-list*)

CSTARFILL=*color* | (*colors-list*)

CSTARS=*color* | (*colors-list*)

CTEXT=*text-color*

CTILES=(*variable*) | *color*

DESCRIPTION=*'entry-description'*
 DOWN=*variable*
 FONT=*font*
 FREQ=*variable*
 FRAME | NOFRAME
 HEIGHT=*height*
 HTML=*variable*
 HTML_LEGEND=*variable*
 IFRAME=*fileref* | *'external-image-file'*
 IMAGESTYLE=TILE | FIT
 INBORDER
 INHEIGHT=*value*
 INTERTILE=*value*
 LAST=*'variable'*
 LSPOKE=*linetype*
 LSTARCIRCLES=(*linetypes*)
 LSTARS=(*linetypes*)
 MAXNVERT=*n*
 MAXVERT=*n*
 MISSING
 MODE=SHARE | PROTECT | RESERVE
 NAME=*'entry-name'*
 NCOLS=*n*
 NROWS=*n*
 NOFRAME
 NOZEROREF
 ORDERACROSS=FREQ | DATA | INTERNAL | FORMATTED | EXTERNAL
 OTHER=*'variable'*
 OVERLAY=*overlay-variable*
 SPIDER | SPIDERWEB
 SPKLABEL=CATEGORY | NONE
 STARAXIS = (AXIS<1...99><, . . . ,AXIS<1...99>>)
 STARCIRCLES=(*values*)
 STARFILL= lists of (SOLID | EMPTY) one for each star.
 STARINRADIUS=*value*
 STARLEGEND=CLOCK | CLOCK0 | NUMBER | DEGREES | NONE
 STARLEGENDLAB=*'legend-label'*
 STAROUTRADIUS=*value*
 STARSTART=*value*
 STARTYPE=CORONA | POLYGON | RADIAL | SPOKE | WEDGE
 SUMVAR=*summary-variable*
 TILELEGEND=*variable*
 TILELEGLABEL=*'label'*
 WAXIS=*n*
 WEIGHT=*numeric-variable*

WFRAME=*n*
 WSPOKES=*n*
 WSTARCIRCLES=(*line-widths*)
 WSTARS=*line-widths* | (*line-widths*)

Required Arguments

chart-variable(s)

specifies one or more variables that define the categories of data to be charted. The values of the chart variable determine the spokes in the corresponding radar chart. These values are the observations in the reference data set for the chart variable. You must have at least three observations in the data set as it takes three points to define a plane. Technically, you can create a GRADAR chart with only one or two observations, but a true chart will not display.

Options

Options in a CHART statement affect all graphs produced by that statement. You can specify as many options as you want and list them in any order.

ACROSS=*variable*

ACROSSVAR=*variable*

generates a radar chart for each value of the specified variable, and displays the charts from left-to-right across the graphics area. If used with the DOWN= option, the charts are drawn in left-to-right and top-to-bottom order. To limit the number of columns and/or rows that are displayed, use the NCOLS= and NROWS= options. Used with the ORDERACROSS= option.

Featured in: Example 4 on page 1201 and Example 5 on page 1202.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate charts produced by the CHART statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

CAXIS=*grid-color*

CAXES=*grid-color*

CA=*grid-color*

specifies a color for the chart frame. The specified color must be a valid SAS/GRAPH color name. If you omit the CAXIS= option, PROC GRADAR uses the first color in the colors list as the chart frame’s color.

Not supported by: ActiveX

CFRAME=*background-color* | (*variable*)

CFR=*background-color* | (*variable*)

fills the frame area with the specified color. You can specify a valid SAS/GRAPH color name, or a character variable of length eight whose value is the background color.

Featured in: Example 2 on page 1198 and Example 9 on page 1208.

CFRAMESIDE=*color* | (*variable*)

specifies the color for filling the frame area for the row labels displayed along the left side of a chart. The specified color must be a valid SAS/GRAPH color name, or a character variable of length eight whose value is the background color. If a label is associated with the classification variable, the specified color is also used to fill the frame area for this label. By default, these areas are not filled.

Featured in: Example 5 on page 1202

CFRAMETOP=*color* | (*variable*)

specifies the color for filling the frame area for the column labels that are displayed across the top of a chart. The specified color must be a valid SAS/GRAPH color name, or a character variable of length eight whose value is the color. If a label is associated with the classification variable, the specified color is also used to fill the frame area for this label. By default, these areas are not filled.

Featured in: Example 5 on page 1202

CSPOKES=*spoke-color* | (*variable*)

CSPOKE=*spoke-color*

specifies a color to use for the spokes in a chart. The specified color must be a valid SAS/GRAPH color name, or a character variable of length eight whose value is the color. By default, the spokes are colored by the first color in the colors list.

Featured in: Example 7 on page 1205

CSTARCIRCLES=*color* | (*colors-list*)

CSTARCIRCLE=*color* | (*colors-list*)

specifies a color or list of colors for the circles that are requested with the STARCIRCLES= option. All specified colors must be valid SAS/GRAPH color names, or a character variable of length eight whose value is the color. By default, the color specified with the CSTARS= option is used. If CSTARS= is not specified, the first color in the colors list is used.

Featured in: Example 3 on page 1199

CSTARFILL=*color* | (*colors-list*)

specifies a color or colors for filling the interior of stars that are produced for a radar chart. All specified colors must be valid SAS/GRAPH color names.

- If the OVERLAY= option is not used, each chart contains only one star. In that case, specify a single fill color for the star. If the ACROSS= and/or DOWN= options are used, the specified color is applied to each star in the tiled display.
- If the OVERLAY= option is used, the chart contains multiple stars. In that case, specify a list of colors in parentheses. Be sure that there are at least as many colors in the list as there are stars in the chart. If you do not specify enough colors for each star in the chart to have a separate color, the default color list is used to assign colors to additional stars. The color for the star positioned at subgroup n on the chart is the value of the color corresponding to the color at position n in the list of colors.

By default, the interior of the stars is empty. For empty stars, use the CSTARS= option to specify a color for the star outline.

If CSTARFILL= is specified and CSTARS= is not specified for the outline, then the outline is the same as CSTARFILL.

If CSTARFILL= is not specified and STARFILL=SOLID, then the star is filled with the color that is specified on CSTARS=.

If STARFILL= is not set or is set to EMPTY, then CSTARFILL= is ignored.

Featured in: Example 6 on page 1204

CSTARS=*color* | (*colors-list*)

CSTAR=*color* | (*colors-list*)

specifies a color or colors for the outlines of stars that are produced for a radar chart. All specified colors must be valid SAS/GRAPH color names.

- If the OVERLAY= option is not used, each chart contains only one star. In that case, specify a single color for the star. If the ACROSS= and/or DOWN= options are used, the specified color is applied to each star in the tiled display.

- If the OVERLAY= option is used, the chart contains multiple stars. In that case, specify a list of colors in parentheses. Be sure that there are at least as many colors in the list as there are stars in the chart. If you do not specify enough colors for each star in the chart to have a separate color, the default color list is used to assign colors to additional stars. The color for the star positioned at subgroup *n* on the chart is the value of the color corresponding to the color at position *n* in the list of colors.

By default, the colors in the devices are used, starting with the second color.

Featured in: Example 2 on page 1198 and Example 9 on page 1208.

CTEXT=*text-color*

specifies a color for all text on the chart. The specified color must be a valid SAS/GRAPH color name, or a character variable of length eight whose value is the color. For ActiveX devices, the default color is black. For other devices, if you omit CTEXT=, PROC GRADAR searches for a color specification in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the first color in the GOPTIONS colors list (the default).

CTILES=(*variable*) | color

CTILE=(*variable*) | color

specifies a character variable of length eight whose values are the fill colors for the tiles in a radar chart. By default, the tiles are not filled. The values of the specified variable must be identical for all observations with the same level of the classification variables. The same color can be used to fill more than one tile. Use the special value, EMPTY, to indicate that a tile is not to be filled.

Alternatively, CTILES= can specify a valid SAS/GRAPH color name to color all the tiles the same color.

The CTILES= option cannot be used in conjunction with the NOFRAME option or the CFRAME= option. You can use the TILELEGEND= option in conjunction with the CTILES= option to add an explanatory legend for the CTILES= colors at the bottom of the chart.

Not supported by: ActiveX

DESCRIPTION=*'entry-description'*

DES=*'entry-description'*

specifies the description of the catalog entry for the chart. The maximum length for *entry-description* is 256 characters long. The description does not appear on the chart. By default, the GRADAR procedure assigns a description of the form RADAR CHART OF *variable*, where *variable* is the name of the chart variable.

The *entry-description* can include the #BYLINE, #BYVAL, and #BYVAR substitution options, which work as they do when used on TITLE, FOOTNOTE, and NOTE statements. The 256-character limit applies before the substitution takes place for these options; thus, if in the SAS program the entry-description text exceeds 256 characters, it is truncated to 256 characters, and then the substitution is performed.

The descriptive text is shown in the "description" portion of each of the following:

- the Results window
- the catalog-entry properties that you can view from the Explorer window
- the Table of Contents that is generated when you use CONTENTS= on an ODS HTML statement (assuming the GRADAR output is generated while the contents page is open)
- the Description field of the PROC GREPLAY window.

DOWN=variable**DOWNVAR=variable**

generates a radar chart for each value of the specified variable, and displays the charts from top-to-bottom in the graphics area. If used with the ACROSS= option, the charts are drawn in left-to-right and top-to-bottom order. To limit the number of columns and/or rows that are displayed, use the NCOLS= and NROWS= options.

Featured in: Example 5 on page 1202.

FONT=font

specifies the font for all text strings in the radar chart. If you omit FONT=, the font that is specified by the FTEXT= graphics option is used.

Featured in: Example 7 on page 1205.

FREQ=variable

specifies a frequency variable whose value provides the counts (numbers of occurrences) of the values of the process variable. Specifying a FREQ= variable is equivalent to replicating the observations in the input data set. The FREQ= variable must be a numeric variable with non-negative integer values. If you specify more than one process variable on the CHART statement, the FREQ= variable values are used with each process variable. If you do not specify a FREQ= variable, each value of the process variable is counted exactly once.

Featured in: Example 2 on page 1198.

FRAME | NOFRAME

FRAME (the default) draws a frame around the procedure output area. By default, the frame color is the first color in the colors list. If you want to specify a different color for the frame, use the CFRAME= option for a filled frame, and CAXIS= for only the frame outline color.

NOFRAME suppresses the frame that is drawn around the chart by default. The NOFRAME option cannot be specified in conjunction with the CFRAME= or CTILES= options.

Not supported by: ActiveX

HEIGHT=height**HLABEL=height**

specifies the height in percent screen units of text for labels and legends. This option should be used only in conjunction with the FONT= option. The HEIGHT= option takes precedence over the HTEXT= option in a GOPTIONS statement.

HTML=variable

identifies the variable in the input data set whose values create links in the HTML file created by the ODS HTML statement. The maximum length for the value of this variable is 1024 characters.

HTML_LEGEND=variable

identifies the variable in the input data set whose values create links in the HTML file created by the ODS HTML statement. These links are associated with a legend value and point to the data or graph you wish to display when the user drills down on the value. The maximum length for the value of this variable is 1024 characters.

IFRAME=fileref | 'external-image-file'

specifies an image file to use on the chart's frame. *Fileref* must be a valid SAS fileref up to eight characters long and must have been previously assigned with a FILENAME statement. *External-image-file* must specify the complete file name of the image file you want to use. The format of *external-image-file* varies across operating environments. For more information, see "Placing a Backplane Image on Graphs with Frames" on page 115.

Featured in: Example 7 on page 1205.

Not supported by: ActiveX

IMAGESTYLE=TILE | FIT

specifies the way to display the image file that is specified on the IFRAME= option. TILE copies the image as many times as needed to fit the frame. FIT stretches the image so that a single copy fits within the frame.

Note: When used with IFRAME, the IMAGESTYLE option must be within the PROC statement. When used with IBACK, the IMAGESTYLE option goes on the GOPTIONS statement. △

Featured in: Example 7 on page 1205.

INBORDER

requests an inside border around the chart (this border is inside the chart's frame).

Not supported by: ActiveX

INHEIGHT=*value*

specifies the height in percent screen units of text used inside the frame of the chart, such as sample size legends and bar labels. This does not change the size of titles or footnotes.

Not supported by: ActiveX

INTERTILE=*value*

INTERCHART=*value*

specifies the distance in horizontal percent screen units between tiles (cells) in a chart and is used only with the ACROSS= and/or DOWN= options. By default, the tiles are contiguous (*value*=0).

Featured in: Example 5 on page 1202.

LAST=*'category'*

specifies that the spoke corresponding to the category is displayed to the left of the start angle. The *category* must be a formatted value of the process variable and must be enclosed in quotes.

Not supported by: ActiveX

LSPOKE=*linetype*

specifies a line type for the spokes in a radar chart. The default linetype is 1, which produces a solid line.

LSTARCIRCLES=(*linetypes*)

LSTARCIRCLE=(*linetypes*)

specifies one or more line types for the circles requested with the STARCIRCLES= option. The number of line types should match the number of circles requested as follows:

```
starcircles = (0.0 1.0 0.25 0.5)
lstarcircles = (1 1 2 2)
```

The line types are paired with the circles in the order specified. The default linetype is 1, which produces a solid line.

LSTARS=(*linetypes*)

LSTAR=(*linetypes*)

specifies the line types for the outlines of stars that are produced for a radar chart.

- If the OVERLAY= option is not used, each chart contains only one star. In that case, specify a single line type for the star.
- If the OVERLAY= option is used, the chart contains multiple stars. In that case, specify a list of line types in parentheses. Be sure that there are at least

as many line types in the list as there are stars in the chart. The line type for the star positioned at subgroup n on the chart is the value of the line type corresponding to the color at position n in the list of line types.

By default, the outlines rotate through different line types and colors. To specify line colors, use the CSTARTS= option.

Featured in: Example 9 on page 1208.

MAXNVERT= n

MAXVERT= n

specifies the maximum number of vertices, from 1 to 360, in the radar chart. Either spelling of the option is accepted. The MAXNVERT= or MAXVERT= options are for use with the OTHER= option.

MISSING

accepts a missing value as a valid midpoint for the chart variable. By default, observations with missing values are ignored. Missing values are always valid for the overlay variables.

Not supported by: ActiveX

MODE=SHARE | PROTECT | RESERVE

specifies the display mode for a radar chart. The following keywords are available:

SHARE	shares the drawing space between the text and the graph.
PROTECT	shares the drawing space but maintains a solid rectangle (using the background color) behind the text. This is useful when the text is illegible because of the iframe image or cframe color.
RESERVE	reduces the size of the text and graph in order to accommodate both. RESERVE is the default.

Featured in: Example 10 on page 1209.

NAME='entry-name'

specifies a string, up to 8 characters long. Default: RADAR. The name duplicates an existing entry name and SAS/GRAPH software adds a number to the duplicate name to create a unique name—for example, RADAR1.

NCOLS= n

NCOL= n

specifies the number of columns in a chart. You can use the NCOLS= option in conjunction with the NROWS= option. By default, NCOLS=1 and NROWS=2 if one classification variable is specified, and NCOLS=2 and NROWS=2 if two classification variables are specified. If used with the ACROSS= and DOWN= options, the default number of columns and rows are calculated by the number of classifications for the variables that are listed on ACROSS= and DOWN=. In that case, you can use NCOLS= and NROWS= to limit the number of columns and rows that are specified.

Featured in: Example 5 on page 1202.

Not supported by: ActiveX

NOFRAME

suppresses the frame that is otherwise drawn around the chart by default. The NOFRAME option cannot be specified in conjunction with the CFRAME= or CTILES= options

NOZEROREF

turns off the zero reference line when negative values are plotted. When a negative value is plotted, a dashed line indicates the zero position. You can not change the appearance of this zero reference line, but you can turn it off with the NOZEROREF option. The zero reference line does not appear if there are no negative values plotted.

NROWS=*n***NROW=*n***

specifies the number of rows in a chart. You can use the NROWS= option in conjunction with the NCOLS= option. See NCOLS= for details. By default, NROWS=1.

Featured in: Example 5 on page 1202.

Not supported by: ActiveX

ORDERACROSS=FREQ | DATA | INTERNAL | FORMATTED | EXTERNAL

Specifies the display order for the values of the ACROSS=variable.

Not supported by: ActiveX

OTHER='category'

specifies a new category that merges all categories not selected because of the MAXNVERT= or MAXVERT= option. The category should be specified as a formatted value of the process variable. The OTHER= option is applicable only if you also specify the MAXNVERT= or MAXVERT= option.

OVERLAY=overlay-variable**OVERLAYVAR=overlay-variable**

creates a comparative radar chart using the levels of the overlay variable. All charts are displayed in the same set of spokes. This option cannot be used with the ACROSS= or DOWN= options.

Featured in: Example 3 on page 1199.

SPIDERWEB | SPIDER

displays lines connecting the points where tick marks would be instead of displaying the tick marks, using the same number of points for all axes as for the first axis. The default number of web lines is three.

If there is an axis statement along with the SPIDERWEB option, then the web gets its values such as number, thickness, and color from the MAJOR=() values for the axis drawn at the first position (default is 12 o'clock).

SPKLABEL=CATEGORY | NONE

labels the chart spokes with the category of the variable that is being charted.

NONE suppresses the labels. The default is CATEGORY; however, if the STARLEGEND= option is specified, the default is NONE.

STARAXIS= (AXIS<1...99><, . . . ,AXIS<1...99>>)**STARAXES= (AXIS<1...99><, . . . ,AXIS<1...99>>)**

assigns one or more axis definitions to the axis spokes in the radar chart. GRADAR displays axis spokes clockwise, starting at the 12 o'clock position. The axis definitions that are specified using the STARAXIS= option are assigned consecutively to the spokes, starting from the first spoke. For example, STARAXIS=(AXIS3, AXIS1, AXIS2) assigns the AXIS3 statement's definition to the first axis spoke (at the 12 o'clock position), the AXIS1 statement's definition to the second axis spoke, and the AXIS2 statement's definition to the third axis spoke.

The axis definitions are assigned consecutively, and you cannot skip a spoke. For example, to assign a definition to the seventh spoke, you must also assign definitions to the first six spokes. However, you do not have to assign definitions to all of the spokes. Any remaining axis spokes on the GRADAR chart are displayed with the default settings. For example, if the STARAXIS= option specifies three definitions and the chart has more than three axis spokes, the fourth and remaining spokes are displayed with the default settings. If there are more definitions specified than there are axis spokes in the chart, the excess definitions are ignored.

Featured in: Example 11 on page 1210.

Not supported by: ActiveX

STARCIRCLES=(values)**STARCIRCLE=(values)**

specifies reference circles that are superimposed on the stars that are produced for a radar chart. All of the circles are displayed and centered at each point plotted on the primary chart. The *value* determines the diameter of the circle as follows: a value of 0.0 specifies a circle with the *inner radius*, which displays a circle at the minimum data value, and a value of 1.0 specifies a circle with the *outer radius*, which is the length of the spokes in the chart. In general, a value of *h* specifies a circle with a radius equal to $\text{inradius} + h \times (\text{outradius} - \text{inradius})$.

For example, the values 0.0 and 1.0 correspond to an *inner circle* and an *outer circle*. The value 0.5 specifies a circle with a radius of $\text{inradius} + 0.5 \times (\text{outradius} - \text{inradius})$, or a circle halfway between the inner circle and the outer circle. Likewise, the value 0.25 specifies a circle one-fourth of the way from the inner circle to the outer circle.

To specify the line types for the circles, use the LSTARCIRCLES= option. To specify colors for the circles, use the CSTARCIRCLES= option.

Featured in: Example 3 on page 1199.

STARFILL= lists of (SOLID | EMPTY) one for each star

determines whether the star(s) in the radar chart are empty or filled with a solid color. Valid values are EMPTY (the default) and SOLID. If there are multiple stars in the chart, specify, in parentheses, a separate value for each star.

If STARFILL=SOLID and CSTARFILL= is not specified, then the star is filled with the color specified on the CSTAR= option.

If STARFILL= is not set or is set to EMPTY, then CSTARFILL= is ignored.

Featured in: Example 6 on page 1204.

STARINRADIUS=value**STAROUTRADIUS=value**

STARINRADIUS= specifies inner radius of stars. The value must be specified in horizontal percent screen values, and it must be less than the value that is specified with the STAROUTRADIUS= option. The inner radius of a star is the distance from the center of the star to the circle that represents the lower limit of the standardized vertex variables. The lower limit can correspond to the minimum value, a multiple of standard deviations below the mean, or a lower specification limit. The default value is one-third of the outer radius.

Not supported by: ActiveX

STARLEGEND=CLOCK | CLOCK0 | NUMBER | DEGREES | NONE

specifies the style of the legend used to identify the vertices of stars that are produced for a radar chart. The following keywords are available:

CLOCK	identifies the vertex variables by their positions on the clock (starting with 12:00).
CLOCK0	identifies the vertex variables by their positions on the clock (starting with 0:00 corresponding to 12:00).
NUMBER	identifies the vertex variables by numbers, with 1 corresponding to 12 o'clock.
DEGREES	identifies the vertex variables by angles in degrees, with 0 degrees corresponding to 12 o'clock.
NONE	suppresses the legend. This is the default.

Featured in: Example 4 on page 1201.

STARLEGENDLAB='legend-label'

specifies the label displayed to the left of the legend for stars requested with the STARLEGEND= option. The label can be up to 16 characters and must be enclosed in quotes. The default label is *Vertices*.

Featured in: Example 4 on page 1201.

Not supported by: ActiveX

STARSTART=*value*

specifies the vertex angle for the first variable that is specified on the CHART statement. Vertex angles for the remaining variables are uniformly spaced clockwise and assigned in the order listed. You can specify the value in the following ways:

- *Clock position:* If you specify the value as a time literal (between '0:00'T and '12:00'T), the corresponding clock position is used for the first vertex variable. For example, '12:00'T indicates the 12 o'clock position, '03:00'T the 3 o'clock position (90 degrees), and '09:00'T the 9 o'clock position (270 degrees).
- *Degrees:* To specify a value in degrees you must specify a negative number. (This is to distinguish degrees from clock values, which are stored internally as positive numbers.) If you specify a negative number, the absolute value is used for the first vertex angle in degrees. Here, 0 degrees corresponds to 12:00, -90 degrees to 3:00, and -270 degrees to 9:00. Always specify the value in degrees as a negative number.

The default value is zero, so the first vertex variable is positioned at 12:00.

STARTYPE=CORONA | POLYGON | RADIAL | SPOKE | WEDGE

specifies the style of the stars that are produced for a radar chart. The following keywords are available:

CORONA	polygon with star-vertices emanating from the inner circle
POLYGON	closed polygon
RADIAL	rays emanating from the center
SPOKE	rays emanating from the inner circle
WEDGE	closed polygon with rays from the center to the full spoke length (this is the default).

Featured in: Example 5 on page 1202 and Example 8 on page 1207.

SUMVAR=*summary-variable*

specifies the variable used to calculate the sum or means.

TILELEGEND=*variable*

specifies a variable used to add a legend for CTILES= colors. The variable can have a formatted length less than or equal to 32. If a format is associated with the variable, then the formatted value is displayed. The TILELEGEND= option must be used in conjunction with the CTILES= option for filling the tiles in a chart. If CTILES= is specified and TILELEGEND= is not specified, a color legend is not displayed.

The values of the CTILES= and TILELEGEND= variables should be consistent for all observations with the same level of the classification variables. The value of the TILELEGEND= variable is used to identify the corresponding color value of the CTILES= variable in the legend.

Not supported by: ActiveX

TILELEGLABEL=*'label'*

specifies a label displayed to the left of the legend that is created when you specify a TILELEGEND= variable. The label can be up to 16 characters and must be enclosed in quotes. The default label is *Tiles*.

Not supported by: ActiveX

WFRAME=*n***WAXIS=*n***

specifies the width in pixels for the frame lines. The default width is 1.

Not supported by: ActiveX

WEIGHT=*numeric-variable*

specifies a weight variable used to construct weighted radar charts. The WEIGHT= variable must be numeric, and its values must be non-negative (non-integer values are permitted). If a WEIGHT= variable is not provided for a chart variable, the weights applied to that chart variable are assumed to be 1.

Not supported by: ActiveX

WSPOKES=*line-width***WSPOKE=*line-width***

specifies the width in pixels of the spokes in a radar chart. The default width is 1.

WSTARCIRCLES=(*line-widths*)**WSTARCIRCLE=(*line-widths*)**

specifies the width in pixels of the outline of circles requested by the STARCIRCLES= option. The default width is 1. Works only if STARCIRCLES is specified.

WSTARS=*line-width* | (*line-widths*)**WSTAR=*line-width* | (*line-widths*)**

specifies the width in pixels of the outline of stars that are produced for a radar chart. The default width is 1.

Featured in: Example 3 on page 1199.

Examples

Example 1: Generating the Data Set for the GRADAR Examples

Procedure features: Data set generation

Sample library member: GGDDSGR1

All of the GRADAR procedure examples in this help system use the data from this SAS code. You must submit this code before you can run any of the other examples for GRADAR.

During the manufacture of a metal-oxide semiconductor (MOS) capacitor, different cleaning processes were used by two manufacturing systems that were operating in parallel. Process A used a standard cleaning solution, while Process B used a different cleaning mixture that contained less particulate matter. For five consecutive days the causes of failure with each process were observed, recorded, and saved in the SAS data set called FAILURE.

```
data failure;
  label cause = 'Cause of Failure' ;
  input process $ 1-9 day $ 13-19 cause $ 23-36 count 40-41;
  datalines;
Process A   March 1   Contamination   15
Process A   March 1   Corrosion       2
```

Process A	March 1	Doping	1
Process A	March 1	Metallization	2
Process A	March 1	Miscellaneous	3
Process A	March 1	Oxide Defect	8
Process A	March 1	Silicon Defect	1
Process A	March 2	Contamination	16
Process A	March 2	Corrosion	3
Process A	March 2	Doping	1
Process A	March 2	Metallization	3
Process A	March 2	Miscellaneous	1
Process A	March 2	Oxide Defect	9
Process A	March 2	Silicon Defect	2
Process A	March 3	Contamination	20
Process A	March 3	Corrosion	1
Process A	March 3	Doping	1
Process A	March 3	Metallization	0
Process A	March 3	Miscellaneous	3
Process A	March 3	Oxide Defect	7
Process A	March 3	Silicon Defect	2
Process A	March 4	Contamination	12
Process A	March 4	Corrosion	1
Process A	March 4	Doping	1
Process A	March 4	Metallization	0
Process A	March 4	Miscellaneous	0
Process A	March 4	Oxide Defect	10
Process A	March 4	Silicon Defect	1
Process A	March 5	Contamination	23
Process A	March 5	Corrosion	1
Process A	March 5	Doping	1
Process A	March 5	Metallization	0
Process A	March 5	Miscellaneous	1
Process A	March 5	Oxide Defect	8
Process A	March 5	Silicon Defect	2
Process B	March 1	Contamination	8
Process B	March 1	Corrosion	2
Process B	March 1	Doping	1
Process B	March 1	Metallization	4
Process B	March 1	Miscellaneous	2
Process B	March 1	Oxide Defect	10
Process B	March 1	Silicon Defect	3
Process B	March 2	Contamination	9
Process B	March 2	Corrosion	0
Process B	March 2	Doping	1
Process B	March 2	Metallization	2
Process B	March 2	Miscellaneous	4
Process B	March 2	Oxide Defect	9
Process B	March 2	Silicon Defect	2
Process B	March 3	Contamination	4
Process B	March 3	Corrosion	1
Process B	March 3	Doping	1
Process B	March 3	Metallization	0
Process B	March 3	Miscellaneous	0
Process B	March 3	Oxide Defect	10
Process B	March 3	Silicon Defect	1

```

Process B   March 4   Contamination   2
Process B   March 4   Corrosion       2
Process B   March 4   Doping          1
Process B   March 4   Metallization   0
Process B   March 4   Miscellaneous   3
Process B   March 4   Oxide Defect    7
Process B   March 4   Silicon Defect  1
Process B   March 5   Contamination   1
Process B   March 5   Corrosion       3
Process B   March 5   Doping          1
Process B   March 5   Metallization   0
Process B   March 5   Miscellaneous   1
Process B   March 5   Oxide Defect    8
Process B   March 5   Silicon Defect  2
run;
quit;

```

Example 2: Producing a Basic Radar Chart

Procedure features:

```

FREQ=
CSTARS=
CFRAME=

```

Sample library member: GGDPBRC1

In a radar chart, the vertices are determined by the levels of a single variable, which is specified on the CHART statement. In this example, the variable CAUSE is specified as the chart variable. The spokes in the chart start at the twelve o'clock position and go in a clockwise order. The output shows that Contamination and Oxide Defects are the most frequently occurring problems.

This example features the following options:

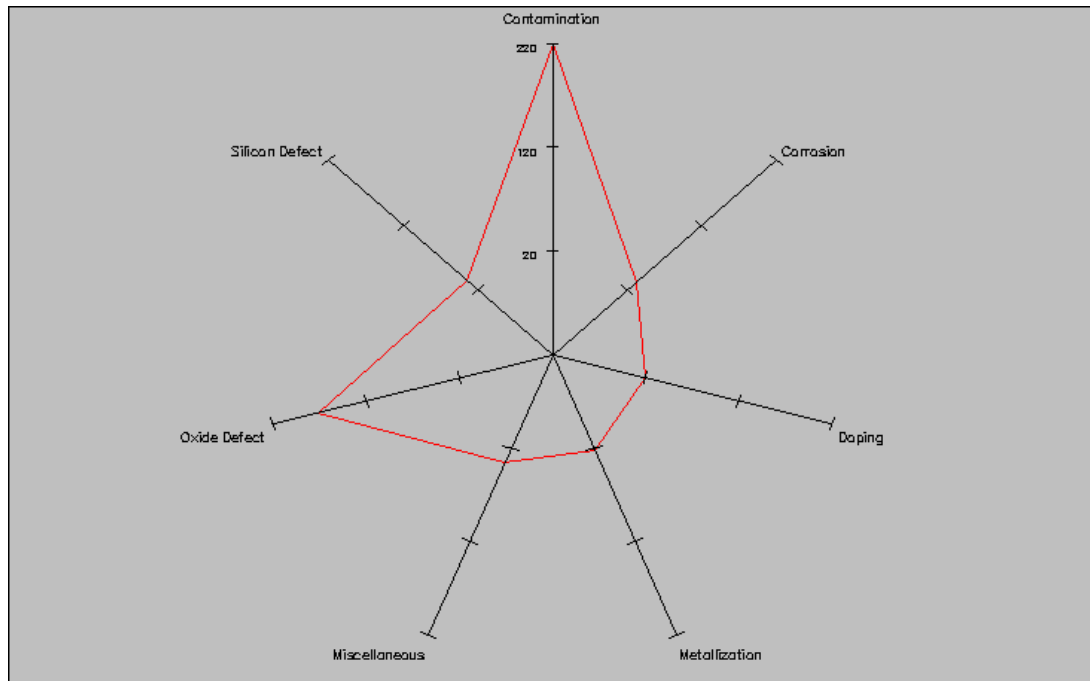
- FREQ= specifies variable COUNT to score vertex lengths. Thus, the values of COUNT weigh the contribution of each observation in the computation of the chart statistic.
- CSTARS= specifies a color for the star's outline.
- CFRAME= specifies a color to use to fill the chart's axis area.

To run the code in this example, you must first generate the FAILURE data set Example 1 on page 1196.

```

proc gradar data=failure;
  chart cause / freq=count
              cstars=red
              cframe=ltgray;
run;
quit;

```



Example 3: Overlaying Radar Charts

Procedure features:

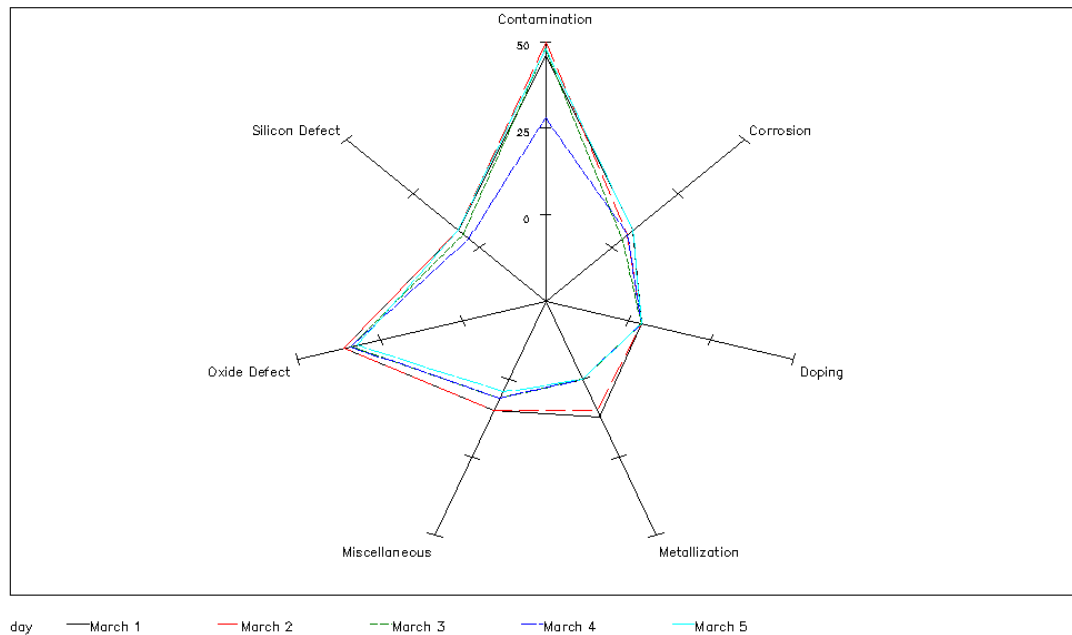
OVERLAY=

Sample library member: GGDOVRC1

The most typical way that radar charts are displayed is to overlay the charts on top of each other. To produce an overlay chart, use the OVERLAY= option on the CHART statement. On OVERLAY=, specify a classification variable whose values will determine the charts to be overlaid. This example shows two blocks of code. The first block generates a simple overlay chart, and the second block uses options to enhance the chart appearance.

In the following example, OVERLAY= specifies variable DAY as the overlay variable. To run the code in this example, you must generate the FAILURE data set Example 1 on page 1196.

```
proc gradar data=failure;
  chart cause / freq=count
              overlay=day;
run;
quit;
```

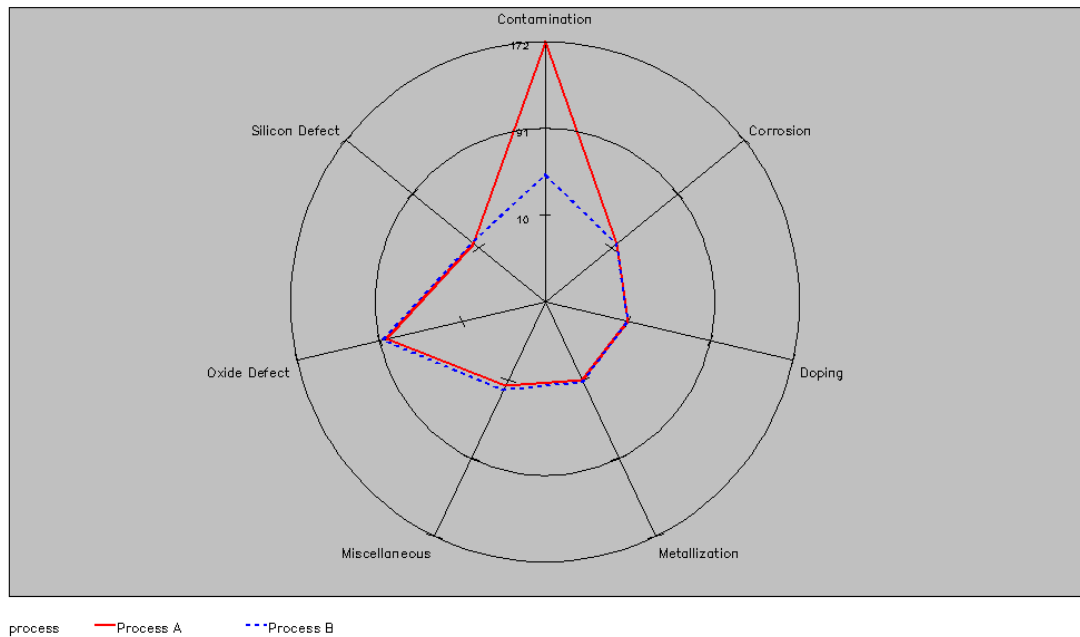


The code above relies on default settings. You can enhance a chart by specifying additional options. The code below specifies variable `PROCESS` as the overlay variable and features the following options:

- `STARCIRCLES=` determines that two reference circles are superimposed on the star charts. The value 1.0 determines that a circle with a radius equal to the spoke length is displayed. The value 0.5 determines that a circle is displayed half way between the outer circle and the smallest circle (value 0.0) that could be drawn for the chart. The value 0.0 would display a circle at the minimum data value, which does not mean that it is actually 0. For example, for data values of 4, 8, 10, and 12, `STARCIRCLES=(0.0 1.0)` would draw circles at 4 and 12.
- `CSTARCIRCLES=` determines that both circles are colored black. If this option were not used, both stars would be colored with the first color that is listed on `CSTARS=`.
- `CSTARS=` determines that the first star, which represents Process A, is colored red, and that the second star, which represents Process B, is colored blue.
- `WSTARS=` specifies pixel widths for the outlines of both stars. A separate width is required for each star.

```
proc gradar data=failure;
  chart cause / freq=count
          overlay=process
          starcircles=(0.5 1.0)
          cstarcircles=black
          cstars=(red blue)
          wstars=(2 2)
          cframe=ltgray;

run;
quit;
```



Example 4: Tiling Radar Charts

Procedure features:

ACROSS=
 INTERTILE=
 STARLEGEND=CLOCK
 STARLEGENDLAB=

Sample library member: GGDTLRC1

As an alternative to overlaying multiple radar charts Example 3 on page 1199, you can *tile* charts horizontally, vertically, or in both directions Example 5 on page 1202 using the ACROSS= and/or DOWN= options. Each cell in the output corresponds to a level of the classification variable. By default, the cells are arranged in alphabetical order of the values of the variable from top to bottom. The *key cell* is the left cell (corresponding to *PROCESS = Process A* in this example).

The output in this example shows that the main difference in the Radar frequencies for Process A and Process B is a drop in contamination using Process B.

This example features the following options:

- ACROSS= specifies variable PROCESS as the categorical variable whose values determine the number of charts that are tiled.
- INTERTILE= specifies 0.5 percent screen units as the distance between tiles in the chart.
- STARLEGEND=CLOCK generates a legend that identifies spoke positions. Value CLOCK determines that the positions are identified using a clock metaphor.
- STARLEGENDLAB= specifies the category-legend label *Failure Causes*.

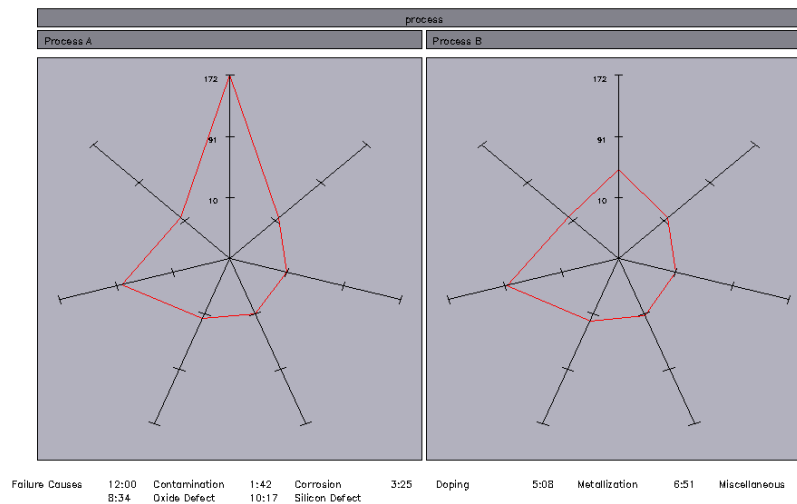
To run the code in this example, you must generate the FAILURE data set Example 1 on page 1196.

```

proc gradar data=failure;
  chart cause / across=process
           freq=count
           intertile=0.5
           cframe=CXB3B2BF
           cframetop=CX83838C
           starlegend=clock
           starlegendlab='Failure Causes'
           cstars=red;

run;
quit;

```



Example 5: Using Multiple Classification Variables in Radar Charts

Procedure features:

ACROSS=
 DOWN=
 CFRAMENLEG=
 CFRAMETOP=
 CFRAMESIDE=
 STARTYPE=
 INTERTILE=
 NCOLS=
 NROWS=

Sample library member: GGDMCVR1

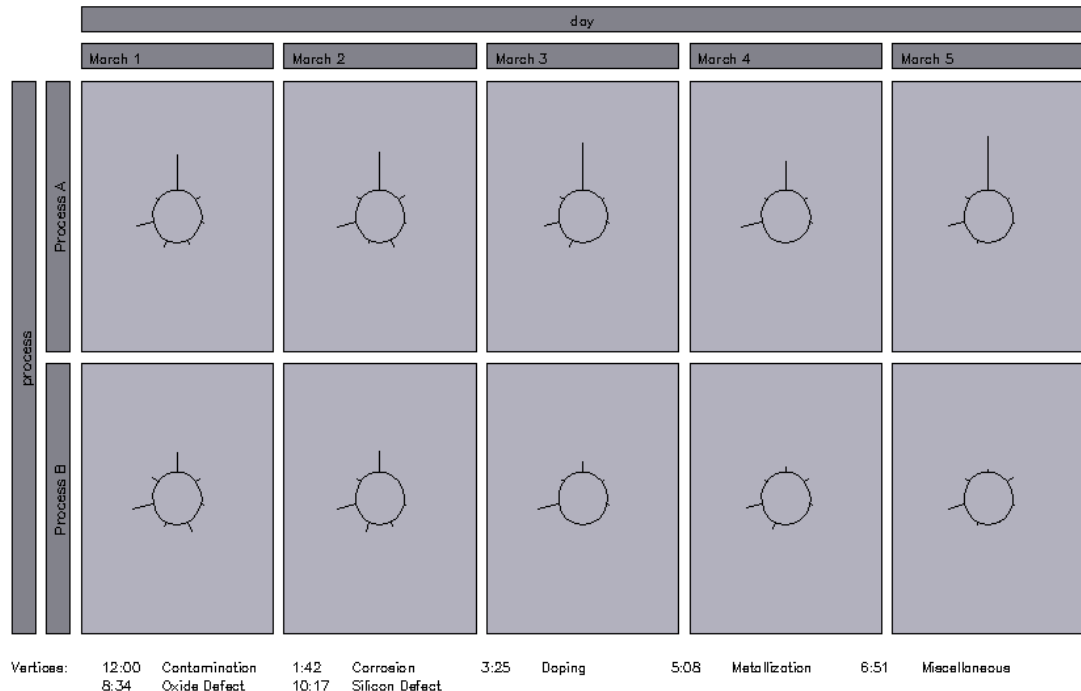
You can study the effects of two classifications simultaneously with a two-way comparative radar chart. This arrangement provides the opportunity to discover both one-way marginal effects and interaction effects. To produce the chart, use both the ACROSS= and DOWN= options.

This example features the following options:

- ACROSS= specifies variable DAY as the variable whose values determine the rows in the chart matrix.
- DOWN= specifies variable PROCESS as the variable whose values determine the columns in the chart matrix.
- CFRAMENLEG= specifies that the legend be framed and filled with the specified color.
- CFRAMETOP= specifies a color for the top labels.
- CFRAMESIDE= specifies a color for the side labels.
- STARTYPE= determines that the stars are displayed with rays emanating from the inner circle.
- INTERTILE= specifies the distance between cells in the tiled chart.
- NCOLS= specifies the number of columns in the chart.
- NROWS= specifies the number of rows in the chart.

To run the code in this example, you must generate the FAILURE data set Example 1 on page 1196.

```
proc gradar data=failure;
  chart cause / across=day
              down=process
              freq=count
              cframe=CXB3B2BF
              cframetop=CX83838C
              cframeside=cx83838c
              startype=spoke
              intertile=1
              ncols=5
              nrows=2
              starlegend=clock
              spklabel=none;
run;
quit;
```



Example 6: Filling the Stars in Radar Charts

Procedure features:

STARFILL=

CSTARFILL=

Sample library member: GGDFSRC1

By default, the stars in a radar chart are empty. To fill the stars with a solid color, use STARFILL=SOLID. When the stars are solid filled, the outline of each underlying star and the spokes in the chart are drawn on top of the stars so that each chart can be easily seen.

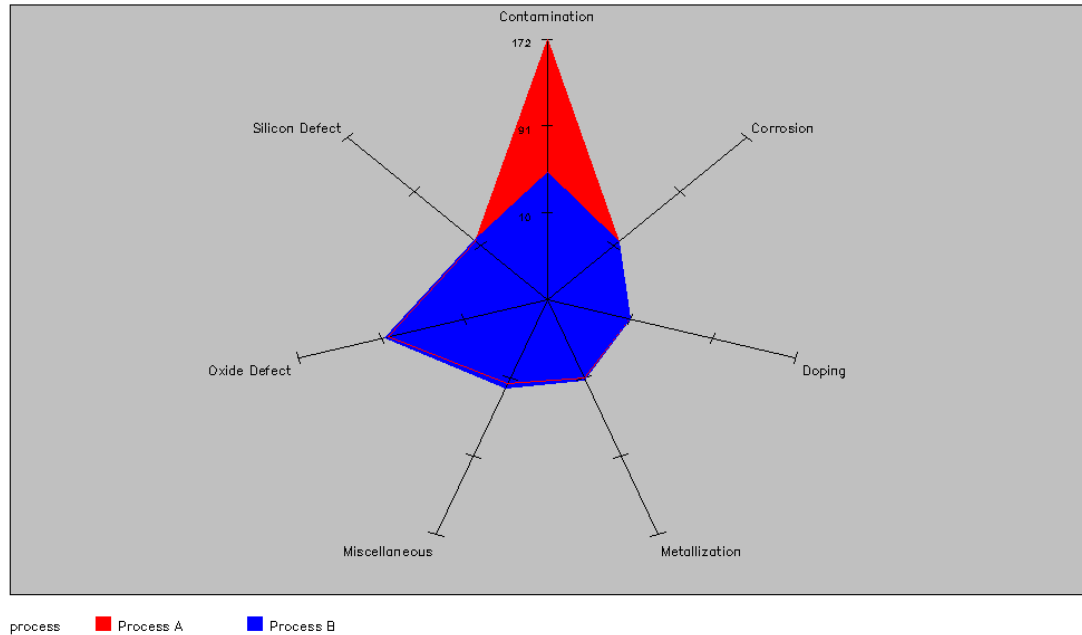
This example features the following options:

- STARFILL= specifies a solid fill for each of the two stars that are generated by the program. Because the default fill is EMPTY, STARFILL= must specify SOLID two times; otherwise, the first star would be solid filled, but the second star would be empty.
- CSTARFILL= specifies colors for the two stars.

To run the code in this example, you must generate the FAILURE data set Example 1 on page 1196.

```
proc gradar data=failure;
  chart cause / overlay=process
           freq=count
           cstarfill=(red blue)
           starfill=(solid solid)
           cframe= ltgray;
```

```
run;
quit;
```



Example 7: Using Images in Radar Charts

Procedure features:

IBACK=

IMAGESTYLE=

CSPOKES=

Sample library member: GGDUIRC1

As with other SAS/GRAPH procedures, the GRADAR procedure enables you to display images in your charts. You can display the image in the graph background area using on the IBACK= graphics option. Or you can display the image on the chart's frame using the GRADAR procedure's IFRAME= option. Either way, you can use an IMAGESTYLE= option to indicate the image treatment.

This example shows both techniques. To run the code that is shown, you must generate the FAILURE data set Example 1 on page 1196.

The first block of code displays an image in the graph background area. It features the following options:

- IBACK= on the GOPTIONS statement specifies the image file.
- IMAGESTYLE= on the GOPTIONS statement specifies that the image be scaled to fit within the chart's background area.
- CSPOKES= on the GRADAR procedure specifies a color for the chart spokes so that they can be easily seen against the background image.

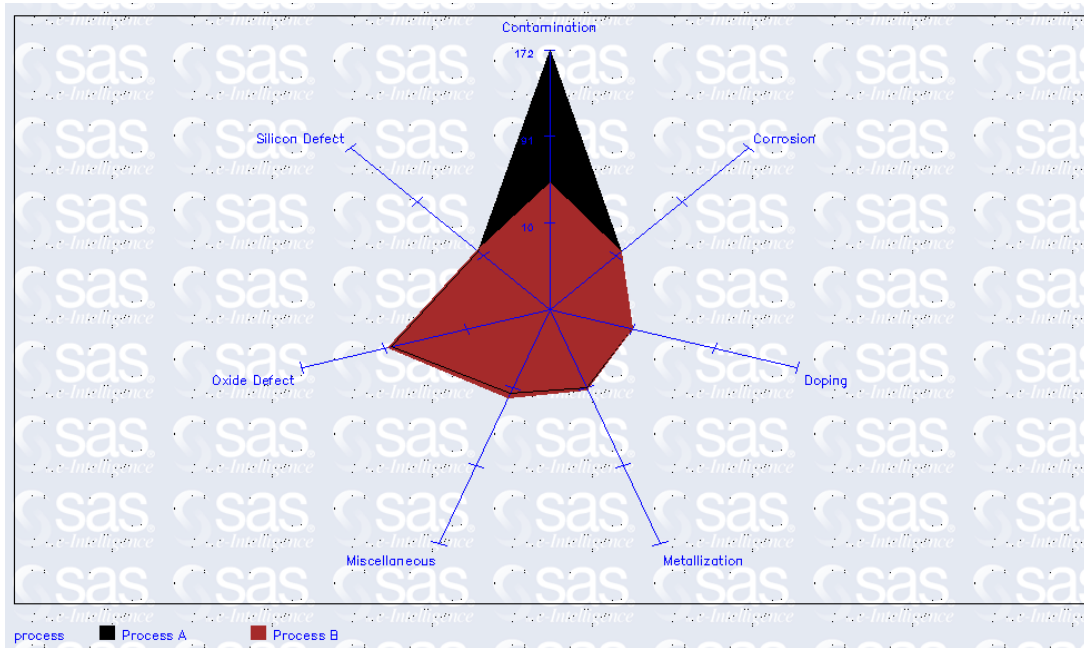
```

filename backingg 'C:\My Documents\sas\gradar\iback_image.gif';
/* use of the IBACK= option on the GOPTIONS statement */
goptions iback=backingg imagestyle=tile;

proc gradar data=failure;
  chart cause / overlay=process
    freq=count
    cstarfill=(black brown)
    starfill=(solid solid)
    cspokes=blue
    ctext=blue;

run;
quit;

```



In this next block of code, the image is displayed in the radar chart's frame, so only GRADAR options are used to specify the image. The code features the following options:

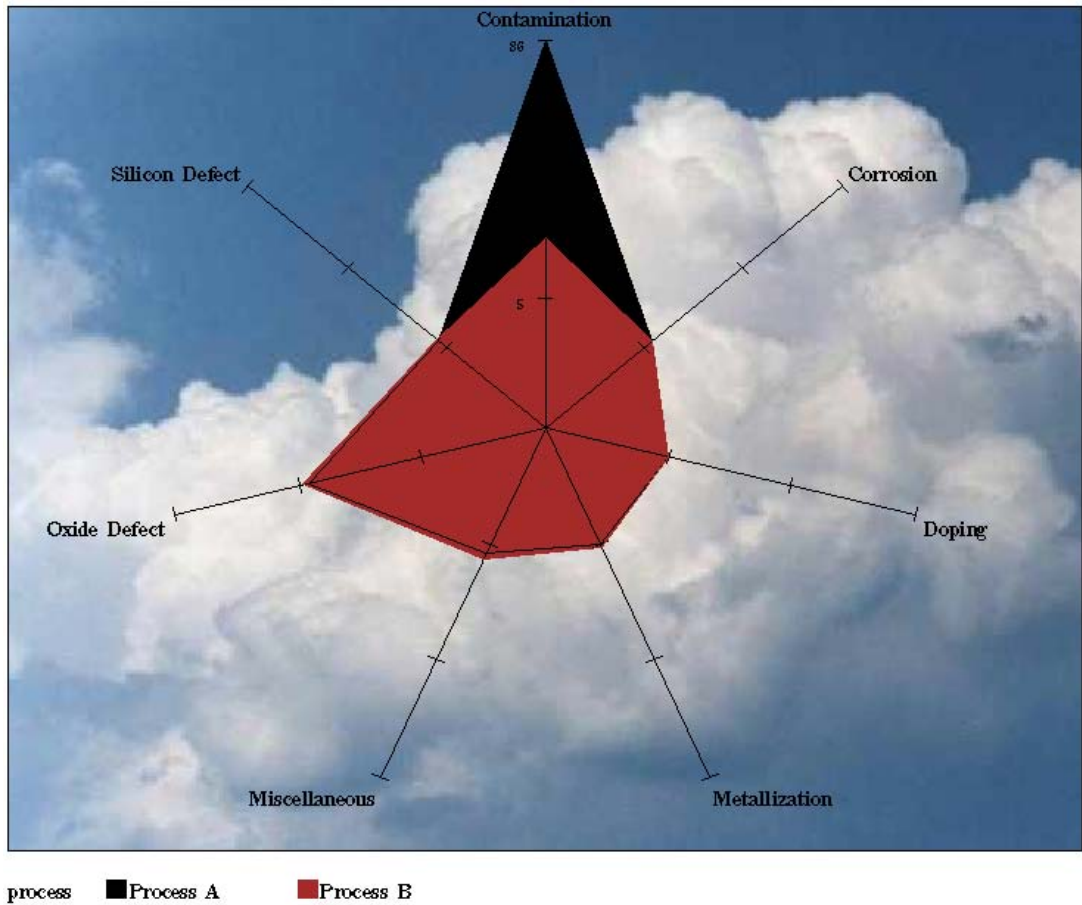
- IFRAME= specifies the image file to display in the frame.
- IMAGESTYLE= specifies that the image be scaled to fit within the frame.
- FONT= specifies a font for all of the chart text so that it can be easily seen against the background image.

```

goptions reset=all; /* cancel the previous iback option */
filename frameimg 'C:\My Documents\sas\gradar\clouds.gif';
/* use the IFRAME= option in the CHART statement */
proc gradar data=failure;
  chart cause / overlay=process
    freq=count
    cstarfill=(black brown)
    starfill=(solid solid)
    iframe=frameimg
    imagestyle=fit

```

```
font=centb;
run;
quit;
```



Example 8: Changing the Star Type in Radar Charts

Procedure features:

OVERLAY=

FREQ=

STARTYPE=

Sample library member: GGDCSTR1

By default, the stars in a radar chart are displayed as wedges. You can specify an alternative style with the `STARTYPE=` option. This example specifies `STARTYPE=CORONA`.

To run the code in this example, you must generate the FAILURE data set Example 1 on page 1196.

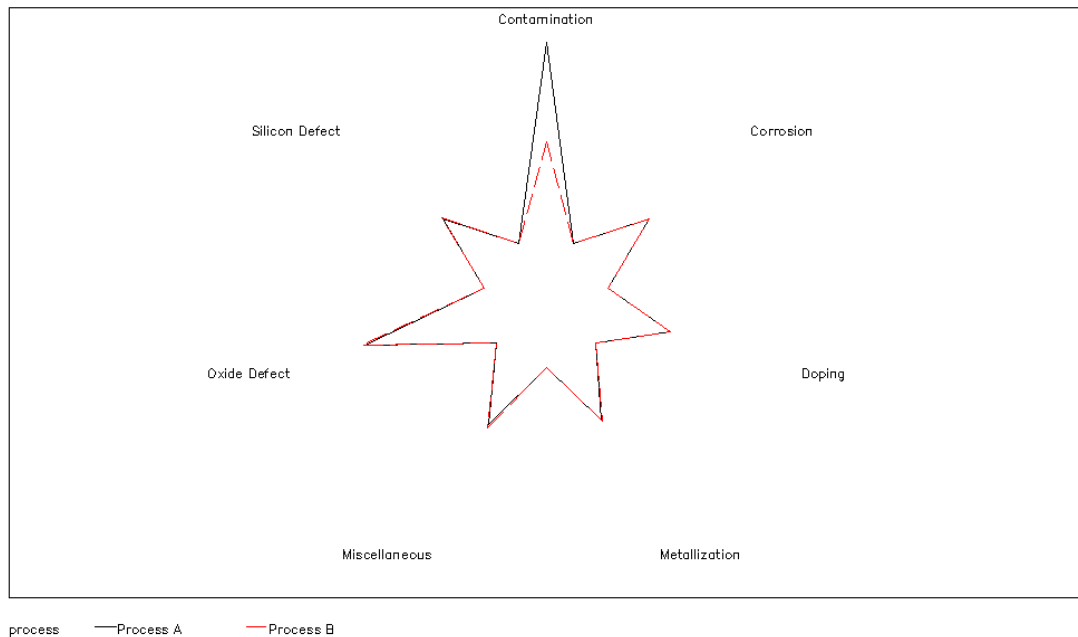
```
proc gradar data=failure;
  chart cause / overlay=process
```

```

freq=count
startype=corona;

run;
quit;

```



Example 9: Using Color and Line Styles in Radar Charts

Procedure features:

CSTARS=

LSTARS=

CFRAME=

Sample library member: GGDUCLS1

For overlay charts with multiple stars, the lines for the stars are rotated through different line styles and colors so that the different stars can be easily seen. Rather than relying on the default rotation patterns, you can control the line colors and line styles with the CSTARS= and LSTARS= options.

This example features the following options:

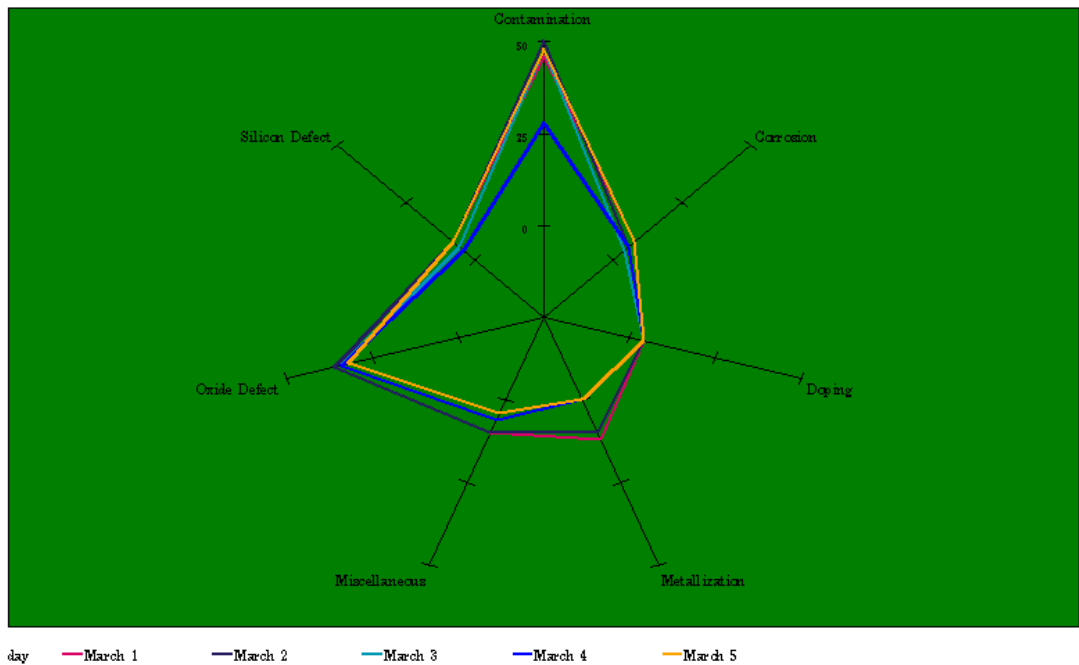
- CSTARS= specifies a different color for each of the star outlines in the chart. In this example the colors are specified as hexadecimal values, but you can use any valid SAS/GRAPH color names.
- LSTARS= specifies a solid line as the line style for each star outline.
- CFRAME= specifies a background color to display on top of the image that is specified on the GOPTIONS statement's IBACK= option.

To run the code in this example, you must generate the FAILURE data set Example 1 on page 1196.

```

goptions htext=.95 cell iback='external-file'
        border ftext=centb;
proc gradar data=failure;
  chart cause / overlay=day
        freq=count
        cstars=(cxdc0369, cx261e62,
                cx0099b6, cxb0885, cxffa300)
        cframe=cxcccffc
        starlegend=none
        wstars=2 2 2 2 2
        lstars=1 1 1 1 1
        ;
run;
quit;

```



Example 10: Specifying the Mode for a Radar Chart

Procedure features:

MODE=
 ACROSS=
 FREQ=

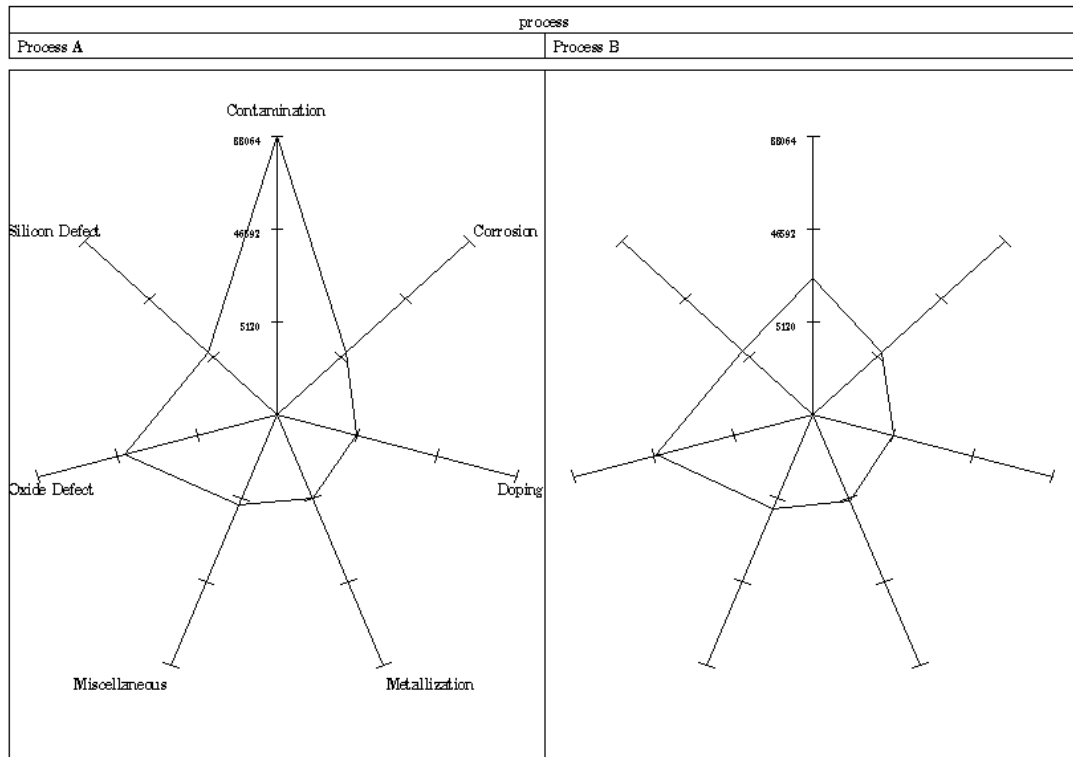
Sample library member: GGDSMRC1

The MODE= option specifies the display mode for a radar chart. If a radar chart is generated without the MODE= option set, the labels are sometimes too small and illegible. This example uses the SHARE= keyword, which is one of three keywords available for the MODE= option. The SHARE= keyword shares the drawing space

between the text and the graph. The result is that the text is enlarged and becomes legible.

To run the code in this example, you must generate the FAILURE data set Example 1 on page 1196.

```
proc gradar data=failure;
  chart cause / across=process
              mode=share
              freq=count
              ;
run;
quit;
```



Example 11: Assigning Axis Definitions to Axis Spokes

Procedure features:

STARAXIS=

FREQ=

Sample library member: GGDADAS1

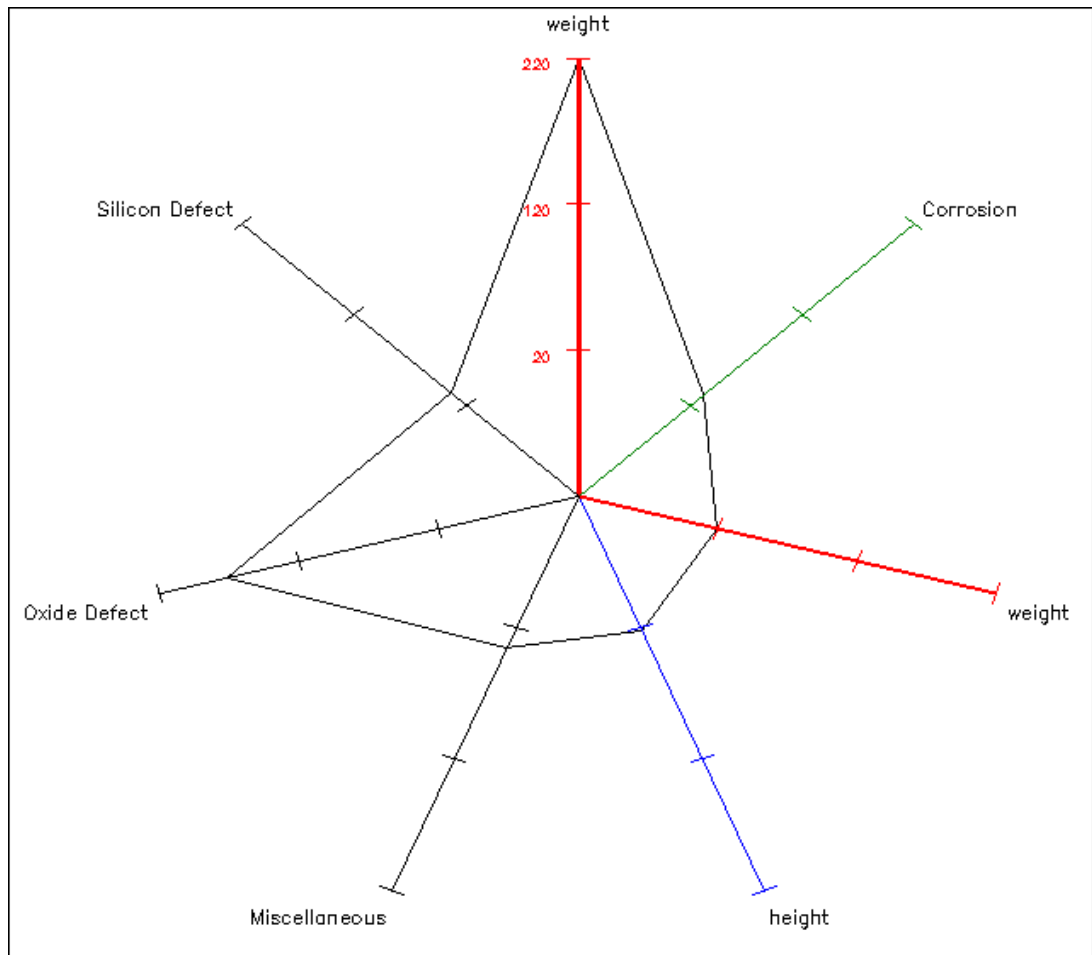
The STARAXIS= option allows you to assign axis definitions to spokes in a radar chart. In this example, three axis definitions are specified. However, there are seven spokes in the radar chart. Spokes one and three are colored the same because they both use the definition axis1. Spoke two uses the definition axis3. Spoke 4 uses the definition axis2, and spokes five through seven receive the default settings.

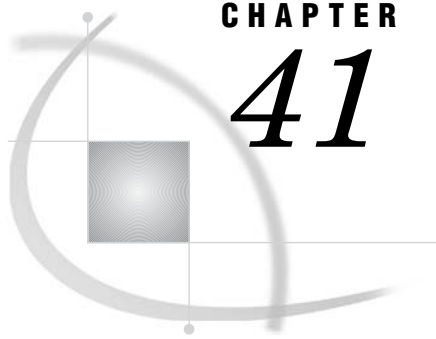
To run the code in this example, you must generate the FAILURE data set Example 1 on page 1196.

```
axis1 color=red
      label=('weight')
      major=(height=.75 width=3)
      width=3;
axis2 color=blue
      label=('height');
axis3 color=green;

proc gradar data=failure;
  chart cause / staraxis=(axis1, axis3, axis1, axis2)
              freq=count;

run;
quit;
```





CHAPTER

41

The GREDUCE Procedure

<i>Overview</i>	1213
<i>Concepts</i>	1215
<i>About the Input Map Data Set</i>	1215
<i>About Unmatched Area Boundaries</i>	1215
<i>Procedure Syntax</i>	1215
<i>PROC GREDUCE Statement</i>	1216
<i>ID Statement</i>	1217
<i>Using the GREDUCE Procedure</i>	1218
<i>Specifying Density Levels</i>	1218
<i>Subsetting a Map Data Set</i>	1220
<i>Examples</i>	1220
<i>Example 1: Reducing the Map of Canada</i>	1220
<i>References</i>	1222

Overview

The GREDUCE procedure processes map data sets so that they can draw simpler maps with fewer boundary points. It creates an output map data set that contains all of the variables in the input map data set plus a new variable named DENSITY. For each observation in the input map data set, the procedure determines the significance of that point for maintaining a semblance of the original shape and gives the observation a corresponding DENSITY value.

You can then use the value of the DENSITY variable to create a subset of the original map data set. The observations in the subset can draw a map that retains the overall appearance of the original map but contains fewer points, requires considerably less storage space, and can be drawn much more quickly.

GREDUCE does not produce any graphics output. Instead, it produces an output map data set that can become either

- the input map data set for the GMAP procedure
- the input map data set for a DATA step that removes points from the map.

Figure 41.1 on page 1214 and Figure 41.2 on page 1214 illustrate the effect of reduction on a typical map data set. Figure 41.1 on page 1214 uses observations with all DENSITY values as input to the GMAP procedure.

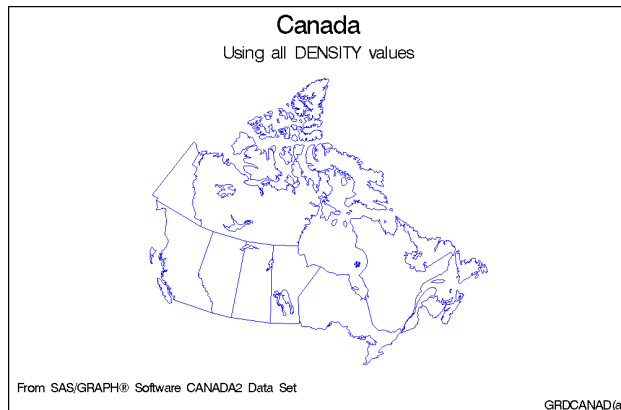
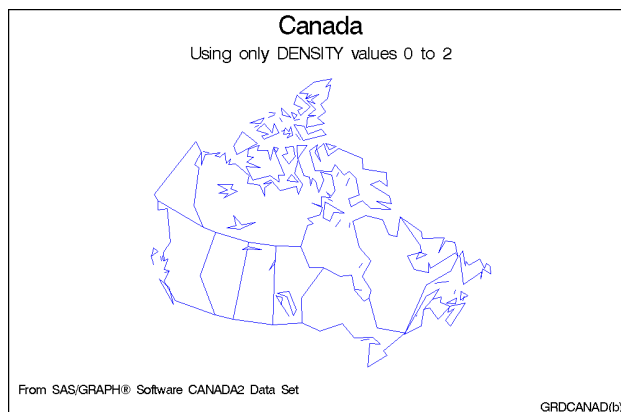
Figure 41.1 CANADA2 Map before Reduction (GRDCANAD(a))

Figure 41.2 on page 1214 uses only those observations with a DENSITY value of 0 or 2 as input to the GMAP procedure.

Figure 41.2 CANADA2 Map after Reduction (GRDCANAD(b))

The program for these maps is in Example 1 on page 1220.

The reduced map shown in Figure 41.2 on page 1214 retains the overall shape of the original but requires only 463 observations compared to the 4302 observations that are needed to produce the map in Figure 41.1 on page 1214.

Note: Many of the map data sets that are supplied by SAS Institute already have been processed by GREDUCE. If the map data set contains a DENSITY variable, you do not need to process the data set using GREDUCE. \triangle

See also Chapter 42, “The GREMOVE Procedure,” on page 1223 for more information on how to

- combine groups of unit areas into larger unit areas to create regional maps
- remove some of the boundaries in a map and create a subset of a map that combines the original areas.

Concepts

About the Input Map Data Set

The input map data set must be a traditional map data set and contain these variables:

- a numeric variable named X that contains the horizontal coordinates of the map boundary points.
- a numeric variable named Y that contains the vertical coordinates of the map boundary points.
- one or more *identification variables* that uniquely identify the unit areas in the map. These variables are listed in the ID statement.

It also can contain

- one or more variables that identify groups of unit areas (for BY-group processing)
- the variable SEGMENT, which distinguishes nonconterminous segments of the unit areas.

Any other variables in the input map data set do not affect the GREDUCE procedure.

About Unmatched Area Boundaries

If you are using map data sets in which area boundaries do not match precisely (for example, if the boundaries were digitized with a different set of points), PROC GREDUCE will not be able to identify common boundaries properly, and this results in abnormalities in your maps. These abnormalities include mismatched borders, missing vertex points, stray lines, gaps, and distorted polygons.

If the points in the area boundaries match up except for precision differences, round each X and Y value in your map data set accordingly, using the DATA step function ROUND before using PROC GREDUCE. (See *SAS Language Reference: Dictionary* for information on the ROUND function.)

For example, if the map data set APPROX has horizontal and vertical coordinate values for interior boundaries of unit areas that are exactly equal only to three decimal places, then this DATA step creates a new map data set, EXACT, that will be better suited for use with PROC GREDUCE:

```
data exact;
  set approx;
  if x ne . then x=round(x,.001);
  if y ne . then y=round(y,.001);
run;
```

See “About Map Data Sets” on page 999 for additional information on map data sets.

Procedure Syntax

Requirements: Exactly one ID statement is required.

Reminder: The procedure can include the BY statement.

```
PROC GREDUCE <option(s)>;
```

ID *id-variable(s)*;

PROC GREDUCE Statement

Identifies the input and output map data sets. Optionally specifies the reduction criteria.

Requirements: An input map data set is required.

Syntax

PROC GREDUCE *<option(s)>*;

option(s) can be one or more options from any or all of the following categories:

- data set options:

DATA=*input-map-data-set*

OUT=*output-map-data-set*

- level options:

E1=*min-distance*

E2=*min-distance*

E3=*min-distance*

E4=*min-distance*

E5=*min-distance*

N1=*max-points*

N2=*max-points*

N3=*max-points*

N4=*max-points*

N5=*max-points*

Options

DATA=*input-map-data-set*

identifies the map data set that you want to process. By default, the procedure uses the most recently created SAS data set.

See also: “About the Input Map Data Set” on page 1215 and “SAS Data Sets” on page 29.

E1=*min-distance*

E2=*min-distance*

E3=*min-distance*

E4=*min-distance*

E5=*min-distance*

specify the minimum distance that a point must lie from a straight line segment to be included at density level 1, 2, 3, 4, or 5, respectively. That is, in a reduced curve of three points, the middle point is at least a distance that is *min-distance* from a straight line between the two outside points.

Express *min-distance* values in the units for the coordinate system of the input map data set. For example, if the input map data set contains coordinates that are expressed in radians, express the *min-distance* values in radians.

Specify the $E_n=$ values in decreasing order. For example, the $E_2=$ value should be less than the $E_1=$ value and so on.

N1=max-points

N2=max-points

N3=max-points

N4=max-points

N5=max-points

specify that for density level 1, 2, 3, 4, or 5, the boundary of a unit area should contain no more than *max-points* points.

Specify the $N_n=$ values in increasing order. For example, the $N_2=$ value should be greater than or equal to the $N_1=$ value and so on.

By default, if you omit $N_n=$ and $E_n=$, the GREDUCE procedure calculates values for the five $N_n=$ parameters using this formula:

$$N_n = n^2 \times N_{\max}/36$$

Here N_{\max} is the maximum number of points in any unit area in the input map data set. However, the restriction that the number of points for any level cannot be less than the number of points in level 0 still applies.

OUT=output-data-set

names the new map data set, which contains all of the observations and variables in the original map data set plus the new DENSITY variable. If the input map data set contains a variable named DENSITY, the GREDUCE procedure replaces the values of the variable in the output map data set. The original values of the DENSITY variable from the input map data set are not included in the output map data set.

By default, the GREDUCE procedure names the new data set that uses the $DATA_n$ naming convention. That is, the procedure uses the name $WORK.DATA_n$, where n is the next unused number in sequence. Thus, the first automatically named data set is DATA1, the second is DATA2, and so on.

ID Statement

Identifies the variable or variables that define the hierarchy of the current unit areas in the input map data set.

Requirements: At least one *id-variable* is required.

Featured in: Example 1 on page 1220.

Syntax

ID *id-variable(s)*;

Required Arguments

id-variable(s)

specifies one or more variables in the input map data set that identify unit areas. *Id-variable(s)* can be either numeric or character.

Each group of observations with a different ID variable value is evaluated as a separate unit area.

Using the GREDUCE Procedure

Specifying Density Levels

GREDUCE uses default criteria for determining the appropriate DENSITY variable value for each observation in the input map data set. If you do not want to use the default criteria, use PROC GREDUCE options to select

- the maximum number of observations for each DENSITY level
- the minimum distance that an intermediate point must lie from a line between two end points to be included in the level.

If you do not explicitly specify criteria, the procedure computes and uses default values.

GREDUCE creates seven density levels, numbered 0 through 6. Specify criteria for density levels 1 through 5. You cannot define criteria for level 0, which is reserved for map vertex points, such as common corners of unit areas. You also cannot define criteria for level 6, which is assigned to those points that do not meet the criteria for any lower level.

Specify the maximum number of observations per density level using $Nn=$ in the PROC GREDUCE statement, and specify the minimum point distance using $En=$. You must have knowledge of the X and Y variable values in the particular input map data set to determine appropriate values for $En=$. See the $En=$ and $Nn=$ option on page 1216 for details.

Figure 41.3 on page 1219 illustrates how to use the minimum distance parameter to determine which points belong in a particular density level. At density level n , only point C lies at a distance greater than the $En=$ value (70) from a line between points A and B. Thus, after reduction only point C remains between points A and B at density level n , and the resulting reduced boundary is shown in Figure 41.4 on page 1219. See Douglas and Peucker (1973) for details of the algorithm used.

Figure 41.3 Points in Data Set before Reduction

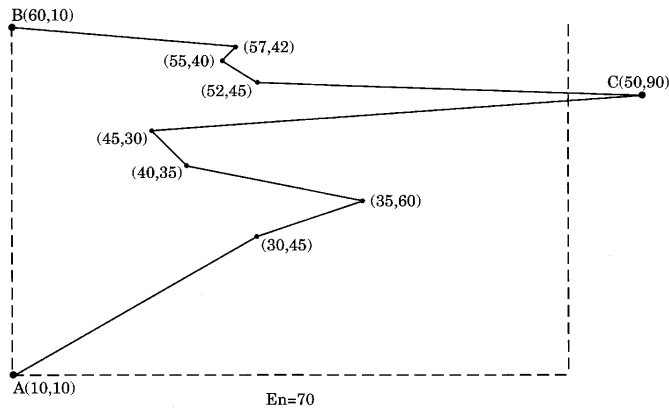
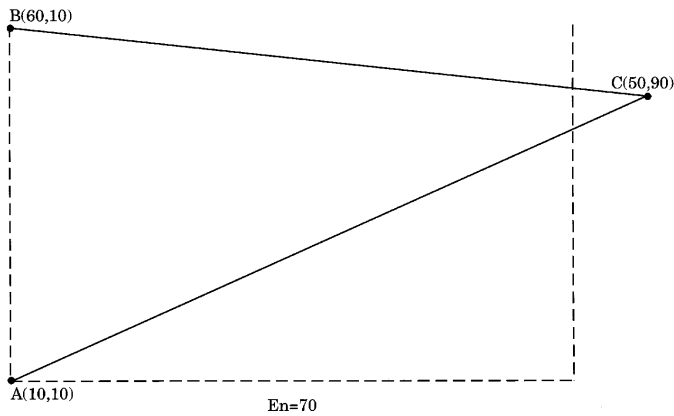


Figure 41.4 Points in Data Set at Density n after Reduction



GREDUCE uses the usual Euclidean distance formula to determine the distance between points. For example, the distance d between the points (x_0, y_0) and (x_1, y_1) is

GREDUCE uses the usual Euclidean distance formula to determine the distance between points. For example, the distance d between the points (x_0, y_0) and (x_1, y_1) is

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

If this distance function is not suitable for the coordinate system in your input map data set, transform the X and Y values to an appropriate coordinate system before using GREDUCE. An example of inappropriate coordinates is latitude and longitude values around one of the poles. In this case, the data values should be projected before they are reduced. See Chapter 39, “The GPROJECT Procedure,” on page 1161 for more information on map projection.

If you specify both $Nn=$ and $En=$ values for a density level, GREDUCE attempts to satisfy both criteria. However, the number of points for any level is never reduced below the number of points in density level 0. If you specify a combination of $Nn=$ or $En=$ values such that the resulting DENSITY values are not in order of increasing

density, a note is printed in the SAS log, and the DENSITY values are calculated in increasing order of density.

Subsetting a Map Data Set

A map data set that is processed by GREDUCE does not automatically result in a map that uses fewer points. By default, the GMAP procedure produces a map that uses all of the points in the map data set, even if the data set has been processed by the GREDUCE procedure. To decrease the number of points that produce the map, you must create a subset of the original data set using a DATA step or the WHERE= data set option. For example, to create a subset of a map that uses only the DENSITY values 0, 1, and 2, use this DATA step:

```
data smallmap;
  set map;
  if density <= 2;
run;
```

Alternatively, you can use WHERE= in the PROC GMAP statement:

```
proc gmap map=map(where=(density<=2))
  data=response;
```

Note: GREDUCE does not reduce the size of the output map data set compared to the input map data set. By default, the output map data set from PROC GREDUCE will be larger than the input map data set because it contains all of the variables and observations from the original data set, with the addition of the DENSITY variable if it was not present in the original data set. If the input map data set already had a DENSITY variable, the output map data set will be the same size as the input map data set. \triangle

Examples

The following example illustrates major features of the GREDUCE procedure. Because the example uses one of the map data sets that are supplied with SAS/GRAPH, you may need to replace *SAS-data-library* in the LIBNAME statement with the actual location of the SAS data library that contains the Institute-supplied map data sets on your system. Contact your SAS Software Consultant for the location of the map data sets at your site. If your site automatically assigns the libref MAPS to the SAS data library that contains the Institute-supplied map data sets, delete the LIBNAME statement in this example.

Example 1: Reducing the Map of Canada

Procedure features:

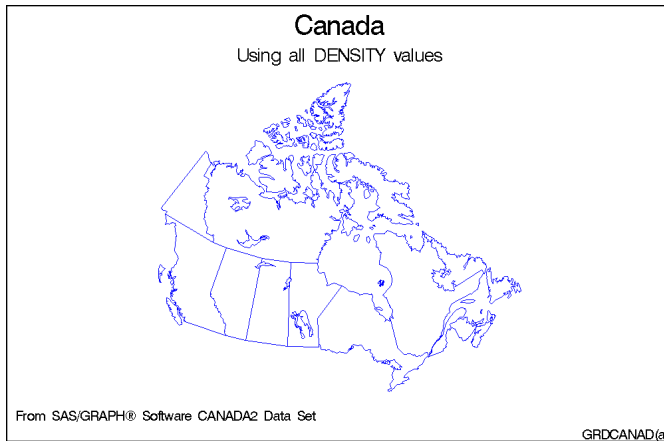
ID statement

Other features:

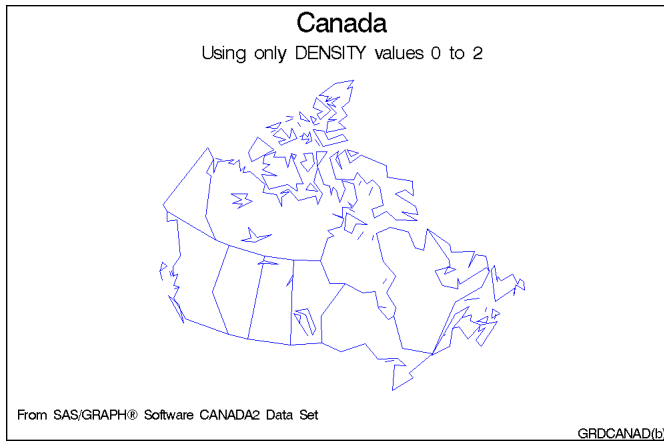
PROC GMAP option:

WHERE=

Sample library member: GRDCANAD



In this example, the GREDUCE procedure creates the DENSITY variable for the CANADA2 map data set that is provided with SAS/GRAPH. First, the map is displayed at its original density by using the GMAP procedure. Second, the map is displayed by using density values of 0 to 2.



Assign the libref and set the graphics environment.

```
libname maps 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(black blue green red)
          ftext=swiss htitle=6 htext=3;
```

Define titles and footnotes for the first map.

```
title1 'Canada';
title2 h=4 'Using all DENSITY values';
```

```

footnote1 j=1 ' From SAS/GRAPH
' '02'x
      ' Software CANADA2 Data Set';
footnote2 j=r 'GRDCANAD(a) ';
```

Define pattern characteristics.

```
pattern value=memory repeat=12 color=blue;
```

Show the unreduced map. The ID statement specifies the variable in the map data set that defines unit areas.

```

proc gmap map=maps.canada2 data=maps.canada2 all;
  id province;
  choro province / nolegend;
run;
```

The GREduce procedure creates a new map data set, CAN2, containing a DENSITY variable. The ID statement specifies the variable in the map data set that defines unit areas.

```

proc greduce data=maps.canada2 out=can2;
  id province;
run;
```

Define title and footnote for the second map.

```

title2 h=4 'Using only DENSITY values 0 to 2';
footnote2 j=r 'GRDCANAD(b) ';
```

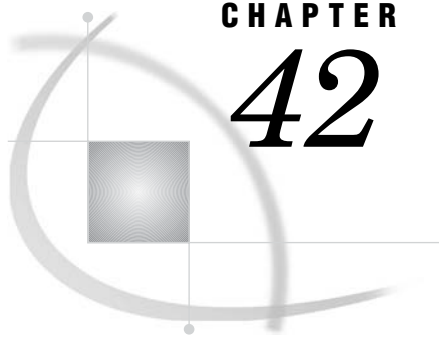
Show reduced map with density levels 0-2. WHERE= selects map coordinates with the appropriate DENSITY values.

```

proc gmap map=can2(where=(density<3))
  data=can2 all;
  id province;
  choro province / nolegend;
run;
quit;
```

References

Douglas, D.H. and Peucker, T.K. (1973), "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature," *The Canadian Cartographer*, 10, 112–122.



CHAPTER

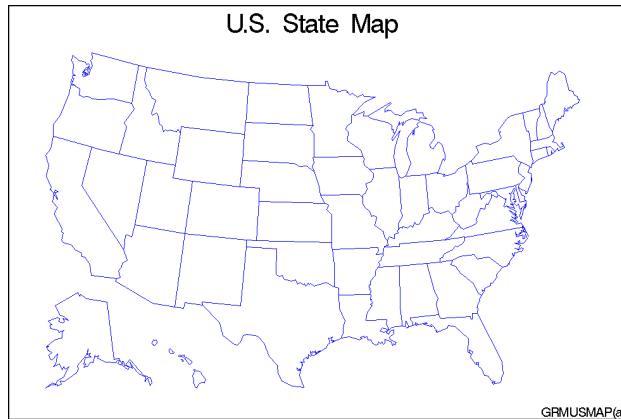
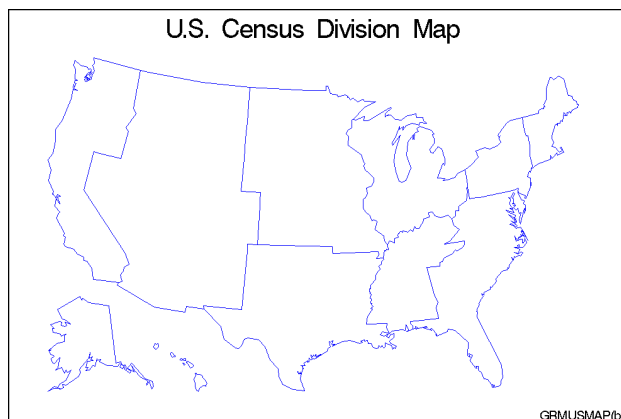
42

The GREMOVE Procedure

<i>Overview</i>	1223
<i>Concepts</i>	1224
<i>About the Input Map Data Set</i>	1224
<i>About the Output Map Data Set</i>	1225
<i>About Unmatched Area Boundaries</i>	1225
<i>Procedure Syntax</i>	1226
<i>PROC GREMOVE Statement</i>	1226
<i>BY Statement</i>	1227
<i>ID Statement</i>	1228
<i>Examples</i>	1228
<i>Example 1: Removing State Boundaries from U.S. Map</i>	1228
<i>Example 2: Creating an Outline Map of Africa</i>	1232

Overview

The GREMOVE procedure processes a map data set that is used as input. It does not produce any graphics output. Instead, it produces an output data set that typically becomes the input map data set for the GMAP procedure (see Chapter 35, “The GMAP Procedure,” on page 995). The GREMOVE procedure combines unit areas defined in a map data set into larger unit areas by removing shared borders between the original unit areas. For example, Figure 42.1 on page 1224 and Figure 42.2 on page 1224 show combined unit areas in a typical map data set by removing state boundaries to create regional census divisions.

Figure 42.1 Map before Removing Borders (GRMUSMAP(a))**Figure 42.2** Map after Removing Borders (GRMUSMAP(b))

The program for these maps is shown in Example 1 on page 1228.

Concepts

The GREMOVE procedure processes the input map data set to remove internal boundaries and creates a new map data set. The PROC GREMOVE statement identifies the input and output map data sets. The ID statement identifies the variable or variables in the input map data set that define the current unit areas. The BY statement identifies the variable or variables in the input map data set that define the new unit areas.

About the Input Map Data Set

The input map data set must be in traditional map data set format (see “About Map Data Sets” on page 999) and it must contain these variables:

- a numeric variable named X that contains the horizontal coordinates of the map boundary points.
- a numeric variable named Y that contains the vertical coordinates of the map boundary points.
- one or more variables that uniquely identify the current unit areas in the map. These variables are listed in the ID statement. Each group of observations with a different ID variable value is evaluated as a separate unit area.
- one or more variables that identify the new unit areas to be created in the output map data set. These variables are listed in the BY statement.

It may also contain the variable SEGMENT, which is used to distinguish non-conterminous segments of the same unit areas. Other variables may exist in the input map data set, but they do not affect the GREMOVE procedure and they will not be carried into the output map data set.

About the Output Map Data Set

The output map data set contains the newly defined unit areas. These new unit areas are created by removing all interior line segments from the original unit areas. All variables in the input map data set except X, Y, SEGMENT, and the variables listed in the BY statement are omitted from the output map data set.

The output map data set may contain missing X, Y coordinates to construct any polygons that have enclosed boundaries (like lakes or combined regions that have one or more hollow interior regions).

The SEGMENT variable in the output map data set is ordered according to the size of the bounding box around the polygon that it describes. A SEGMENT value of 1 describes the polygon whose bounding box is the largest in the unit area and so on. This information is useful for removing small polygons that clutter up maps.

All current unit areas with common BY-variable value(s) are combined into a single unit area in the output map data set. The new unit area contains

- all boundaries that are not shared, such as islands and lakes
- all boundaries that are shared by two different BY groups.

About Unmatched Area Boundaries

If you are using map data sets in which area boundaries do not match precisely (for example, if the boundaries were digitized with a different set of points), PROC GREMOVE will not be able to identify common boundaries properly, resulting in abnormalities in your output data set.

If the points in the area boundaries match up except for precision differences, before using PROC GREMOVE round each X and Y value in your map data set accordingly, using the DATA step function ROUND. See *SAS Language Reference: Dictionary* for information on the ROUND function.

For example, if you have a map data set named APPROX in which the horizontal and vertical coordinate values for interior boundaries of unit areas are exactly equal only to three decimal places, this DATA step creates a new map data set, EXACT, that is better suited for use with the GREMOVE procedure:

```
data exact;
  set approx;
  if x ne . then x=round(x,.001);
  if y ne . then y=round(y,.001);
run;
```

Procedure Syntax

Requirements: The BY and ID statements are required.

```
PROC GREMOVE <DATA=input-map-data-set>
  <OUT=output-map-data-set>;
BY <DESCENDING>variable-l
  <...<DESCENDING>variable-n>
  <NOTSORTED>;
ID variable(s);
```

PROC GREMOVE Statement

Identifies the input and output map data sets.

Requirements: An input map data set is required.

Syntax

```
PROC GREMOVE <DATA=input-map-data-set>
  <OUT=output-map-data-set>;
```

Options

DATA=input-map-data-set

specifies the map data set that is to be processed. By default, the procedure uses the most recently created SAS data set. The GREMOVE procedure expects the observations in the input map data set to be sorted in ascending order of the BY-variable values.

See also: “About the Input Map Data Set” on page 1224 and “SAS Data Sets” on page 29.

Featured in: Example 2 on page 1232.

OUT=output-data-set

names the new map data set, which contains the coordinates of the new unit areas created by the GREMOVE procedure. By default, the GREMOVE procedure names the new data set using the DATA n naming convention. That is, the procedure uses the name WORK.DATA n , where n is the next unused number in sequence. Thus, the first automatically named data set is DATA1, the second is DATA2, and so on.

See also: “About the Output Map Data Set” on page 1225.

Featured in: Example 2 on page 1232.

BY Statement

Lists the variable or variables that identify the new unit areas.

Requirements: At least one variable is required.

See also: “BY Statement” on page 141.

Featured in: Example 1 on page 1228.

Syntax

```
BY <DESCENDING>variable-l
    <...<DESCENDING>variable-n>
    <NOTSORTED>;
```

Required Arguments

variable(s)

identifies one or more variables in the input map data set that define the new unit areas. *Variable(s)* can be either numeric or character.

The BY variables in the input map data set become the ID variables for the output map data set.

Options

DESCENDING

indicates that the input map data set is sorted in descending order. By default, the GREMOVE procedure expects all BY-variable values to appear in ascending order.

This option affects only the variable that immediately follows the option.

NOTSORTED

indicates that observations with the same BY-variable values are to be grouped as they are encountered without regard for whether the values are in alphabetical or numerical order. NOTSORTED can appear anywhere in the BY statement. It affects all of the variables that are specified in the statement. NOTSORTED overrides DESCENDING if both appear in the same BY statement.

Ordering Observations

To sort the input map data set, use the SORT procedure in base SAS, for example

```
/* arrange the observations in desired order */
proc sort data=mapdata out=mapsort;
    by state;
run;
    /* remove the county boundaries */
proc gremove data=mapsort out=newmap;
    by state;
    id county;
run;
```

Notice that the GREMOVE procedure uses the same BY statement as the SORT procedure.

See the *Base SAS Procedures Guide* for further information on the SORT procedure.

Note: If an observation is encountered for which the BY-variable value is out of the proper order, the GREMOVE procedure stops and issues an error message. \triangle

ID Statement

Identifies the variable or variables that define the hierarchy of the current unit areas in the input map data set.

Requirements: At least one *id-variable* is required.

Featured in: Example 1 on page 1228.

Syntax

ID *id-variable(s)*;

Required Arguments

id-variable(s)

specifies one or more variables in the input map data set that identify the unit areas to be combined. These variables are not included in the output map data set.

Id-variable(s) can be either numeric or character.

See also: “About the Input Map Data Set” on page 1224.

Examples

The following examples illustrate major features of the GREMOVE procedure.

Example 1: Removing State Boundaries from U.S. Map

Procedure features:

BY statement

ID statement

Other features:

SORT procedure

MERGE procedure

LIBNAME statement

Sample library member: GRMUSMAP

This example processes the MAPS.US map data set, supplied with SAS/GRAPH, to produce a new map data set containing boundaries for the U.S. Bureau of the Census divisions. Because the MAPS.US map data set does not contain a variable to identify any unit area other than states, this example creates a map data set that contains the census divisions and that can be processed with the GREMOVE procedure.

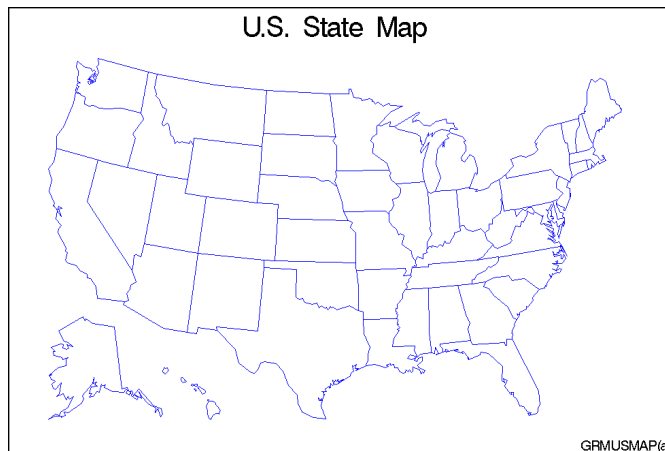
The STATE variable in the MAPS.US data set, containing numeric FIPS codes for each state, is used as the BY-variable to merge the CBSTATES and MAPS.US data sets. Output 42.1 shows the variables that are present in the data set before using the GREMOVE procedure:

Output 42.1 The MAPS.US Data Set

OBS	STATE	MAPS.US Data Set		X	Y
		SEGMENT			
1	1	1		0.16175	-0.10044
2	1	1		0.12305	-0.10415
3	1	1		0.12296	-0.10678
.					
.					
1524	56	1		-0.18757	0.15035
1525	56	1		-0.10158	0.13997
1526	56	1		-0.10398	0.11343

And Figure 42.3 on page 1229 shows the map before processing:

Figure 42.3 Map before Removing Borders (GRMUSMAP(a))

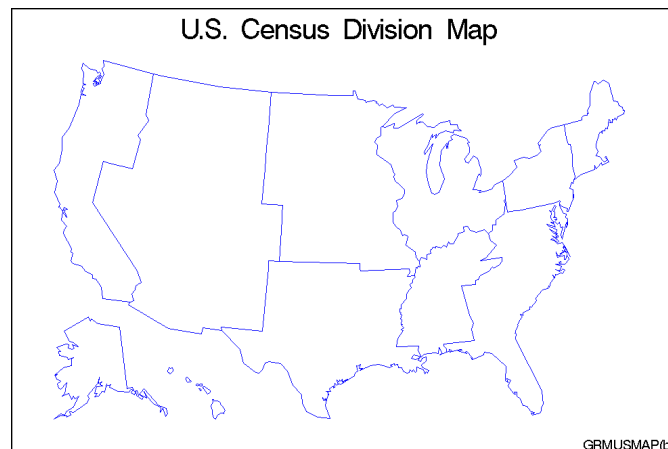


Output 42.2 shows the variables that are present in the data set after you use the GREMOVE procedure. Notice that the new map data set contains a new variable called DIVISION:

Output 42.2 The REMSTATE Data Set

OBS	REMSTATE Data Set		SEGMENT	DIVISION
	X	Y		
1	0.29825	0.17418	1	1
2	0.29814	0.17820	1	1
3	0.30206	0.18045	1	1
.				
.				
1082	-0.18715	-0.16010	8	9
1083	-0.18747	-0.15971	8	9
1084	-0.18747	-0.15951	8	9

Figure 42.4 on page 1230 shows the new map after PROC GREMOVE has removed interior state boundaries.

Figure 42.4 Map after Removing Borders (GRMUSMAP(b))

Assign the libref and set the graphics environment. If the libref MAPS is already assigned, omit the LIBNAME statement.

```
libname maps 'SAS-maps-library';
goptions reset=global gunit=pct border cback=white
          colors=(black blue green red)
          ftext=swiss htitle=6 htext=3;
```

Create data set CBSTATES. This data set includes a variable, DIVISION, that contains the number of the U.S. Bureau of the Census division for the state. This data step converts letter codes to numeric FIPS codes that match those in the STATE variable of MAPS.US.

```
data cbstates;
  length state 8 stcode $ 2 division 4;
  input stcode division;
```

```

state=stfips(stcode);
drop stcode;
datalines;
CT 1
MA 1
...more data lines...
OR 9
WA 9
;

```

Sort data set in FIPS-code order. Create a sorted data set, CBSORT. It can be properly match-merged with the MAPS.US map data set, which is already sorted in FIPS-code order.

```

proc sort data=cbstates out=cbsort;
  by state;
run;

```

Add DIVISION variable to map data set by merging the CBSORT data set with MAPS.US. Create a new map data set, USCB, that contains all of the state boundary coordinates from the MAPS.US data set plus the added variable DIVISION.

```

data uscb;
  merge cbsort maps.us;
  by state;
run;

```

Sort data set in DIVISION order. Sort USCB by the DIVISION variable to create the DIVSTATE data set.

```

proc sort data=uscb out=divstate;
  by division;
run;

```

Remove interior boundaries within divisions. BY specifies the variable, DIVISION, in the input map data set that identifies the new unit areas. ID specifies the variable, STATE, in the input map data set that identifies the current unit areas.

```

proc gremove data=divstate out=remstate;
  by division;
  id state;
run;

```

Define title and footnote for map.

```

title 'U.S. State Map';
footnote j=r 'GRMUSMAP(a) ';

```

Define pattern characteristics.

```
pattern value=mempty repeat=48 color=blue;
```

Show the original map.

```
proc gmap map=maps.us data=maps.us all;
  id state;
  choro state / nolegend;
run;
```

Define new title and footnote for map.

```
title 'U.S. Census Division Map';
footnote j=r 'GRMUSMAP(b) ';
```

Show the regional map. ID specifies the variable, DIVISION, that identifies the unit areas in the processed data set. CHORO specifies DIVISION as the response variable.

```
proc gmap map=remstate data=remstate all;
  id division;
  choro division / nolegend;
run;
quit;
```

Example 2: Creating an Outline Map of Africa

Procedure features:

PROC GREMOVE options:

DATA=
OUT=

Other features:

GMAP procedure

Sample library member: GRMAFRIC

This example processes the MAPS.AFRICA map data set, supplied with SAS/GRAPH, to produce a new map data set that contains no internal boundaries. This is done by adding a new variable, REGION, to the map data set and setting it equal to 1. Unit areas from the input map data set that have the same BY-variable value are combined into one unit area in the output map data set. Output 42.3 shows the variables present in the original map data set:

Output 42.3 The MAPS.AFRICA Data Set

OBS	MAPS.AFRICA Data Set			
	ID	SEGMENT	X	Y
1	125	1	0.57679	1.43730
2	125	1	0.57668	1.43467
3	125	1	0.58515	1.42363
.				
.				
3462	990	1	1.04249	0.50398
3463	990	1	1.04184	0.50713
3464	990	1	1.04286	0.50841

Figure 42.5 on page 1233 shows the map before processing:

Figure 42.5 Map before Removing Borders (GRMAFRIC(a))



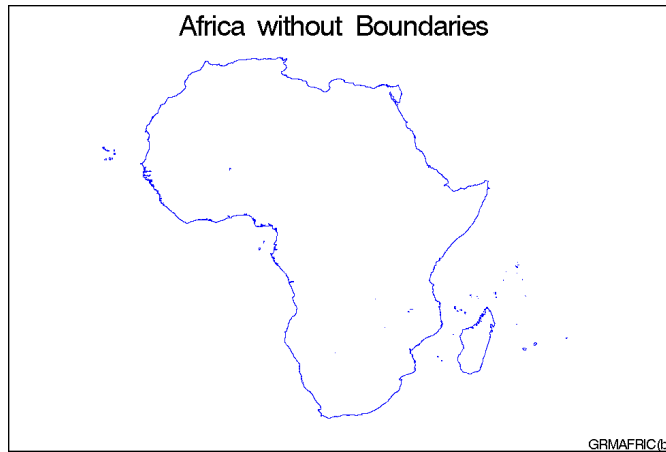
The new AFRICA map data set is created with a new variable, REGION. Output 42.4 shows the variables that are present in the new map data set created by the GREMOVE procedure:

Output 42.4 The AFRICA Data Set

OBS	AFRICA Data Set			
	X	Y	SEGMENT	REGION
1	0.24826	1.02167	1	1
2	0.25707	1.02714	1	1
3	0.26553	1.03752	1	1
.				
.				
982	1.19071	1.30043	3	1
983	1.18675	1.30842	3	1
984	1.18518	1.32822	3	1

Figure 42.6 on page 1234 shows the new map after PROC GREMOVE has removed all of the interior boundaries:

Figure 42.6 Map after Removing Borders (GRMAFRIC(b))



Assign the libref and set the graphics environment.

```
libname maps 'SAS-maps-library';
options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss htitle=6 htext=3;
```

Create the NEWAF data set. This new map data set contains all the variables in the SAS/GRAPH supplied MAPS.AFRICA map data set plus the added variable REGION.

```
data newaf;
  set maps.africa;
  region=1;
run;
```

Remove the unit areas from the AFRICA data set. DATA= specifies the input map data set and OUT= specifies the output map data set. The input map data set has a variable called REGION that is used as the BY-variable to identify the new unit areas. The ID statement specifies the current unit areas from the input map data set.

```
proc gremove data=newaf out=africa;
  by region;
  id id;
run;
```


Define the title and footnote.

```
title 'Africa with Boundaries';
footnote j=r 'GRMAFRIC(a) ';
```

Define pattern characteristics.

```
pattern value=mempty r=50 color=blue;
```

Display the original map.

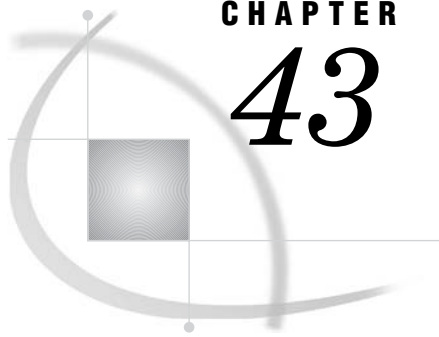
```
proc gmap map=maps.africa data=maps.africa all;
  id id;
  choro id / nolegend;
run;
```

Define a new title and footnote for the map.

```
title 'Africa without Boundaries';
footnote j=r 'GRMAFRIC(b) ';
```

Display the map with no boundaries. ID specifies the variable, REGION, that identifies the unit areas in the processed data set.

```
proc gmap data=africa map=africa;
  id region;
  choro region / nolegend;
run;
quit;
```

CHAPTER

43

The GREPLAY Procedure

<i>Overview</i>	1238
<i>Concepts</i>	1239
<i>About Catalog Entries</i>	1239
<i>Duplicate Entry Names</i>	1240
<i>Ways to Use the GREPLAY Procedure</i>	1241
<i>Windowing Environment</i>	1241
<i>Code-based Statements</i>	1241
<i>Procedure Syntax</i>	1242
<i>PROC GREPLAY Statement</i>	1243
<i>? Statement</i>	1246
<i>BYLINE Statement</i>	1247
<i>CC Statement</i>	1247
<i>CCOPY Statement</i>	1247
<i>CDEF Statement</i>	1248
<i>CDELETE Statement</i>	1249
<i>CMAP Statement</i>	1250
<i>COPY Statement</i>	1250
<i>DELETE Statement</i>	1251
<i>DEVICE Statement</i>	1251
<i>FS Statement</i>	1252
<i>GOUT Statement</i>	1252
<i>GROUP Statement</i>	1252
<i>IGOUT Statement</i>	1253
<i>LIST Statement</i>	1253
<i>MODIFY Statement</i>	1254
<i>MOVE Statement</i>	1255
<i>NOBYLINE Statement</i>	1256
<i>PREVIEW Statement</i>	1256
<i>QUIT Statement</i>	1256
<i>REPLAY Statement</i>	1257
<i>TC Statement</i>	1257
<i>TCOPY Statement</i>	1258
<i>TDEF Statement</i>	1259
<i>TDELETE Statement</i>	1262
<i>TEMPLATE Statement</i>	1262
<i>TREPLAY Statement</i>	1263
<i>Using the GREPLAY Procedure</i>	1264
<i>Using the GREPLAY Windows</i>	1264
<i>GREPLAY Window Commands</i>	1264
<i>PROC GREPLAY Window</i>	1265
<i>PRESENTATION Window</i>	1265

<i>DIRECTORY Window</i>	1265
<i>TEMPLATE DESIGN Window</i>	1265
<i>COLOR MAPPING Window</i>	1266
<i>Managing Catalog Entries</i>	1267
<i>Replaying Catalog Entries</i>	1268
<i>Creating Templates and Color Maps</i>	1268
<i>Replaying Graphics Output in a Template</i>	1270
<i>Examples</i>	1270
<i>Example 1: Creating a Template</i>	1270
<i>Example 2: Replaying Graphics Output in a Template</i>	1272
<i>Example 3: Creating a Color Map</i>	1274

Overview

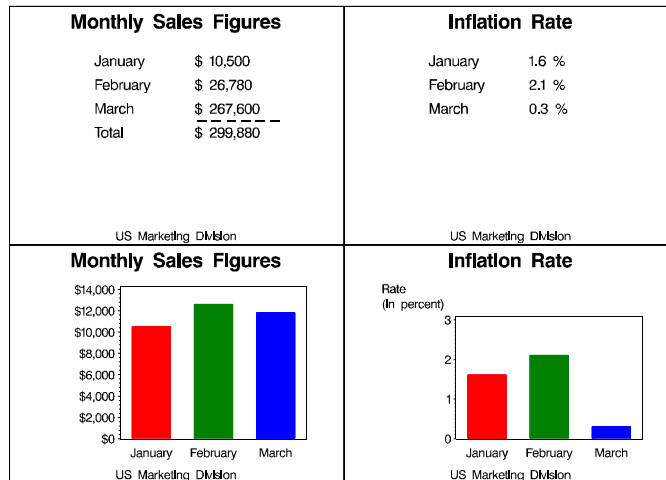
The GREPLAY procedure displays and manages graphics output that is stored in SAS catalogs. The GREPLAY procedure also creates templates and color maps that you can use when you replay your graphics output. The GREPLAY procedure operates in both windowing and line-mode environments.

With the GREPLAY procedure, you can

- select one or more catalog entries from the same catalog for replay and route them to your display or other devices, such as plotters and printers.
- use, create, or modify templates. You can use templates to describe positioning on a single display for the graphics output that is stored in one or more catalog entries.
- use, create, or modify color maps. Color maps enable you to change the colors in graphics output by mapping existing colors to new colors.
- manage entries in SAS catalogs by
 - creating logical groupings of catalog entries that contain graphics output
 - renaming, deleting, or copying catalog entries that contain graphics output, templates, and color maps
 - rearranging catalog entries that contain graphics output.
- create new graphics output by replaying one or more catalog entries into panels within a template.

Figure 43.1 on page 1239 shows four catalog entries that were replayed into a template and displayed as a single graph.

Figure 43.1 Graphics Output in a Template



For an example of replaying graphics in a template, see Example 2 on page 1272.

Concepts

About Catalog Entries

The GREPLAY procedure uses three kinds of catalog entries:

graphics output

catalog entries of type GRSEG. The catalog in which these entries are stored is referred to as the input-catalog and output-catalog. The *input-catalog* is the catalog that contains the graphics output, stored in catalog entries, that you want to replay. You can change the input catalog during a catalog management session. The *output-catalog* is the catalog in which graphics output that is produced by the template facility is stored. The output catalog is also the destination of copied catalog entries.

templates

catalog entries of type TEMPLATE. This catalog is referred to as the template-catalog. The *template-catalog* is the catalog that stores templates that are created by the GREPLAY procedure. The template catalog also may contain previously created templates that you want to modify or templates to use for replaying your graphics output. SASHELP.TEMPLT is the Institute-supplied template catalog.

color maps

catalog entries of type CMAP. This catalog is referred to as color-map-catalog. The *color-map-catalog* is the catalog that stores color maps that are created by the GREPLAY procedure. The color map catalog also may contain previously created color maps that you want to modify or color maps to use when you replay your graphics output. Note that image entries may exist in this catalog but are not recognized by the GREPLAY procedure.

Note: Image entries may exist in the catalog but are not recognized by PROC GREPLAY. \triangle

You can store all of the previous entry types in a single SAS catalog, or you can store them in separate catalogs and use a different catalog for each type of entry. A single SAS catalog may contain graphics output, color maps, and templates.

Because the GREPLAY procedure operates on catalog entries, you must assign at least one catalog before you can perform any tasks. The GREPLAY procedure has several ways to assign the catalogs, as shown in Table 43.1 on page 1240.

Table 43.1 Assigning Catalogs

Catalog	Ways to Assign
input	IGOUT= option in the PROC GREPLAY statement IGOUT statement IGOUT field in the PROC GREPLAY window
output	GOUT= option in the PROC GREPLAY statement GOUT statement GOUT field in the PROC GREPLAY window
template	TC= option in the PROC GREPLAY statement TC statement TC field in the PROC GREPLAY window
color	CC= option in the PROC GREPLAY statement
map	CC statement CC field in the PROC GREPLAY window

In addition, you can assign a current template, which you can use when you replay graphics output, and a current color map, which you can use to remap colors when you replay graphics output. To assign the current template, use one of the following:

- the TEMPLATE= option in the PROC GREPLAY statement
- the TEMPLATE statement
- the Template field in the PROC GREPLAY window.

To assign the current color map, use one of the following:

- the CMAP= option in the PROC GREPLAY statement
- the CMAP statement
- the Cmap field in the PROC GREPLAY window.

Duplicate Entry Names

If you try to create a catalog entry with the same name as an existing entry, the GREPLAY procedure uses the following naming conventions to prevent duplication of the name.

- For names that are fewer than eight characters, the procedure adds a number to the end of the name. For example, if you copy an entry that is named PLOT to a catalog that already contains an entry with that name, the procedure assigns the name PLOT1 to the new copy.
- For names that are eight characters long, the procedure drops the last character from the name before it adds the suffix. For example, if you copy an entry

TITLEONE to a catalog that already contains an entry with that name, the procedure assigns the name TITLEON1 to the copied entry.

- Template entries that contain individual entries will reserve the individual entry names as well as the template entry name.

The GREPLAY procedure uses the same technique for the names of entries that contain graphics output that is produced by the template facility.

Ways to Use the GREPLAY Procedure

You can use the GREPLAY procedure to replay or manage catalog entries in two different ways:

- by browsing or editing the fields in the GREPLAY procedure windows (see Figure 43.2 on page 1264)
- by submitting code-based GREPLAY procedure statements (see “Code-based Statements” on page 1241).

If you are running SAS software in a nonwindowing environment (such as line mode or batch), you can only submit code-based GREPLAY statements. However, if you are running SAS software in a windowing environment, you can use either the GREPLAY windows or the GREPLAY statements.

If your device supports a windowing environment, the GREPLAY procedure automatically opens the GREPLAY procedure windows. Otherwise, the GREPLAY procedure expects you to submit GREPLAY procedure statements.

Windowing Environment

To invoke the GREPLAY windows, submit the PROC GREPLAY statement without the NOFS option, as follows:

```
proc greplay;
run;
```

SAS/GRAPH then opens the PROC GREPLAY window. For more information, see “Using the GREPLAY Procedure” on page 1264.

If you are in a windowing environment, you can switch between the windows and code-based statements while you run the procedure. See the “FS Statement” on page 1252 and the NOFS window command in the SAS Help facility.

Code-based Statements

If you do not use the GREPLAY windows, you can use code-based statements to replay or manage the catalog entries. The GREPLAY procedure automatically uses code-based statements if you do not have a windowing device or if you are running the GREPLAY procedure in a batch environment. To use the GREPLAY procedure with code-based statements on a windowing device, submit the PROC GREPLAY statement with the NOFS option as follows:

```
proc greplay nofs;
```

Once you submit the PROC GREPLAY statement, you can enter and submit statements and run them without re-entering the PROC GREPLAY statement.

You can exit the GREPLAY procedure with code-based statements in two ways:

- submit the END, QUIT, or STOP statement
- submit another PROC statement or DATA step.

Procedure Syntax

Requirements: Use statements other than the PROC GREPLAY statement only in a nonwindowing or batch environment, or with the NOFS option. In these environments at least one additional statement is required.

Note: You must have write access to a catalog in order to modify, add, or delete device entries. Only GRSEG entry types may be replayed with the GREPLAY procedure.

Supports: RUN-group processing Output Delivery System (ODS)

```

PROC GREPLAY <BYLINE>
    <CC=color-map-catalog>
    <CMAP=color-map-entry>
    <FS>
    <GOUT=<libref.>output-catalog>
    <IGOUT=<libref.>input-catalog>
    <IMAGEMAP=output-data-set>
    <NOBYLINE>
    <NOFS>
    <PRESENTATION>
    <TC=template-catalog>
    <TEMPLATE=template-entry>;
? required -argument;
BYLINE;
CC color-map-catalog;
CCOPY <color-map-catalog.>color-map-entry<.CMAP>;
CDEF color-map-entry
    <color-definition(s)>
    <DES='entry-description'>;
CDELETE color-map-entry(s) | _ALL_;
CMAP color-map-entry;
COPY entry-id(s) | _ALL_;
DELETE entry-id(s) | _ALL_;
DEVICE device-name;
FS;
GOUT <libref.>output-catalog;
GROUP entry-id(s);
IGOUT <libref.>input-catalog ;
LIST required-argument;
MODIFY modify-pair(s);
MOVE entry-id-1 AFTER | BEFORE entry-id-2;
NOBYLINE;
PREVIEW template-entry(s) | _ALL_ ;
QUIT | END | STOP;
REPLAY entry-id(s) | _FIRST_ | _LAST_ | _ALL_ ;
TC template-catalog;
TCOPY <template-catalog.>template-entry<.TEMPLATE>;
TDEF template-entry

```



```

    <panel definition(s)>
    <DES='entry-description'>;
TDELETE template-entry(s) | _ALL_ ;
TEMPLATE template-entry;
TREPLAY select-pair(s);

```

PROC GREPLAY Statement

Determines whether the procedure starts in a windowing or nonwindowing environment, and whether the session is used for catalog management or output presentation.

Syntax

```

PROC GREPLAY <BYLINE>
    <CC=color-map-catalog>
    <CMAP=color-map-entry>
    <FS>
    <GOUT=<libref.>output-catalog>
    <IGOUT=<libref.>input-catalog>
    <IMAGEMAP=output-data-set>
    <NOBYLINE>
    <NOFS>
    <PRESENTATION>
    <TC=template-catalog>
    <TEMPLATE=template-entry>;

```

Options

Each PROC GREPLAY statement option has an equivalent statement that you can use instead.

BYLINE

specifies that the BY statement information for the SAS catalog entries should be displayed. The BY statement information appears directly beneath the primary description of the entry. By default, the BY statement information is displayed.

CC=*color-map-catalog*

identifies the color map catalog to be used with the GREPLAY procedure. Use the CMAP= option to assign a current color map that is contained in *color-map-catalog*.

To assign a current color map or create new color maps, you must assign a color map catalog with the CC= option.

To replay graphics output using a color map, you must assign a color map catalog and a current color map with the CC= and CMAP= options.

Featured in: Example 3 on page 1274.

CMAP=*color-map-entry*

assigns a current color map to use when replaying graphics output, where *color-map-entry* names an existing color map in the catalog specified in the CC= option. If *color-map-entry* is not in the catalog, an error message is written to the SAS log. *Color-map-entry* must have a catalog entry type of CMAP.

If you do not specify a color map catalog using the CC= option when using the CMAP= option, a warning message is written to the SAS log.

To replay graphics output using a color map, you must assign a color map catalog and a current color map with the CC= and CMAP= options.

FS

specifies that the GREPLAY procedure should use windows. By default, if your device supports windows, the GREPLAY procedure uses windows. If your device does not support windows, the procedure begins execution in line-mode and the FS option has no effect.

GOUT=<libref.>output-catalog

specifies the SAS catalog in which to save the graphics output that is produced by the GREPLAY procedure. In addition, catalog entries that contain graphics output can be copied to *output-catalog*. If you omit the libref, SAS/GRAPH looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist. *Output-catalog* can be the same catalog that is specified in the IGOUT= option.

To copy catalog entries, you must assign an input and, optionally, an output catalog with the IGOUT= and GOUT= options.

See also: “Storing Graphics Output in SAS Catalogs” on page 53

Featured in: Example 2 on page 1272.

IGOUT=<libref.>input-catalog

specifies the input catalog to use with the GREPLAY procedure. The input catalog that you specify with the IGOUT= option should be a catalog that contains the graphics output that will be replayed. If you omit the libref, SAS/GRAPH looks for the catalog in the temporary library called WORK. *Input-catalog* can be the same catalog that you specified in the GOUT= option.

To move, group, or delete catalog entries or to replay graphics output, you must assign an input catalog with the IGOUT= option.

To copy catalog entries, you must assign an input and, optionally, an output catalog with the IGOUT= and GOUT= options.

Featured in: Example 2 on page 1272.

IMAGEMAP=output-data-set

must be used in conjunction with the REPLAY statement (see “REPLAY Statement” on page 1257). The IMAGEMAP= option creates a temporary SAS data set that contains information about the graph that is replayed from the graphics catalog. The information in the image map data set includes the shape and coordinates of the elements in the graph, along with values that were associated with those elements in variables that were identified for that purpose in the HTML= and/or HTML_LEGEND= options. The image map data set can be used to generate an HTML image map in an HTML output file using the IMAGEMAP macro. The IMAGEMAP macro takes two arguments, the name of the image map data set and the name or fileref of an HTML output file, as shown in the following example:

```
%imagemap(imgmapds, myimgmap.html);
```

See also: “Adding Links with the HTML= and HTML_LEGEND= Options” on page 574

NOBYLINE

suppresses the BY statement information for the SAS catalog entries. The BY statement information appears directly beneath the primary description of the entry. By default, the BY statement information is displayed.

NOFS

specifies that the GREPLAY procedure should use line mode. By default, if your device supports windows, the GREPLAY procedure uses windows. If your device does not support windows, the procedure uses line mode, regardless of whether you used the FS option or the NOFS option.

Featured in: Example 1 on page 1270.

PRESENTATION

specifies that the GREPLAY procedure should open the PRESENTATION window and use the catalog specified by the IGOUT= option as the input catalog. The PRESENTATION option is often used in applications to prevent the application users from deleting or reordering the catalog entries. You can only replay graphics output from the PRESENTATION window; you cannot manage catalogs or create templates and color maps from this window.

You must use the IGOUT= option when you use the PRESENTATION option. The PRESENTATION option overrides the NOFS option on full-screen devices.

TC=*template-catalog*

identifies the template catalog to use with the GREPLAY procedure. Use the TEMPLATE= option to assign a current template from *template-catalog*.

To assign a current template or create new templates, you must assign a template catalog with the TC= option.

To replay graphics output in a template, you must assign a template catalog and a current template with the TC= and TEMPLATE= options.

Featured in: Example 1 on page 1270.

TEMPLATE=*template-entry*

assigns a current template to use when replaying graphics output where *template-entry* names an existing template in the template catalog that is specified in the TC= option. If *template-entry* is not in the catalog, an error message is written to the SAS log. *Template-entry* must have a catalog entry type of TEMPLATE.

When you use the TEMPLATE= option, you must also specify the name of a template catalog with the TC= option. Otherwise, a warning message is written to the SAS log.

Featured in: Example 2 on page 1272.

Details

When you submit the PROC GREPLAY statement, the mode of operation depends on both the environment in which the statement is submitted and whether the NOFS option is included, as shown in Table 43.2 on page 1245.

Table 43.2 Ways of Invoking the GREPLAY Procedure

Environment	Statement	Result
windowing	PROC GREPLAY;	GREPLAY procedure windows
windowing	PROC GREPLAY NOFS;	line mode
nonwindowing	PROC GREPLAY;	line mode

You can switch back and forth between windows and line-mode within a session.

? Statement

Prints the current value of certain PROC GREPLAY options or of the current device driver.

Procedure output: Output is sent to the SAS log.

Syntax

? *required-argument*;

required-argument must be one of the following:

CC

CMAP

DEVICE

GOUT

IGOUT

TC

TEMPLATE

Required Arguments

CC

prints the name of the current color map catalog. If no color map catalog has been assigned, the GREPLAY procedure issues a message.

CMAP

prints the name of the current color map. If no color map has been assigned, the GREPLAY procedure issues a message.

DEVICE

DEV

prints the name of the current device driver.

GOUT

prints the name of the current output catalog. If you did not assign an output catalog, the GREPLAY procedure issues a message.

IGOUT

prints the name of the current input catalog. If you did not assign an input catalog, the GREPLAY procedure issues a message.

TC

prints the name of the current template catalog. If you did not assign a template catalog, the GREPLAY procedure issues a message.

TEMPLATE

prints the name of the current template. If you did not assign a template, the GREPLAY procedure issues a message.

BYLINE Statement

Displays BY statement information directly beneath the primary description of the catalog entries when you list the contents of the input catalog.

Note: BY statement information is displayed by default.

See also: NOBYLINE statement

Syntax

```
BYLINE;
```

CC Statement

Specifies a color map catalog and allows you to change the color map catalog without exiting the procedure.

Syntax

```
CC color-map-catalog;
```

Required Arguments

color-map-catalog

identifies the SAS catalog where color maps should be stored or the name of a SAS catalog containing color maps.

CCOPY Statement

Copies a color map from another catalog to the color map catalog or creates a duplicate copy of a color map within the color map catalog.

Requirements: Assign a color map catalog before using the CCOPY statement.

See also: CC statement

Syntax

```
CCOPY <color-map-catalog.>color-map-entry<.CMAP>;
```

Required Arguments

<color-map-catalog.>color-map-entry <.CMAP>

identifies the color map entry to be copied.

color-map-catalog

is the SAS catalog that contains the color map to be copied.

color-map-entry

is the name of the entry color map.

CMAP

is the catalog entry type.

If a color map of the same name already exists in the color map catalog, the GREPLAY procedure creates a new name.

See also: “Duplicate Entry Names” on page 1240

Details

To copy a color map from another catalog to the color map catalog, use the CC statement to specify *color-map-catalog* as the catalog from which the color map should be copied. For example, the following statements copy HP.CMAP from the catalog named ONE.CCAT to the catalog named TARGET.CLRMAP:

```
libname target 'SAS-data-library';
libname one 'SAS-data-library';

proc greplay nofs;
  cc target.clrmap;
  ccopy one.ccat.hp.cmap;
quit;
```

To create a duplicate copy of a color map, simply omit *color-map-catalog* from your CCOPY statement. For example, to create a duplicate copy of the color map named HP.CMAP in the color map catalog, use the following statement:

```
ccopy hp.cmap;
```

CDEF Statement

Defines or modifies a color map in the color map catalog.

Requirements: Assign a color map catalog before using the CDEF statement.

See also: CC statement

Featured in: Example 3 on page 1274

Syntax

```
CDEF color-map-entry
      <color-definition(s)>
      <DES='entry-description'>;
```

color-definition has the following form:

color-number / from-color:to-color

color-definition has the following form: *color-number / from-color:to-color*

Required Arguments

color-map-entry

identifies an existing or new color map. *Color-map-entry* is the name of a catalog entry.

If the color map name is not in the color map catalog, then the procedure creates a new color map. If the color map name is already in the color map catalog, then the procedure modifies or adds to that color map.

Options

color-number / from-color:to-color

specifies a color pair and how it is defined.

color-number

specifies the number of a color pair.

from-color:to-color

defines the colors that are being mapped:

from-color

is the color to be mapped.

to-color

is the new color that replaces *from-color* in the replayed graphics output.

DES='entry-description'

specifies a description of the catalog entry for the color map. The maximum length for the *entry-description* is 256 characters. By default, the GREPLAY procedure assigns a description of ***** NEW COLOR MAP ***** to the color map.

CDELETE Statement

Deletes one or more color maps from the current color map catalog.

Caution: The GREPLAY procedure does not prompt you to confirm your request to delete color maps.

Alias: CDEL

Syntax

CDELETE *color-map-entry(s) | _ALL_ ;*

Required Arguments

color-map-entry(s)

identifies one or more color maps that you want to delete from the color map catalog. You can submit a single entry or a list of entries in one CDELETE statement.

ALL

deletes all of the color maps from the color map catalog.

CMAP Statement

Assigns the current color map to be used when replaying graphics output.

Requirements: Assign a color map catalog before using the CMAP statement.

See also: CC statement

Featured in: Example 3 on page 1274

Syntax

CMAP *color-map-entry*;

Required Arguments

color-map-entry

identifies an existing color map, contained in the color map catalog, to use when replaying your graphics output. If the color map is not in the current color map catalog, the GREPLAY procedure issues an error message in the SAS log.

COPY Statement

Copies one or more catalog entries containing graphics output from the input catalog to the output catalog.

Requirements: Assign an input catalog and an output catalog before using the COPY statement.

Note: You cannot use the COPY statement to create a duplicate of an entry containing graphics output in the same catalog. You can have only one copy of an entry containing graphics output in a catalog.

See also: GOUT and IGOUT statements

Syntax

COPY *entry-id(s)* | ALL ;

Required Arguments

One of the following is required:

entry-id(s)

is the number or name of a catalog entry, or the number or name of a group of entries to be copied from the input catalog to the output catalog. Entries must contain graphics output. Multiple *entry-id(s)* can contain both numbers and names.

ALL

copies all of the graphics output entries in the input catalog to the output catalog.

DELETE Statement

Deletes SAS catalog entries containing graphics output from the current input catalog.

Caution: The GREPLAY procedure does not prompt you to confirm your request to delete an entry containing graphics output.

Alias: DEL

Syntax

```
DELETE entry-id(s) | ALL ;
```

Required Arguments

One of the following is required:

entry-id(s)

is the number or name of a catalog entry, or the number or name of a group of entries to be deleted from the input catalog. Entries must contain graphics output. Multiple *entry-id(s)* can contain both numbers and names.

ALL

deletes all of the graphics output entries in the input catalog.

DEVICE Statement

Specifies the device driver.

Requirements: You must specify a device driver that your graphics device can support and that is available in your SAS session.

Alias: DEV

Syntax

```
DEVICE device-name;
```

Required Arguments

device-name

specifies the device driver to use when you replay graphics output. The device driver that you specify becomes the current device and is used for subsequent replays until you submit another DEVICE statement or change the device driver in another way.

FS Statement

Switches from line mode to the GREPLAY procedure windows.

Requirements: Your device must support windows.

See also: NOFS on page 1244

Syntax

FS;

GOUT Statement

Assigns the current output catalog used by the GREPLAY procedure.

Note: You may change the output catalog without exiting the procedure by using the GOUT statement.

Syntax

GOUT <libref.>output-catalog;

Required Arguments**<libref.>output-catalog**

identifies the SAS catalog that you want to use as an output catalog. By default, the output catalog is WORK.GSEG.

GROUP Statement

Creates groups of entries in the current input catalog.

Syntax

GROUP entry-id(s);

Required Arguments

entry-id(s)

is the number or name of a catalog entry that contains graphics output. All of the entries that are specified in the GROUP statement are included in a single group with a group header. You can submit a single entry or a list of entries with a single GROUP statement. A list of entries can contain both entry numbers and entry names.

Details

You can manage and display groups of entries with the DELETE, COPY, and REPLAY statements in the same way that you manage single entries.

Only one group can be created per group statement. The default name for a group header is GROUP. The default description for the group header is `*** new group ***`. The GREPLAY procedure uses a naming convention to avoid duplicate names. See “Duplicate Entry Names” on page 1240 for more information on the naming convention.

To change the name (and description) of a group, use the MODIFY statement.

IGOUT Statement

Assigns the current input catalog used by the GREPLAY procedure.

Note: You may change the input catalog without exiting the procedure by using the IGOUT statement.

Syntax

```
IGOUT <libref.>input-catalog;
```

Required Arguments

<libref.>input-catalog

identifies the SAS catalog with entries that contain graphics output that you want to replay.

LIST Statement

Prints entries in the input, template, and color map catalogs, as well as the contents of templates and color maps.

Procedure output: The output from the LIST statement is sent to the SAS log.

Note: Entries are listed in the order of their creation date.

Featured in: Example 3 on page 1274

Syntax

```
LIST required-argument;
```

required-argument must be one of the following:

CC
 CMAP
 IGOUT
 TC
 TEMPLATE

Required Arguments

One of the following is required:

CC

prints the color maps that are in the current color map catalog. If the catalog contains both templates and color maps, only color maps are listed.

CMAP

prints the *From* and *To* color values in the current color map.

IGOUT

prints the number, names, and descriptions of the entries in the input catalog that contains graphics output. In addition, the type of graphics output (dependent or independent) is shown.

TC

prints the templates in the current template catalog. If the catalog contains both templates and color maps, only the templates are listed.

TEMPLATE

prints the panel definition values of the current template.

MODIFY Statement

Changes the name, description, and BY statement information of entries or group headers in the input catalog.

Syntax

MODIFY *modify-pair(s)*;

modify-pair(s) has the following form:

entry-id / entry-description(s)

Required Arguments

entry-id / entry-description(s)

specifies the entry to modify.

entry-id

is the number or name of a catalog entry, or the number or name of a group of entries in the input catalog. Entries must contain graphics output. Multiple *entry-id(s)* can contain both numbers and names.

entry-description(s)

must be at least one of the following:

BYLINE=*'character-string'*

specifies a character string that can be used for additional information or for BY statement information. *Character-string* can be up to 40 characters long and must be enclosed in quotation marks. BY statement information appears directly beneath the primary description of the catalog entry.

NAME=*'entry-name'*

specifies the new name of the catalog entry for the graph. The maximum length for *entry-name* is eight characters. If the specified name duplicates the name of an existing entry, SAS/GRAPH software adds a number to the duplicate name to create a unique entry.

Note: The value for *entry-name* can be either with or without quotation marks. △

DES=*'entry-description'*

specifies the description of the catalog entry for the graph. The maximum length for *entry-description* is 256 characters. The description does not appear on the graph.

MOVE Statement

Rearranges entries in the input catalog by moving entries either before or after other entries.

Syntax

```
MOVE entry-id-1 AFTER | BEFORE entry-id-2;
```

Required Arguments

entry-id-1

is the name or number of an existing catalog entry or a group header that is to be moved.

entry-id-2

is the name or number of an existing catalog entry or a group header. *Entry-id-1* can be placed before or after *entry-id-2*.

AFTER | BEFORE

specifies whether *entry-id-1* should be moved before or after *entry-id-2*.

Details

To move an entire group, use the name of the group for *entry-id-1*. To move an entry into a group, move the entry after a group header or before or after an entry in the group. For example, this statement moves the entry CHART3 into the group that is named NEW_SALES:

```
move chart3 after new_sales;
```

NOBYLINE Statement

Suppresses BY statement information.

Note: By default, the BY statement information is displayed.

See also: BYLINE statement

Syntax

NOBYLINE;

PREVIEW Statement

Displays the panel outlines for one or more templates using the current device. Use the TC statement to specify the template catalog before using the PREVIEW statement.

Tip: When you preview a list of templates, press END or ENTER to preview the next template in the list.

Note: The graphics output produced when you preview a template is stored in a catalog named WORK.GTEM, which is deleted at the end of your session.

Syntax

PREVIEW *template-entry(s)* | ALL ;

Required Arguments

One of the following is required:

template-entry(s)

identifies one or more template entries that are contained in the current template catalog. You can preview one entry or a list of entries with one PREVIEW statement.

ALL

previews all of the templates in the current template catalog.

QUIT Statement

Exits the GREPLAY procedure.

Aliases: END, STOP

Syntax

QUIT;

REPLAY Statement

Selects one or more entries for replay from the current input catalog.

Note: If any entries specified in a REPLAY statement are not found in the input catalog, PROC GREPLAY issues a message in the SAS log and continues to replay valid entries.

Alias: PLAY

Syntax

```
REPLAY entry-id(s) | _FIRST_ | _LAST_ | _ALL_ ;
```

Required Arguments

One of the following is required:

entry-id(s)

is the number or name of a catalog entry, or the number or name of a group of entries in the input catalog. Entries must contain graphics output. Multiple *entry-id(s)* can contain both numbers and names. For example, this statement specifies both the entry named GRAPH and the third entry in the catalog:

```
replay graph 3;
```

ALL

replays all of the entries in the input catalog.

FIRST

replays the first entry in the input catalog.

LAST

replays the last entry in the input catalog.

TC Statement

Specifies the template catalog for the GREPLAY procedure.

Note: SASHELP.TEMPLT is the Institute-supplied template catalog.

Tip: Use the TC statement to change the template catalog without exiting the procedure.

Syntax

```
TC template-catalog;
```

Required Arguments

template-catalog

identifies the SAS catalog where templates are to be stored or identifies the name of a SAS catalog that contains templates.

TCOPY Statement

Copies templates from another catalog to the template catalog or creates a duplicate copy of a template within the template catalog.

Requirements: Assign a template catalog before using the TCOY statement.

See also: TC statement

Syntax

```
TCOPY <template-catalog.>template-entry<.TEMPLATE>;
```

Required Arguments

<template-catalog.>template-entry<.TEMPLATE>

identifies the template entry that is to be copied.

template-catalog

is the SAS catalog that contains the template that is to be copied.

template-entry

is the template entry name.

TEMPLATE

is the catalog entry type. If a template of the same name already exists in the template catalog, the GREPLAY procedure creates a new name.

See also: “Duplicate Entry Names” on page 1240

Details

To copy a template from another catalog to the template catalog, specify *template-catalog* as the catalog from which the template should be copied. For example, if you want to copy NEWTEMP.TEMPLATE from the catalog named ONE.TEMPLT to the catalog named TARGET.TEMPLT, use the following statements:

```
libname target 'SAS-data-library';
libname one 'SAS-data-library';

proc greplay nofs;
  tc target.templt;
  tcopy one.templt.newtemp.template;
quit;
```

To create a duplicate copy of a template, simply omit *template-catalog* from your TCOY statement. For example, to create a duplicate copy of a template named NEWTEMP within the template catalog, you could use the following statement:


```
tcopy newtemp.template;
```

TDEF Statement

Defines or modifies templates in the current template catalog.

Requirements: Assign a template catalog before using the TDEF statement.

See also: TC statement

Featured in: Example 1 on page 1270

Syntax

```
TDEF template-entry
      <panel-definition(s)>
      <DES='entry-description'>;
```

panel-definition has the following form:

```
panel-number / <panel-option(s)>
```

panel-option(s) can be one or more of the following:

```
CLIP
COLOR=border-color
COPY=panel-number
DEF
DELETE
LLX=x
LLY=y
LRX=x
LRY=y
ROTATE=degrees
SCALEX=factor
SCALEY=factor
ULX=x
ULY=y
URX=x
URY=y
XLATEX=distance
XLATEY=distance
```

Required Arguments

template-entry

identifies an existing or a new template. If the template is not in the template catalog, the procedure creates it. If the template is already in the template catalog, the procedure modifies or makes additions to that template.

Only *template-entry* is required, but if you specify only the template name without any options, no changes are made to an existing template and no new template is created.

Options

CLIP

specifies that any panels behind this panel should be clipped. If clipping is in effect for a panel, only the graphics output that is to be placed in that panel can appear in the space that the panel occupies, unless a previous panel occupies all or part of that space.

COLOR=*border-color*

specifies the color of the panel border. If you omit *border-color*, then no border is displayed around the panel when you replay graphics output in the panel. If you preview a template that contains a panel without a border color, the GREPLAY procedure uses the first color in the colors list as the outline for the border.

COPY=*panel-number*

specifies the number of the panel definition that is to be copied to this panel.

DEF

specifies a default panel. A default panel has the following characteristics:

Panel Corner	Coordinates
lower left	(0,0)
upper left	(0,100)
upper right	(100,100)
lower right	(100,0)

DELETE

DEL

deletes the panel.

DES=*'entry-description'*

specifies the description of the catalog entry for the template. The maximum length for *entry-description* is 256 characters. By default, the procedure uses *** new template *** for the description.

LLX=*x*

specifies the X coordinate of the lower-left corner of the panel. Units for *x* are percentage of the graphics output area.

LLY=*y*

specifies the Y coordinate of the lower-left corner of the panel. Units for *y* are percentage of the graphics output area.

LRX=*x*

specifies the X coordinate of the lower-right corner of the panel. Units for *x* are percentage of the graphics output area.

LRY=*y*

specifies the Y coordinate of the lower-right corner of the panel. Units for *y* are percentage of the graphics output area.

panel-number

identifies the number of the panel that is being defined or modified.

ROTATE=*degrees*

specifies the rotation angle for the panel. The coordinates of the panel corners are automatically adjusted.

SCALEX=*factor*

specifies the scale factor for the X coordinates in the panel. You can use this scale factor to increase or decrease the size of the panel in the X direction or to reverse the X coordinates for the panel.

SCALEY=*factor*

specifies the scale factor for Y coordinates in the panel. You can use this scale factor to increase or decrease the size of the panel in the Y direction or to reverse the Y coordinates for the panel.

ULX=*x*

specifies the X coordinate of the upper-left corner of the panel. Units for *x* are percentage of the graphics output area.

ULY=*y*

specifies the Y coordinate of the upper-left corner of the panel. Units for *y* are percentage of the graphics output area.

URX=*x*

specifies the X coordinate of the upper-right corner of the panel. Units for *x* are percentage of the graphics output area.

URY=*y*

specifies the Y coordinate of the upper-right corner of the panel. Units for *y* are percentage of the graphics output area.

XLATEX=*distance*

specifies the distance to move the X coordinates of the panel. Units for *distance* are percentage of the graphics output area.

XLATEY=*distance*

specifies the distance to move the Y coordinates of the panel. Units for *distance* are percentage of the graphics output area.

Details

Use coordinate values that are less than 0 and greater than 100 in the LLX=, LLY=, LRX=, LRY=, ULX=, ULY=, URX=, and URY= options to zoom in on the graphics output. That is, you can see only that part of the replayed graphics output in the range from 0 to 100 percent of the graphics output area.

The values that you supply for the SCALEX= and SCALEY= options are used to change the size and orientation of the panel. The scale factors are used for the corresponding X and Y coordinates of the panel. For example, if you specify

```
scalex=.5
scaley=2
```

the X coordinates are scaled to half the original size, and the Y coordinates are scaled to twice the original size.

If you supply a scale factor of 0, all of the coordinates are set to the same value. If you use a scale factor of 1, nothing happens. If you use a scale factor greater than 1, the values of the coordinates are increased and hence the size of the panel increases. If you use a scale factor less than 1 but greater than 0, the values of the coordinates are decreased and hence the size of the panel decreases. If you use a negative scale factor, the coordinates are reversed and hence the panel (and any graphics output replayed in the panel) is reversed.

TDELETE Statement

Deletes templates from the template catalog.

Caution: The GREPLAY procedure does not prompt you to confirm your request to delete templates.

Alias: TDEL

Syntax

```
TDELETE template-entry(s) | ALL ;
```

Required Arguments

One of the following is required:

template-entry(s)

identifies a template that is to be deleted from the template catalog. You can submit a single entry or a list of entries in a single TDELETE statement.

ALL

deletes all of the templates in the template catalog.

TEMPLATE Statement

Assigns a current template to use when replaying graphics output.

Requirements: Assign a template catalog before using the TEMPLATE statement.

Note: If you specify a template that is not in the current template catalog or if you specify a template before you have assigned a template catalog, the GREPLAY procedure issues an error message.

Featured in: Example 1 on page 1270

Syntax

```
TEMPLATE template-entry;
```

Required Arguments

template-entry

identifies an existing template to use when replaying graphics output. Use the TREPLAY statement to replay graphics output in the template.

TREPLAY Statement

Copies one or more entries from the graphics input catalog into a new entry in the graphics output catalog, using positioning information provided by the current template.

Requirements: Before issuing the TREPLAY statement, first specify a graphics input catalog with the “IGOUT Statement” on page 1253, assign a template catalog with the “TC Statement” on page 1257, and choose a template with the “TEMPLATE Statement” on page 1262.

Alias: TPLAY

Featured in: Example 2 on page 1272

Syntax

```
TREPLAY select-pairs<DES="entry-description" NAME="entry-name">;
```

select-pairs are of the following form:

```
template-panel-number1:entry-id1 <...template-panel-numberN:entry-idN>
```

Required Arguments

template-panel-number:entry-id

specifies the panel number and the name of a catalog entry.

template-panel-number

determines the position of the graph in the new entry in the graphics output catalog, based on the position of the specified panel in the current template.

entry-id

specifies the name or number of the entry in the graphics input catalog that is to be added to the new entry in the graphics output catalog.

Options

DES="entry-description"

adds a description to the new entry in the graphics output catalog. The description is truncated after 40 characters.

NAME="entry-name"

names the new entry in the graphics output catalog. The name must begin with a letter and continue with up to seven more letters, numbers, or underscores. If the NAME= option is not specified, the new entry is named TEMPLATE by default. If an entry named TEMPLATE already exists in the graphics output catalog, the new entry is named TEMPLATx, where x is a unique number. This naming convention also applies if the value of the NAME= option already exists in the graphics output catalog.

Details

When you replay existing GRSEG entries in a template, the GREPLAY procedure creates new graphics output that is stored in the output catalog.

You can replay as many entries as you want in a single TREPLAY statement as shown here:

```
treplay 1:plot1 2:plot2 3:chart1;
```

PLOT1 will be placed in panel 1 of the current template, PLOT2 will be placed in panel 2, and CHART1 will be placed in panel 3. You can use entry numbers in the place of entry names.

Using the GREPLAY Procedure

Using the GREPLAY Windows

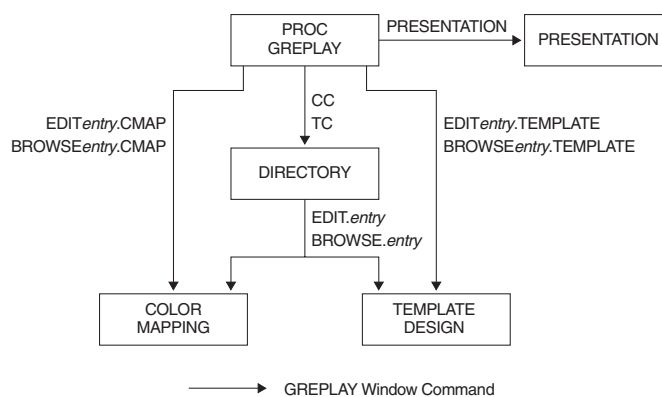
You can use the GREPLAY windows instead of code-based statements to replay and manage catalog entries. You perform tasks that use the GREPLAY procedure windows by entering values in the fields that are displayed in the windows and by issuing commands from the command line.

There are five GREPLAY windows:

- PROC GREPLAY window
- PRESENTATION window
- DIRECTORY window
- TEMPLATE DESIGN window
- COLOR MAPPING window.

Figure 43.2 on page 1264 shows how these windows relate to each other. Each window can be scrolled forward or backward as needed to display additional fields and information.

Figure 43.2 GREPLAY Procedure Windows



This section briefly describes the GREPLAY windows; for a complete description of each window and its fields, refer to the SAS Help facility.

GREPLAY Window Commands

You can navigate and manipulate the GREPLAY windows by entering commands on the command line or by selecting them from the menus. For a complete description of all the GREPLAY window commands, refer to the SAS Help facility.

PROC GREPLAY Window

This window is the first to appear when you submit the PROC GREPLAY statement on a full-screen device without the PRESENTATION or NOFS option. You can use it to both replay graphics output and to manage catalog entries that contain graphics output.

Display 43.1 The PROC GREPLAY Window

The screenshot shows a window titled "PROC GREPLAY" with a menu bar (File, Edit, View, Tools, Help). Below the menu bar, there are fields for ICOUT: REFLIB.EXCAT, TC: (with a cursor), GC: (with a cursor), GOUT: (with a cursor), Template: (with a cursor), Cmap: (with a cursor), Device: HP7475, and Scroll: PAGE. Below these fields is a table with columns: Sel, Name, Type, Description, and Created.

Sel	Name	Type	Description	Created
___	GR23N01B	I	CHOROPLETH MAP OF STATE	03/24/98
___	GR23N02	I	CHOROPLETH MAP OF STATE	03/24/98
___	GR23N03	I	CHOROPLETH MAP OF STATE	03/24/98
___	GR23N04	I	CHOROPLETH MAP OF STATE	03/24/98
___	GR21N01	I	Bubble of dollars * eng = num	03/24/98
___	GR21N02	I	Bubble of dollars * eng = num	03/24/98
___	GR21N03	I	Bubble of dollars * eng = num	03/24/98
___	GR21N04A	I	Plot of height * weight	03/24/98
___	GR21N04B	I	Plot of height * weight	03/24/98
___	GR21N05	I	Plot of high * year	03/24/98
___	GR21N06	I	Plot of high * year	03/24/98
___	GR21N07	I	Plot of low * year	03/24/98
___	GR21N08A	I	Plot of faren * month = city	03/24/98

PRESENTATION Window

This window is a modified version of the PROC GREPLAY window that enables you to replay graphics output while it prevents you from deleting entries or changing templates and color maps. Once you have created and organized your catalog, you may want to use the PRESENTATION window in an application for replaying graphics output.

DIRECTORY Window

This window lists the names of the catalog entries, gives a brief description of each, and indicates the date on which each entry was created or last changed. Although all catalog entry types are displayed in the DIRECTORY window, you can manage only entries of type CMAP or TEMPLATE from this window.

Display 43.2 The DIRECTORY Window

The screenshot shows a window titled "GREPLAY: DIRECTORY MYLIB.TEMPCAT (E)" with a menu bar (File, Edit, View, Tools, Help). Below the menu bar, there is a single entry: NEWTEMP TEMPLATE Five panel template 09/23/97.

■	NEWTEMP	TEMPLATE	Five panel template	09/23/97
---	---------	----------	---------------------	----------

TEMPLATE DESIGN Window

This window lets you design templates that you can use to present graphics output. You design a template by specifying the coordinates of its panels and determining the order in which the template panels are filled. Once you enter coordinates for a panel, you can alter them easily by using the Scale, Xlate (translate), and Rotate utility fields. These utility fields recalculate coordinate values automatically.

Table 43.3 GREPLAY Procedure Windows

If you are in the...	and you want to open the...	then...
PROC GREPLAY statement	PROC GREPLAY window	Submit the PROC GREPLAY statement without using the PRESENTATION or NOFS options.
	PRESENTATION window	Submit the PROC GREPLAY statement and include the PRESENTATION and IGOUT= options.
PROC GREPLAY window	PRESENTATION window	Specify a catalog and issue the PRES command.
	DIRECTORY window	Specify a template catalog and issue the TC command. OR Specify a color map catalog and issue the CC command.
	TEMPLATE DESIGN window	Specify a template catalog and issue the following command: edit <i>template-name</i> .template
	COLOR MAPPING window	Specify a color map catalog and issue the following command: edit <i>color-map-name</i> .cmap
	DIRECTORY window	Place an S beside the name of an existing template. OR Issue the following command: edit <i>template-name</i> .template
DIRECTORRY window	TEMPLATE DESIGN window	Place an S beside the name of an existing template. OR Issue the following command: edit <i>template-name</i> .template
	COLOR MAPPING window	Place an S beside the name of an existing color map. OR Issue the following command: edit <i>color-map-name</i> .cmap

Managing Catalog Entries

You can replay entries or perform a variety of catalog management tasks with GREPLAY code-based statements. Table 43.4 on page 1268 lists several common tasks and the statements that you use to perform them.

Table 43.4 Ways of Performing Common GREPLAY Procedure Tasks

Task	Code-based Statement
copy graphics output (GRSEGs) from the input catalog to the output catalog*	COPY statement
arrange GRSEG entries into logical groupings	GROUP statement
reorder the entries	MOVE statement
delete unneeded GRSEG entries	DELETE statement
change entry names and descriptions of entries in the input catalog	MODIFY statement
replay an entry from the input catalog	REPLAY statement
replay an entry in a template panel	TREPLAY statement

*You must assign an output catalog before copying graphics output.

Replaying Catalog Entries

To select catalog entries for replay, first assign an input catalog that contains the graphics output that is to be replayed. Then assign the entry with the REPLAY statement.

For example, the following statements replay the GRSEG entry named GRAPH1 from the catalog MYLIB.GRAPHs, which is assigned with the IGOUT= option:

```
libname mylib 'SAS-data-library';

proc greplay igout=mylib.graphs nofs;
  replay graph1;
run;
quit;
```

If you do not specify an output catalog with the GOUT= option in the PROC statement of the SAS/GRAPH procedure that creates the graphics output, the graphics output is automatically stored in the WORK.GSEG catalog. Replay the graphics output that is stored in this catalog as follows:

```
proc greplay nofs;
  igout work.gseg;
  replay _all_;
run;
quit;
```

Creating Templates and Color Maps

You can use the GREPLAY procedure to create templates and color maps. You can use templates to replay graphics output from several catalog entries on a single display,

or to change the shape or size of graphics output. You can use color maps to remap colors when replaying graphics output.

A color map is a list of up to 256 pairs of colors that enables you to change the colors in graphics output by mapping the original colors to a list of new colors. Color maps are useful for controlling how colors that are not available on the current device are remapped.

When you assign a current color map and replay graphics output that is stored in a catalog entry, any color in your graphics output that appears in the From column of the color map is mapped to the corresponding color in the To column of the color map. The new colors are not saved with the graphics output, and you do not create new graphics output when you use a color map to replay graphics output.

Table 43.5 Ways of Performing Common GREPLAY Procedure Tasks

Task	In line mode
assign a color map catalog	CC statement
copy a color map from another catalog	CCOPY statement
define or modify a color map in the current catalog	CDEF statement
assign a color map to use when you replay graphics output	CMAP statement
delete unneeded entries	DELETE statement
assign a template catalog	TC statement
copy a template from another catalog	TCOPY statement
delete a template	TDELETE statement
define a template	TDEF statement
display the panel outlines for a template	PREVIEW statement
assign a template to use when you replay graphics output	TEMPLATE statement
replay an entry in a template panel	TREPLAY statement

Before you create a template, you must assign a template catalog. If you are use the GREPLAY procedure in line mode, use the TDEF statement to define a template and the PREVIEW statement to preview a template. For example, the following statements define and preview a template named `TEMPLT`:

```
tdef templt 1/def;
preview templt;
```

Before you create a color map, you must assign a color map catalog. If you use the GREPLAY procedure in line mode, use the CDEF statement to define a color map. For example, the following statement defines a color map named `CLRMAP`:

```
cdef clrmap 1 / cyan : blue;
```

Replaying Graphics Output in a Template

You can use the GREPLAY procedure to create new graphics output by replaying existing graphics output in templates. You can create your own templates, or you can use the templates that are provided with SAS/GRAPH software and stored in the catalog SASHELP.TEMPLT. To view the templates that are provided with SAS/GRAPH, open the SAS Explorer window, and click the TEMPLT folder under SASHELP library.

Before you can replay graphics output in a template, you must assign a template catalog and a current template, as well as an input catalog. Then assign the entries to the template with the TREPLAY statement.

For example, the following statements replay the entries GRAPH1 and GRAPH2 into the V2 template, which is stored in the catalog SASHELP.TEMPLT. The TC statement specifies the catalog that contains the template, and the TEMPLATE statement specifies the template. The TREPLAY statement assigns each entry to a panel. (The V2 template has two panels, so there is an assignment for panel 1 and panel 2.)

```
proc greplay igout=mylib.graphs nofs;
  tc sashelp.templt;
  template v2;
  treplay 1:graph1 2:graph2;
run;
```

When you replay graphics output in a template, the new graphics output that is created by the GREPLAY procedure is automatically provided a default name and is stored in the output catalog, WORK.GSEG.

Templates are often used to describe positioning for replaying graphics output from several catalog entries on a single display.

Examples

The following examples illustrate major features of the GREPLAY procedure.

Example 1: Creating a Template

Procedure features:

GREPLAY statement options:

NOFS

TC=

TDEF statement

TEMPLATE statement

Sample library member: GRECRTM1

This example creates a template with five panels. Four of the panels are small and equal in size. The fifth panel is a large, full-size panel that can be used later to display a common title or footnote for the entire template (see). In this example, the LIST statement displays the template contents in the log. Output 43.1 shows the template definition that is written to the log file. The template that is defined here is also used in Example 2 on page 1272.

Set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss htitle=6 htext=3
```

Start the GREPLAY procedure. NOFS starts the procedure in line-mode. TC=assigns TEMPCAT as the template catalog.

```
proc greplay tc=reflib.tempcat nofs;
```

Define a template with four panels. The TDEF statement defines a template named NEWTEMP and places it in the previously defined template catalog. Each definition identifies the panel number and specifies the coordinates of the four corners. The panels are arranged within the template as follows: panel 1 is lower left; panel 2 is upper left; panel 3 is upper right; panel 4 is lower right; panel 5 is the full size so that it can contain a common title and footnote for all the template entries. COLOR= draws a border for each panel in the specified color.

```
tdef newtemp des='Five panel template'

1/llx=0    lly=10
   ulx=0    uly=50
   urx=50   ury=50
   lrx=50   lry=10
   color=blue

2/llx=0    lly=50
   ulx=0    uly=90
   urx=50   ury=90
   lrx=50   lry=50
   color=red

3/llx=50   lly=50
   ulx=50   uly=90
   urx=100  ury=90
   lrx=100  lry=50
   color=green

4/llx=50   lly=10
   ulx=50   uly=50
   urx=100  ury=50
   lrx=100  lry=10
   color=cyan

5/llx=0    lly=0
   ulx=0    uly=100
   urx=100  ury=100
   lrx=100  lry=0
   color=lipk;
```

Assign the current template. The TEMPLATE statement assigns the newly created template NEWTEMP as the current template.

```
template newtemp;
```

Write the contents of the current template to the log.

```
list template;
quit;
```

Output 43.1 Defining a Template (GRECRTM1)

```
.
.
.
64      /* list contents of current template */
65      list template;

NEWTEMP      Five panel template

Pan Clp Color      Ll-x Ll-y Ul-x Ul-y Ur-x Ur-y Lr-x Lr-y
1      BLUE      0.0 10.0 0.0 50.0 50.0 50.0 50.0 10.0
2      RED      0.0 50.0 0.0 90.0 50.0 90.0 50.0 50.0
3      GREEN      50.0 50.0 50.0 90.0 100.0 90.0 100.0 50.0
4      CYAN      50.0 10.0 50.0 50.0 100.0 50.0 100.0 10.0
5      LIPK      0.0 0.0 0.0 100.0 100.0 100.0 100.0 0.0

66      quit;
.
.
.
```

Example 2: Replaying Graphics Output in a Template

Procedure features:

GREPLAY statement options:

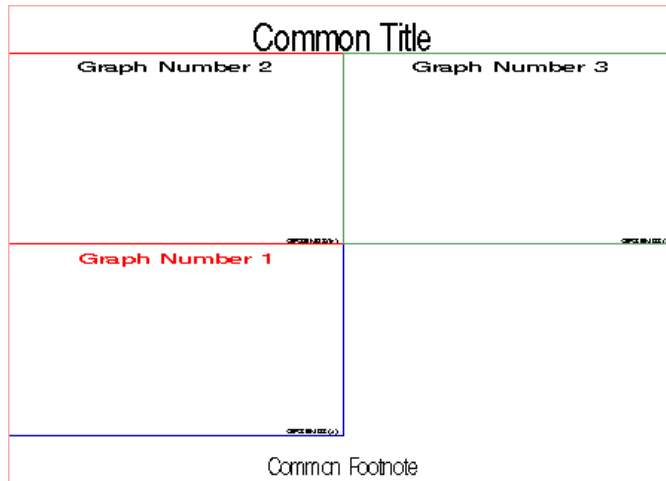
```
GOUT=
IGOUT=
TEMPLATE=
```

TREPLAY statement

Other features:

```
PROC GSLIDE
```

Sample library member: GRERGOT1



The TREPLAY statement replays into the template NEWTEMP four catalog entries that contain graphics output. The NEWTEMP template is defined in Example 1 on page 1270. It contains four equally sized panels and one large, full-size panel. Note that assignments are made to all but one of the panels. Because the fourth panel is not listed in the TREPLAY statement, it does not appear in the graphics output. The HSIZE= and VSIZE= options are adjusted and then reset to default in order to reflect the dimensions of the different-sized template panels.

Set the graphics environment. HSIZE= and VSIZE= are set for the dimensions of template panels 1, 2, 3, and 4.

```
goptions gunit=pct border
         cback=white colors=(black blue green red)
         ftext=swiss htitle=8 htext=5
         hsize=4in vsize=3.8in;
```

Generate three graphs in the permanent catalog GRAFCAT. The GSLIDE procedure creates three text slides and stores them in GRAFCAT as specified by the GOUT= option. By default, these are stored as GSLIDE, GSLIDE1, and GSLIDE2.

```
proc gslide gout=grafcat;
  title c=red 'Graph Number 1';
  footnote h=3 j=r 'GRERGOT1(a) ';
run;
  title 'Graph Number 2';
  footnote h=3 j=r 'GRERGOT1(b) ';
run;
  title 'Graph Number 3';
  footnote h=3 j=r 'GRERGOT1(c) ';
run;
```

Reset HSIZE= and VSIZE= to the default values and generate a text slide with PROC GSLIDE. Resetting the HSIZE and VSIZE values enables you to create a text slide with the proper aspect ratio for use in template panel 5.

```

goptions hsize=0in vsize=0in;
proc gslide gout=grafcat;
    title 'Common Title';
    footnote 'Common Footnote';
run;

```

Start the GREPLAY procedure. IGOUT= assigns GRAFCAT as the input catalog. GOUT= assigns EXCAT as the output catalog. TEMPLATE= assigns NEWTEMP as the current template. Remember, the NEWTEMP template is defined in Example 1.

```

proc greplay igout=grafcat gout=excat
    tc=tempcat nofs;
    template=newtemp;

```

Write the contents of the current template to the log.

```
list template;
```

Replay three graphs into template. The TREPLAY statement assigns three entries to panels in the NEWTEMP template. Each assignment is a panel number and the name of a graphics output entry. Names are the default names assigned by the GSLIDE procedure.

```

treplay 1:gslide
        2:gslide1
        3:gslide2
        5:gslide3;
quit;

```

Example 3: Creating a Color Map

Procedure features:

GREPLAY statement option:

CC=

GOUT=

CDEF statement

CMAP statement

LIST statement

Sample library member: GRECRCM1

This example uses the CDEF statement to define a color map. The LIST statement is used in this example to display the color map definition in the log. Output 43.2 shows a partial listing of the log.

Set the graphics environment.


```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swissb htitle=6 htext=3;
```

Start the GREPLAY procedure. CC= assigns CLRMAP as the color map catalog. GOUT= specifies the permanent catalog in which to place the graphics output.

```
proc greplay cc=clrmap gout=excat nofs;
```

Define a color map. The CDEF statement defines a color map named MYCOLOR that contains three color pairs.

```
cdef mycolor des='Special Color Map'
      1 / pink   : red
      2 / cyan   : blue
      3 / lig    : green;
```

Specify current color map and write contents to the log. The CMAP statement assigns MYCOLOR as the current color map. The contents of CMAP are listed in the log.

```
cmap mycolor;
list cmap;
quit;
```

Output 43.2 Defining a Color Map (GRERCM1)

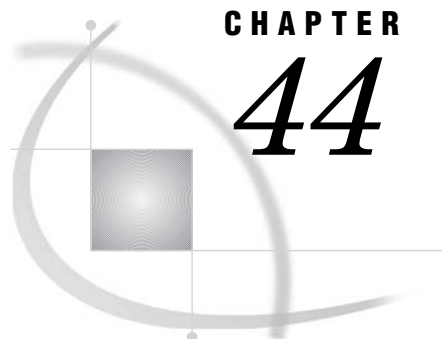
```
.
.
75      /* list the contents of the color map */
76      list cmap;

MYCOLOR      Special Color Map

      FROM      TO

1   PINK      RED
2   CYAN      BLUE
3   LIG       GREEN

77 quit;
.
```

CHAPTER

44

The GSLIDE Procedure

<i>Overview</i>	1277
<i>About Text Slides</i>	1277
<i>About Annotate Output</i>	1278
<i>Procedure Syntax</i>	1278
<i>PROC GSLIDE Statement</i>	1279
<i>Examples</i>	1282
<i>Example 1: Producing Text Slides</i>	1282
<i>Example 2: Displaying Annotate Graphics</i>	1283

Overview

The GSLIDE procedure is useful for creating text slides for presentations. You can overlay text slides on other graphics output with the GREPLAY procedure. The GSLIDE procedure produces graphics output that consists of text and straight lines that are generated by TITLE, FOOTNOTE, and NOTE statements. In addition, the procedure provides an easy way to add titles, notes, and footnotes to output that is produced entirely with an Annotate data set.

About Text Slides

Text slides contain text and graphics that are generated by SAS/GRAPH statements. To display an external text file as graphics output, use the GPRINT procedure.

Figure 44.1 on page 1277 shows a slide containing text that was produced with TITLE, FOOTNOTE, and NOTE statements.

Figure 44.1 Text Slide Produced by the GSLIDE Procedure (GSLTEXTS)



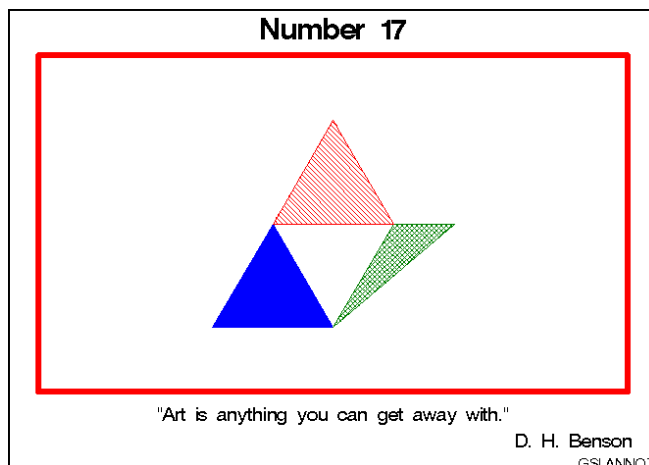
The program for this slide is in Example 1 on page 1282.

About Annotate Output

Annotate output is generated by commands that are stored in an Annotate data set. Use the GSLIDE procedure to display Annotate output when you want to include TITLE and FOOTNOTE statements on the output and use certain graphics options such as BORDER. To display Annotate graphics without these, use the GANNO procedure. See Chapter 24, "Using Annotate Data Sets," on page 587 for more information on creating and displaying Annotate data sets.

Figure 44.2 on page 1278 shows output from an Annotate data set that is displayed with titles and footnotes that were generated by TITLE and FOOTNOTE statements.

Figure 44.2 Output from an Annotate Data Set Displayed with the GSLIDE Procedure (GSLANNOT)



The program for this slide is in Example 2 on page 1283.

Procedure Syntax

Requirements: At least one of these is required: a TITLE, FOOTNOTE, or NOTE statement; an appearance option; the BORDER graphics option.

Global statements: FOOTNOTE, TITLE

Reminder: The procedure can include the SAS/GRAPH NOTE statement.

Supports: RUN-group processing Output Delivery System (ODS)

PROC GSLIDE <option(s)>;

PROC *GSLIDE* Statement

Creates a text slide. Optionally, it provides a border, specifies annotation, and assigns an output catalog. This is the only statement in the procedure.

Syntax

PROC *GSLIDE* *<option(s)>*;

option(s) can be one or more options from any or all of the following categories:

- appearance options:
 - ANNOTATE=*Annotate-data-set*
 - BORDER
 - CFRAME=*frame-color*
 - FRAME
 - IFRAME= *fileref* | '*external-file*'
 - IMAGESTYLE = TILE | FIT
 - LFRAME=*line-type*
 - WFRAME=*n*
- description options:
 - DESCRIPTION='entry-description'
 - GOUT=<*libref.*>*output-catalog*
 - NAME='entry-name'
- HTML option:
 - <IMAGEMAP=*output-data-set*>

Options

You can specify as many options as you want and list them in any order.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set that includes Annotate variables that identify graphics commands and parameters.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587.

Featured in: Example 2 on page 1283.

BORDER

draws a border around the graphics output area, which includes the title area, the footnote area, and the procedure output area. A color specification for the border is searched for in the following order:

- 1 the CTITLE= option in a GOPTIONS statement
- 2 the CTEXT= option in a GOPTIONS statement
- 3 the default, the first color in the colors list.

See also: “Adding Frames, Borders, and Images” on page 1281.

Featured in: Example 1 on page 1282.

CFRAME=*frame-color*

draws a frame around the procedure output area in the specified color. If you use both the CFRAME= and FRAME options, FRAME is ignored. If you use the IFRAME= option, the specified image fills the background of the slide.

Note: CFRAME= does not color the background of the slide. \triangle

See also: “Adding Frames, Borders, and Images” on page 1281.

Featured in: Example 1 on page 1282.

DESCRIPTION=*entry-description*

DES=*entry-description*

specifies the description of the catalog entry for the chart. The maximum length for *entry-description* is 256 characters. The description does not appear on the chart. By default, the GSLIDE procedure assigns the description OUTPUT FROM PROC GSLIDE.

FRAME

draws a frame around the procedure output area. By default, the frame color is the first color in the colors list. If you want to specify a different color for the frame, use the CFRAME= option instead. The FRAME option is overridden by the IFRAME= option, which fills the backplane frame with an image.

See also: “Adding Frames, Borders, and Images” on page 1281.

GOUT=<libref.>output-catalog

specifies the SAS catalog in which to save the graphics output produced by the GSLIDE procedure. If you omit the libref, SAS/GRAPH looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist.

See also: “Storing Graphics Output in SAS Catalogs” on page 53.

IFRAME=*fileref* | '*external-file*'

identifies the image file you wish to apply to the backplane of the plot. See also the IMAGESTYLE= option. The IFRAME= option is overridden by the NOIMAGEPRINT option.

IMAGEMAP=*output-data-set*

creates a temporary SAS data set that is used to generate an image map in an HTML output file. The information in the image map data set includes the shape and coordinates of the elements in the graph and drill-down URLs that have been associated with those elements. The drill-down URLs are provided by one or two variables in the input data set. These variables are identified to the GSLIDE procedure with the HTML= and/or HTML_LEGEND= options.

The %IMAGEMAP macro generates the image map in the HTML output file. The macro takes two arguments, the name of the image map data set and the name or fileref of the HTML output file, as shown in the following example:

```
%imagemap(imgmapds, myimgmap.html);
```

See also: “Adding Links with the HTML= and HTML_LEGEND= Options” on page 574 and “HTML Variable” on page 651.

IMAGESTYLE= TILE | FIT

specifies whether to tile the image to fill the backplane or to stretch the image to fit the backplane. The TILE value is the default. See also the IFRAME= option.

LFRAME=*line-type*

specifies the line type for a frame and draws a frame around the procedure output area. Values for *line-type* are 1 through 46. Line types are shown in Figure 7.22 on page 208. By default, LFRAME=1, which produces a solid line.

NAME=*entry-name*

specifies the name of the catalog entry for the graph. The maximum length for *entry-name* is eight characters. The default name is *GSLIDE*. If the specified name duplicates the name of an existing entry, SAS/GRAPH software adds a number to the duplicate name to create a unique entry, for example, *GSLIDE1*.

WFRAME=*n*

specifies the width of the frame where *n* is a number. The thickness of the frame increases directly with *n*, but the thickness of the line may vary from device to device. By default, *WFRAME=1*, which is the thinnest line. The *WFRAME=* option also draws the frame.

See also: “Adding Frames, Borders, and Images” on page 1281.

Featured in: Example 1 on page 1282.

Adding Frames, Borders, and Images

Like the *BORDER* option in a *GOPTIONS* statement, the *BORDER* option in the *PROC GSLIDE* statement draws a box around the graphics output area. However, the border generated by the *GSLIDE* procedure remains in effect only for the duration of the procedure.

Both *BORDER* options use the color specified by the *CTITLE=* or *CTEXT=* graphics option if either of these options is used; otherwise, the border color is the first color in the colors list.

While the *BORDER* option draws a box around the graphics output area, the *FRAME* option draws a box or frame around the procedure output area. In this case, titles and footnotes are outside of the frame. (See “Procedure Output and the Graphics Output Area” on page 34 for a description of the procedure output area.) Use *FRAME* to draw a frame in the default color, line type, and width. Otherwise, use one or more of the *CFRAME=*, *LFRAME=*, or *WFRAME=* options.

You can specify a colored frame with the *CFRAME=* option. Note that *CFRAME=* does not fill the procedure output area with color. However, you can use the *CBACK=* graphics option to provide a background color for the graphics output area. You can specify the type of line for the frame with the *LFRAME=* option and the width of the frame with the *WFRAME=* option.

You can also use the *IFRAME=* option to fill the background of your slide with an image. If an image is specified, it completely fills the background of the slide, obscuring any frame or border specifications.

Using Data-Dependent Coordinates

If you use the *GSLIDE* procedure with Annotate data sets that contain data-dependent coordinates, the resulting coordinate values may exceed the range of 0 to 100 used by the graphics output area, and some of the output may not be displayed. In this case, use the *GANNO* procedure, which can scale the output to fit the available space. See also Chapter 26, “The *GANNO* Procedure,” on page 707 for details .

Using RUN Groups

Although the *GSLIDE* procedure has no action statements, it can use *RUN*-group processing to display all currently defined titles, footnotes, and notes, as well as specified annotation, each time you submit a *RUN* statement. *TITLE* and *FOOTNOTE* statements that are defined while the *GSLIDE* procedure is active remain in effect after the procedure ends. *NOTE* definitions remain in effect until the *GSLIDE* procedure ends, at which time they are canceled. To cancel *NOTE* definitions while the procedure is active, specify *RESET=NOTE* in a *GOPTIONS* statement or submit a null *NOTE* statement. See “*RUN*-Group Processing” on page 33 for details.

Examples

Example 1: Producing Text Slides

Procedure features:

PROC GSLIDE options:

BORDER
CFRAME=
WFRAME=

Other features:

NOTE statement

Sample library member: GSLTEXTS



This example uses FOOTNOTE, NOTE, and TITLE statements to produce a text slide. PROC GSLIDE statement options add both a border and a frame.

Set the graphics environment.

```
goptions reset=global gunit=pct cback=blue
          colors=(white) ftext=swissb htitle=12 htext=4;
```

Define titles and footnotes.

```
title color=pink 'New Directions';
footnote1 j=1 ' ABC Engineering, Inc';
footnote2 j=1 ' January 1998'
          j=r h=3 f=swiss 'GSLTEXTS ';
```


Generate the slide and define additional text. BORDER draws a box around the entire graphics output area. CFRAME= draws a red box around the procedure output area. WFRAME= specifies the thickness of the frame. The first NOTE statement, which has no text, simply leaves a large blank line above the text specified by the second NOTE statement. The second JUSTIFY= causes a line break.

```
proc gslide border
      cframe=red
      wframe=4;
  note height=20;
  note height=10
      justify=center 'Goals and strategies'
      justify=center 'for the coming year';
run;
quit;
```

Example 2: Displaying Annotate Graphics

Procedure features:

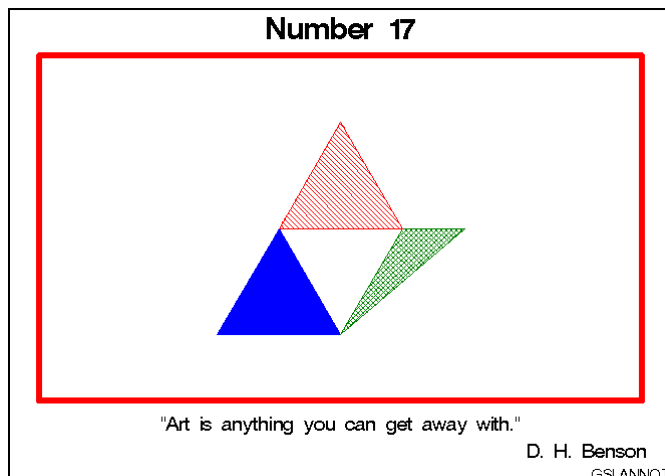
PROC GSLIDE option:

ANNOTATE=

Other features:

Annotate data set

Sample library member: `GSLANNOT`



In this example, the GSLIDE procedure displays Annotate graphics along with current TITLE and FOOTNOTE definitions. See Chapter 24, "Using Annotate Data Sets," on page 587 for information on creating Annotate data sets.

Set the graphics environment.

```
goptions reset=global gunit=pct cback=white
      colors=(black blue green red)
```

```
ftitle=swissb htitle=6 ftext=swiss htext=3;
```

Create the Annotate data set, ART. ART contains the commands that draw the design of triangles.

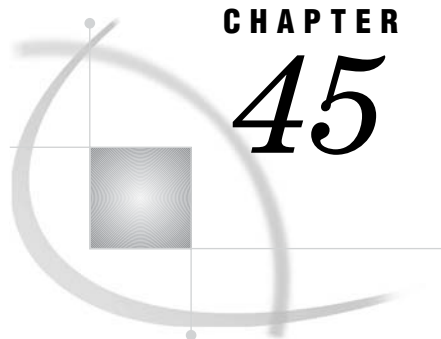
```
data art;
  length function color style $ 8;
  input function $ x y color $ style $;
  xsys='5'; ysys='5';
  datalines;
poly      30 20 blue    solid
polycont  50 20 .      .
polycont  40 50 .      .
poly      50 20 green   xl
polycont  70 50 .      .
polycont  60 50 .      .
poly      40 50 red     ll
polycont  60 50 .      .
polycont  50 80 .      .
;
```

Define title and footnotes displayed by the procedure. FOOTNOTE statements 4 and 5 have no text and are angled vertically to add space on the left and right sides between the border of the output and the frame that surrounds the procedure output area.

```
title 'Number 17';
footnote1 h=4 '"Art is anything you can get away with.'';
footnote2 j=r h=4 'D. H. Benson  ';
footnote3 j=r 'GSLANNOT  ';
footnote4 h=3 angle=90;
footnote5 h=3 angle=-90;
```

Display the annotate graphics on the slide with the title and footnotes. The GSLIDE procedure displays the graphics elements drawn by the commands in the Annotate data set specified by the ANNOTATE= option.

```
proc gslide annotate=art
  border
  wframe=6
  cframe=red;
run;
quit;
```



CHAPTER

45

The GTESTIT Procedure

<i>Overview</i>	1285
<i>About the Pictures</i>	1286
<i>About the LOG</i>	1289
<i>Procedure Syntax</i>	1290
<i>PROC GTESTIT Statement</i>	1290
<i>Examples</i>	1291
<i>Example 1: Testing a GOPTIONS Statement</i>	1291
<i>Example 2: Displaying, Changing and Verifying the Colors List for a Device Driver</i>	1292

Overview

The GTESTIT procedure is a diagnostic tool for testing the installation of SAS/GRAPH software and the configuration of your device. Use the GTESTIT procedure when you want to

- test a new device
- test the settings of a device driver that you are developing
- identify the colors and some of the SAS/GRAPH lines and fills for your device
- review some of your current settings of device parameters and graphics options
- test changes in settings of device parameters and graphics options.

The GTESTIT procedure produces three pictures that help you determine the configuration of your graphics device and graphics options and parameters. Refer to “About the Pictures” on page 1286 for examples of the pictures. Although it does not show the settings of all device parameters and graphics options, the GTESTIT procedure does show some of the most commonly used ones.

If you use a GOPTIONS statement to change one or more graphics options for the current SAS session, or if you run the GDEVICE procedure to change the parameter settings for your device, you can use the GTESTIT procedure to confirm that those changes took effect.

For example, if you use the GOPTIONS statement to set HPOS=45 and COLORS=(RED GREEN), you can display picture 1 in the GTESTIT procedure to confirm that the graphics output area is divided into 45 columns and that foreground colors have been limited to red and green.

See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261, Chapter 31, “The GDEVICE Procedure,” on page 915, and Chapter 3, “Device Drivers,” on page 41 for more information on setting graphics options and device parameters.

About the Pictures

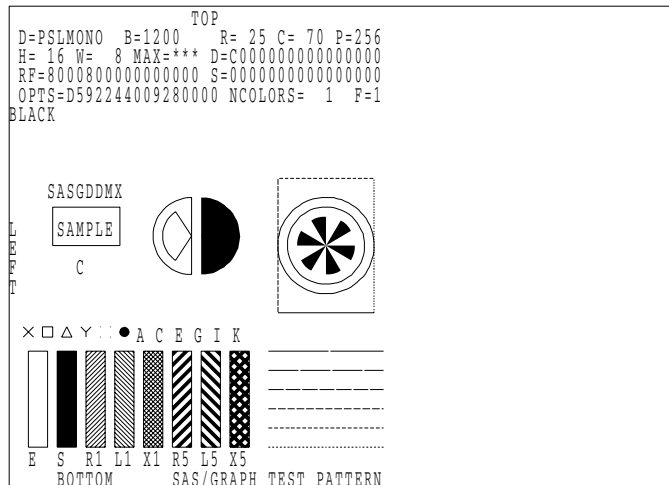
Figure 45.1 on page 1286 shows a test pattern and gives the values of some of the device settings that are currently in effect. Table 45.1 on page 1288 describes the graphics options and device parameters that are displayed in the picture. The values of most of the displayed settings are determined by device parameters that are specified in the catalog entry for the current device or by graphics options that are specified in a GOPTIONS statement.

Note: The following two statements do not return the same parameters when used with PICTURE=1: \triangle

```
goptions dev=xcolor target=ps nodisplay;
goptions dev=ps nodisplay;
```

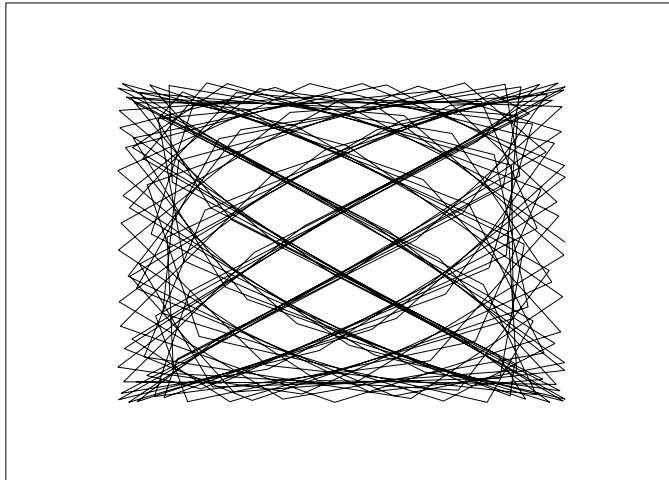
The LOG window for picture 1, shown in Output 45.1, lists some of the same settings that are displayed by picture 1, plus some additional settings.

Figure 45.1 Picture 1 of the GTESTIT Procedure



Picture 2 tests your device’s ability to draw lines. Picture 2 always displays in the first color of the current colors list. Figure 45.2 on page 1287 shows picture 2 of the GTESTIT procedure.

Figure 45.2 Picture 2 of the GTESTIT Procedure



Picture 3 tests your device’s ability to draw simple polygons, polygons with multiple boundaries (also known as *holes*), ellipses, and justified text. Figure 45.3 on page 1287 shows picture 3 of the GTESTIT procedure.

Figure 45.3 Picture 3 of the GTESTIT Procedure

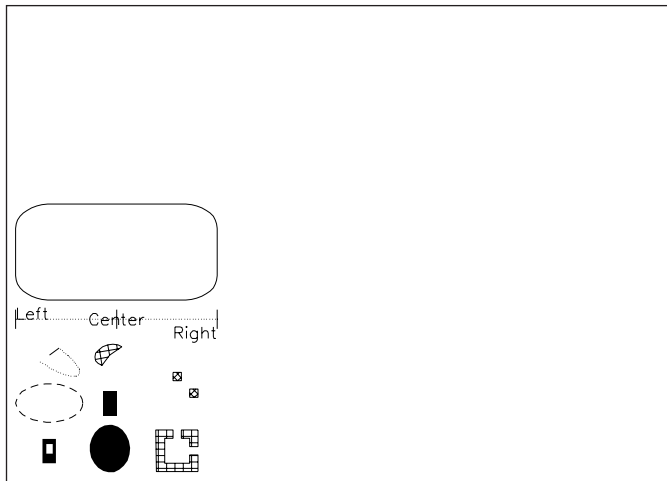


Table 45.1 on page 1288 explains the values displayed in picture 1 of the GTESTIT procedure. It also provides the equivalent graphics option or device parameter. Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 includes a complete description of the graphics options and device parameters.

Table 45.1 GTESTIT Values Displayed in Picture 1

GTESTIT Value	Equivalent Graphics Option or Device Parameter	Description
D=	DEVICE=	shows the device driver you are using.
B=		shows the baud rate for the device.
R=	VPOS=	shows the number of rows.
C=	HPOS=	shows the number of columns.
P=	MAXCOLORS=	shows the total number of colors (foreground and background) that your device can display. If your device can display more than 15 colors, picture 1 shows only 15 colors, but the LOG window lists all of the available colors.
H=		shows the height of character cells in pixels.
W=		shows the width of character cells in pixels.
MAX=	MAXPOLY=	shows the maximum number of vertices that can be processed by a hardware polygon command. If MAX=0, then the number of vertices is unbounded. If MAX=***, then the value is greater than 999.
D= *	DASHLINE=	shows the hardware dashed-line patterns available. The value displayed is a hexadecimal string.
RF= *	RECTFILL=	shows the hardware rectangle-fill patterns available. The value displayed is a hexadecimal string.
S= *	SYMBOLS=	shows the hardware symbols available. The value displayed is a hexadecimal string.
OPTS= *	DEVOPTS=	shows the other hardware options available. The value displayed is a hexadecimal string.
NCOLORS=	COLORS=	shows the number of colors in the colors list or the number of foreground colors.

GTESTIT Value	Equivalent Graphics Option or Device Parameter	Description
F=	FILLINC=	shows the solid fill increment (the number of pixels between strokes when doing a solid fill).

* In the device entry, this field may be blank. If blank, the value displayed by the GTESTIT procedure comes from an internal default in the device driver.

About the LOG

shows a sample of the information that appears in the LOG window after running picture 1 in the GTESTIT procedure. An asterisk (*) after the P=, MAX=, or F= option indicates that the value for that option is greater than 999.

Output 45.1 Sample Log from GTESTIT Procedure

```

1  proc gtestit picture=1;
2  run;
3  quit;
D=PSCOLOR B=1200 R= 25 C= 70 P=256
H= 16 W= 9 MAX=*** D=C000000000000000
RF=8000800000000000 S=0000000000000000
OPTS=D59A244009280000 NCOLORS= 1
Background color = WHITE
Color 1 = BLACK
Ratio = 0.71429
Hsize = 5.99539
Vsize = 4.28242
F=1
    
```

Table 45.2 on page 1289 lists GTESTIT values that appear only in the LOG window for picture 1: these values do not appear in the picture itself. Table 45.2 on page 1289 also provides the equivalent graphics option or device parameter. Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 contains complete information about the graphics options and device parameters.

Table 45.2 GTESTIT Values Shown in the LOG Window

GTESTIT Value	Equivalent Graphics Option or Device Parameter	Description
Background color=	CBACK=	tells the background color used.
Color1=...Colorn=	COLORS=	lists the default colors list for the device. <i>N</i> is equal to the NCOLORS= value.
Ratio=	ASPECT=	shows the aspect ratio of the device, which is the ratio of width to height of character cells.

GTESTIT Value	Equivalent Graphics Option or Device Parameter	Description
Hsize=	HSIZE=	shows the horizontal size of the area used on the device for the graphics display. The default unit is inches.
Vsize=	VSIZE=	shows the vertical size of the area used on the device for the graphics display. The default unit is inches.

Procedure Syntax

Supports: Output Delivery System (ODS)

```
PROC GTESTIT <PICTURE=1 | 2 | 3>
  <GOUT=<libref.>output-catalog>;
```

PROC GTESTIT Statement

Syntax

```
PROC GTESTIT <PICTURE=1 | 2 | 3>
  <GOUT=<libref.>output-catalog>;
```

Options

GOUT=< libref. >output-catalog

specifies the SAS catalog in which to save the graphics output produced by the GTESTIT procedure. If you omit the libref, SAS/GRAPH looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist.

See also: “Storing Graphics Output in SAS Catalogs” on page 53

PICTURE=1 | 2 | 3

PIC=1 | 2 | 3

indicates the number of the test pattern to display. By default, all three display. If you include more than one PICTURE= option, the GTESTIT procedure displays only the last picture you specify.

Values for PICTURE= are

- 1
shows available colors and patterns, line types, and fills.
- 2
shows the test pattern for continuous drawing ability.

3

shows the test pattern for drawing polygons, ellipses, and justified text.

Examples

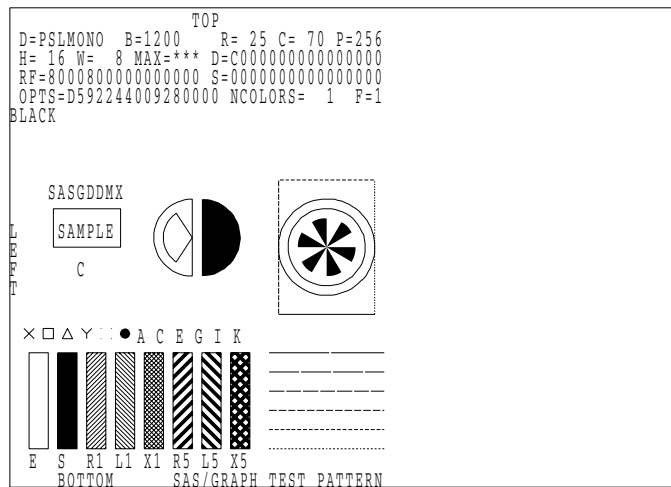
Example 1: Testing a GOPTIONS Statement

Features:

GOPTIONS statement

GTESTIT procedure

Sample library member: GITTGOS1



This example illustrates how you can use the GTESTIT procedure to confirm the settings specified on a GOPTIONS statement. In this example, the GOPTIONS statement enlarges the size of the elements in the graphics output by decreasing the number of columns from the default number of columns for the device, resets the font to the default, and specifies a limited colors list.

Set the graphics environment. HPOS= selects 45 columns. VPOS= selects 25 rows. FTEXT= resets the font to the default font. COLORS= can determine the colors displayed in picture 1 and listed in the LOG, and the value of NCOLORS=.

```
options hpos=45
        vpos=25
        ftext=
        colors=(blue red green);
```

Display the first picture of the GTESTIT procedure.

```
proc gtestit picture=1;
run;
quit;
```

Example 2: Displaying, Changing and Verifying the Colors List for a Device Driver

Features:

- 1 GOPTIONS statement
 - 2 GOPTIONS COLORS= option
 - 3 GTESTIT procedure
-

The colors that SAS/GRAPH uses for the elements of a graph are determined, in order of precedence, by:

- 1 The presence in a GOPTIONS statement of options that control the color of elements (for example, GOPTIONS CTEXT= to control the color of text).
- 2 The COLORS= option on a GOPTIONS statement
- 3 The default color list for the device driver for which the graph is being prepared.

For more information on the order in which SAS/GRAPH chooses colors, see “Defining and Using a Colors List” on page 93

You can use PROC GTESTIT to display the default color list for a device driver. For example, the following procedure displays the default color list for graphs prepared for the ActiveX control.

```
goptions reset=all device=activex;
proc gtestit picture=1;
run;
quit;
```

Running this procedure shows the default color list for the device ActiveX, as seen in the following output from PROC GTESTIT:

```

D=ACTIVEX  B=1200    R= 43 C= 83 P=256
H= 13 W=  9 MAX=  0 D=8000000000000000
RF=8000800000000000 S=0000000000000000
OPTS=3500304009280008 NCOLORS= 10
Background color = WHITE
Color 1 = BLACK
Color 2 = RED
Color 3 = GREEN
Color 4 = BLUE
Color 5 = CYAN
Color 6 = MAGENTA
Color 7 = GRAY
Color 8 = PINK
Color 9 = ORANGE
Color 10 = BROWN
Ratio = 0.74941
Hsize =    8.42
Vsize =    6.31
F=1

```

The following procedure uses the `COLORS=` option of the `GOPTIONS` statement to change (temporarily) the color list for the device driver `ActiveX`. Then, it invokes `PROC GTESTIT` to verify that the color list is changed. The color list is changed at most for the duration of the SAS session. Use the `GDEVICE` procedure to change the color list permanently.

```

goptions reset=all
        device=activex
        colors=(red, green, blue, yellow);
proc gtestit picture=1;
run;
quit;

```

The following output from `PROC GTEST` shows that the color list for `ActiveX` has changed:

```

D=ACTIVEX  B=1200    R= 43 C= 83 P=256
H= 13 W=  9 MAX=  0 D=8000000000000000
RF=8000800000000000 S=0000000000000000
OPTS=3500304009280008 NCOLORS=  4
Background color = WHITE
Color 1 = RED
Color 2 = GREEN
Color 3 = BLUE
Color 4 = YELLOW
Ratio = 0.74941
Hsize =    8.42
Vsize =    6.31
F=1

```

The following code resets the color list for the `ActiveX` device to the default, and then reissues `PROC GTESTIT` to verify that the colors have been reset:

```

goptions colors=(
    device=activex;
proc gtestit picture=1;
run;
quit;

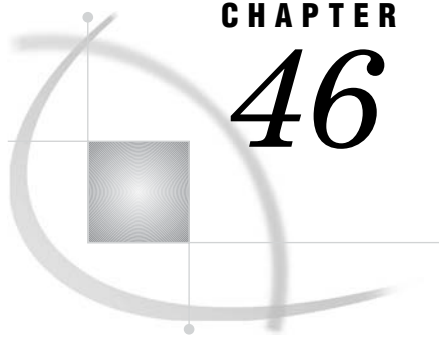
```

The following output confirms that the default color list has been re-established:

```

D=ACTIVEX  B=1200    R= 43 C= 83 P=256
H= 13 W=  9 MAX=  0 D=8000000000000000
RF=8000800000000000 S=0000000000000000
OPTS=3500304009280008 NCOLORS= 10
Background color = WHITE
Color 1 = BLACK
Color 2 = RED
Color 3 = GREEN
Color 4 = BLUE
Color 5 = CYAN
Color 6 = MAGENTA
Color 7 = GRAY
Color 8 = PINK
Color 9 = ORANGE
Color 10 = BROWN
Ratio = 0.74941
Hsize =  8.42
Vsize =  6.31
F=1

```



CHAPTER

46

The G3D Procedure

<i>Overview</i>	1295
<i>About Surface Plots</i>	1295
<i>About Scatter Plots</i>	1296
<i>Concepts</i>	1297
<i>Parts of a Three-dimensional Plot</i>	1297
<i>About the Input Data Set</i>	1298
<i>Data for Surface Plots</i>	1298
<i>Data for Scatter Plots</i>	1298
<i>Changing Data Ranges</i>	1298
<i>About Rotating and Tilting the Plot</i>	1299
<i>About Controlling the Axes</i>	1299
<i>Procedure Syntax</i>	1300
<i>PROC G3D Statement</i>	1300
<i>PLOT Statement</i>	1301
<i>SCATTER Statement</i>	1305
<i>Examples</i>	1314
<i>Example 1: Generating a Default Surface Plot</i>	1314
<i>Example 2: Rotating a Surface Plot</i>	1316
<i>Example 3: Tilting Surface Plot</i>	1317
<i>Example 4: Generating a Simple Scatter Plot</i>	1318
<i>Example 5: Using Shapes in Scatter Plots</i>	1320
<i>Example 6: Rotating a Scatter Plot</i>	1323
<i>References</i>	1325

Overview

The G3D procedure produces three-dimensional graphs that plot one vertical variable (z) for a position on a plane that is specified by two horizontal variables (x and y). The coordinates of each point correspond to the values of three numeric variable values in an observation of the input data set. The observation may contain values in the form $y=f(x, y)$ or independent values such as the altitude at a given longitude and latitude. With the G3D procedure you can generate surface graphs with the PLOT statement or scatter plots with the SCATTER statement.

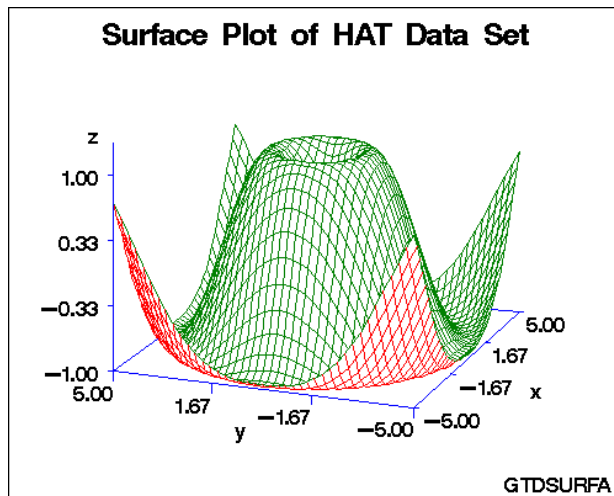
About Surface Plots

Surface plots show the three-dimensional shape of your data and are useful for examining data trends. The plots represent the shape of the surface that is described by the values of two horizontal variables, x and y , and a third vertical variable, z . The

values of the horizontal variables are plotted on x and y axes, which form a horizontal plane. The values of the vertical variable are plotted on a z axis, rising above that plane to form a three-dimensional surface.

Figure 46.1 on page 1296 shows an example of a surface plot that uses all default settings for the plot. The axes are scaled to include the maximum and minimum values for each of the plotted variables x , y , and z . Each variable's value range is divided into three even intervals, which form the major axes tick marks, and the axes are labeled with the names of the plotted variables or associated labels. The horizontal plane formed by the x and y axes is rotated 70° around the z axis and also tilted 70° toward you, and the plot is colored with the colors that are defined in the current colors list.

Figure 46.1 Sample G3D Surface Plot



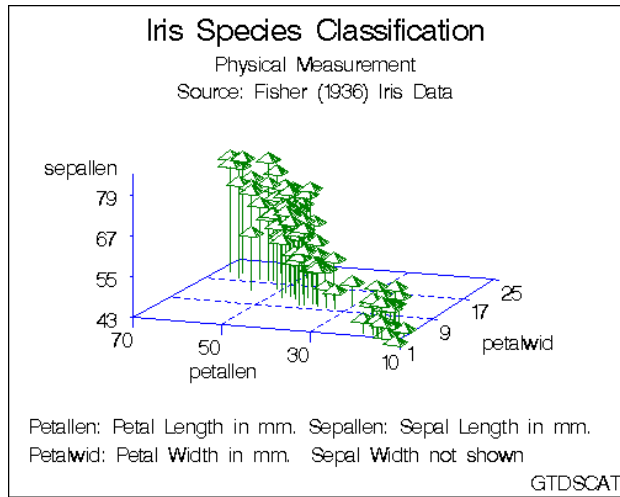
The program for this plot is shown in Example 1 on page 1314. For more information on producing surface plots, see “PLOT Statement” on page 1301.

About Scatter Plots

Scatter plots are three-dimensional plots that are similar to surface plots, but they represent the data as points instead of surfaces. Scatter plots show trends or concentrations in the data by classifying the data by size, color, shape, or a combination of these features. As with surface plots, the values of the x and y variables in scatter plots form a horizontal plane, and the values of the z variable rise above that plane. Rather than forming a surface, however, the values of the z variable are represented as individual symbols that can be optionally connected to the horizontal plane with lines called *needles*.

Figure 46.2 on page 1297 shows a simple scatter plot. As with surface plots, default settings for scatter plots scale the axes to include the maximum and minimum values for each of the plotted variables x , y , and z , and divide each variable's value range into three even intervals to form the major axes tick marks. Default settings also rotate the horizontal plane 70° around the z axis and tilt it 70° toward you, label each axis with the name of the plotted variable or an associated label, and color the plot with colors that are defined in the current colors list. The default settings also add reference lines to the horizontal plane to mark the major x and y axes tick marks, and represent each data point with a pyramid, which is connected to the horizontal plane with a needle.

Figure 46.2 Sample G3D Scatter Plot

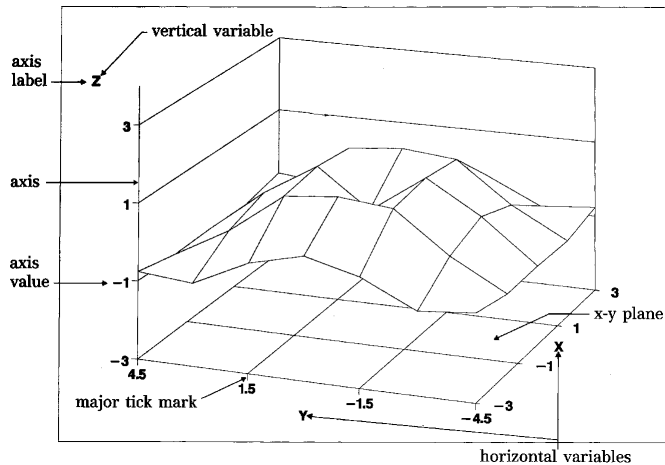


The program for this plot is shown in Example 4 on page 1318. For more information on producing scatter plots, see “SCATTER Statement” on page 1305.

Concepts

Parts of a Three-dimensional Plot

Figure 46.3 G3D Procedure Terms



About the Input Data Set

The G3D procedure requires data sets that include three numeric variables: two horizontal variables plotted on the x and y axes that define an x - y plane, and a vertical variable plotted on the z axis rising from the $\{x$ - $y\}$ plane.

Data for Surface Plots

For surface plots, the observations in the input data set should form an evenly spaced grid of horizontal (x and y) values and exactly one vertical (z) value for each of these combinations. For example, data that contains 5 distinct values for x and 10 distinct values for y should be part of a data set that contains 50 observations with values for x , y , and z .

Only one z point is plotted for each combination of x and y . For example, you cannot draw a sphere using the PLOT statement. If there is more than one observation for a combination of x and y in the data set, only the last such point is used.

For the G3D procedure to produce a satisfactory surface plot, the data set must contain nonmissing z values for at least 50 percent of the grid cells. When the G3D procedure cannot produce a satisfactory surface plot because of missing z values, the SAS/GRAPH software issues a warning message and a graph may not be produced. To correct this problem, process the data set with the G3GRID procedure and use the processed data set as the input data set for G3D. The G3GRID procedure interpolates the necessary values to produce a data set with nonmissing z values for every combination of x and y . The G3GRID procedure can also smooth data for use with the G3D procedure. See Chapter 47, “The G3GRID Procedure,” on page 1327 for more information on the G3GRID procedure.

Data for Scatter Plots

An input data set for scatter plots must include at least two observations that contain different values for each of the three variables that are specified in the plot request so that the G3D procedure can scale the axes. If the data set does not meet these requirements, the SAS/GRAPH software issues an error message and no graph is produced.

For scatter plots, only one z value is plotted for a combination of x and y . For example, you cannot draw a sphere using the SCATTER statement. If there is more than one observation for a combination of x and y in the data set, only the last point is used. See “Simulating an Overlaid Scatter Plot” on page 1310 for information on producing scatter plots with more than one vertical value for each x , y combination.

Changing Data Ranges

By default for both surface plots and scatter plots, the range of the z axis is defined by the minimum and maximum z values in the input data set. Restrict or expand the range of the z axis by using the ZMIN= and ZMAX= options in the PLOT or SCATTER statement. To restrict the range of an x or y axis, use a WHERE statement in the PROC step or a WHERE or IF statement in a DATA step to create a subset of the data set.

Note: AXIS and LEGEND definitions are not supported by the G3D procedure. Use the Annotate facility or TITLE, FOOTNOTE, and NOTE statements to produce legends, tick mark values, and axis labels. See “About Controlling the Axes” on page 1299 and “SCATTER Statement” on page 1305 for information on controlling axis labels and tick mark values with PLOT statement and SCATTER statement options. \triangle

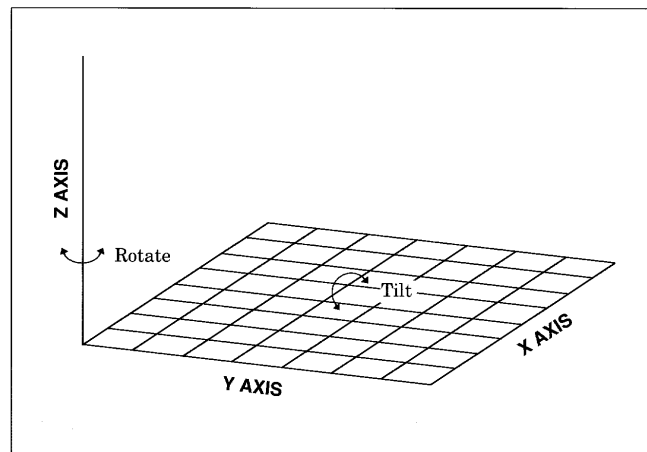
About Rotating and Tilting the Plot

For both surface plots and scatter plots, you can rotate the x - y plane about the z axis, or tilt the plot toward you. When you rotate a plot, you can view data from any angle around the three-dimensional graph. This is useful for bringing into view data points that were previously hidden by other data points on a plot. Tilting a plot enables you to accentuate the location of data points.

Figure 46.4 on page 1299 shows how rotating and tilting can change the viewing angle of a graph.

Note: At certain combinations of tilt and rotation angles, the tick mark values may overlap. Δ

Figure 46.4 Rotating and Tilting a Graph



About Controlling the Axes

Because the relationship between a plot's surface and the actual data values can be difficult to interpret, you can improve a graph by changing the number of tick marks on the axes or restricting the range of the vertical (z) variable.

The G3D procedure does not support AXIS definitions; however, you can use PLOT or SCATTER statement options to

- suppress the axes
- suppress axis labels
- suppress tick mark values
- specify the number of tick marks
- specify minimum and maximum values for the z axis
- specify whether grid lines connect axis tick marks.

You can also change the font and height of axis labels and axis values by specifying the desired font and height with the FTEXT= and HTEXT= options on a GOPTIONS statement.

For information on how to reverse the values on an axis, see "Reversing Values on an Axis" on page 1312.

Procedure Syntax

Requirements: At least one PLOT or SCATTER statement is required.

Global statements: FOOTNOTE, GOPTIONS, TITLE

Reminder: The procedure can include the BY, FORMAT, LABEL, NOTE, and WHERE statements.

Supports: Output Delivery System (ODS)

```
PROC G3D <DATA=input-data-set>
      <ANNOTATE=Annotate-data-set>
      <GOUT=<libref.>output-catalog>;
PLOT plot-request</options>;
SCATTER plot-request</option(s)>;
```

PROC G3D Statement

Identifies the data set that contains the plot variables. Optionally specifies annotation and an output catalog.

Requirements: An input data set is required.

Syntax

```
PROC G3D <DATA=input-data-set>
      <ANNOTATE=Annotate-data-set>
      <GOUT=<libref.>output-catalog>;
```

Options

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate all of the graphs that are produced by the G3D procedure. To annotate individual graphs, use ANNOTATE= in the action statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

DATA=*input-data-set*

specifies the SAS data set that contains the variables to plot. By default, the procedure uses the most recently created SAS data set.

See also: “SAS Data Sets” on page 29 “About the Input Data Set” on page 1298

GOUT=< *libref.* >*output-catalog*

specifies the SAS catalog in which to save the graphics output that is produced by the G3D procedure. If you omit the libref, the SAS/GRAPH software places the output in the temporary catalog WORK.GSEG. The output catalog is created if it doesn't already exist.

See also: “Storing Graphics Output in SAS Catalogs” on page 53

PLOT Statement

Creates three-dimensional surface plots using values of three numeric variables from the input data set.

Requirements: Exactly one plot request is required.

Global statements: FOOTNOTE, TITLE

Description

The PLOT statement specifies one plot request that identifies the three numeric variables to plot. This statement automatically

- scales the axes to include the maximum and minimum values for each of the plotted variables x , y , and z
- divides the value range for each variable into three even intervals, which are represented by four major tick marks on the axis
- rotates the x - y plane 70° around the z axis and tilts it 70° toward you, labeling each axis with the name of the plotted variable or an associated label
- colors the plot with colors that are defined in the current colors list: axis labels and tick mark labels display in the first color from the list, axes display in the second color, the top of the surface plot displays in the third color, and the bottom of the surface plot (if visible) displays in the fourth color.

You can use statement options to modify any of the three plot axes as well as the general appearance of the graph, control the viewing angle, and specify characteristics for reference lines.

In addition, you can use global statements to add text to the graph, and an Annotate data set to enhance the plot.

Syntax

PLOT *plot-request* </option(s)>;

plot-request must be

$y^*x=z$

option(s) can be one or more options from any or all of the following categories:

- appearance options:
 - ANNOTATE=*Annotate-data-set*
 - CBOTTOM=*bottom-surface-color*
 - CTOP=*top-surface-color*
 - ROTATE=*angle-list*
 - SIDE
 - TILT=*angle-list*
 - XYTYPE=1 | 2 | 3
- axes options:
 - CAXIS=*axis-color*
 - CTEXT=*text-color*
 - GRID
 - NOAXIS | NOAXES

NOLABEL
 XTICKNUM=*number-of-ticks*
 YTICKNUM=*number-of-ticks*
 ZMAX=*max-value*
 ZMIN=*min-value*
 ZTICKNUM=*number-of-ticks*

- catalog entry description options:

DESCRIPTION=*'entry-description'*
 NAME=*'entry-name'*

Required Arguments

y*x=z

specifies three numeric variables from the input data set:

y

is one of the variables that is plotted on the horizontal (*x-y*) plane.

x

is another of the variables that is plotted on the horizontal (*x-y*) plane.

z

is the variable that is plotted on the vertical (*z*) axis.

Options

Options in a PLOT statement affect all graphs that are produced by that statement. You can specify as many options as you want and list them in any order.

ANNOTATE=Annotate-data-set

ANNO=Annotate-data-set

specifies a data set to annotate plots that are produced by the PLOT statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

CAXIS=axis-color

specifies a color for axis lines and tick marks. By default, axes are displayed in the second color in the current colors list.

CBOTTOM=bottom-surface-color

specifies a color for the bottom of the plot surface. By default, the bottom surface is displayed in the fourth color in the current colors list.

Featured in: Example 2 on page 1316

CTEXT=text-color

specifies a color for all text on the axes, including tick mark values and axis labels. If you omit this option, a color specification is searched for in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the default, the first color in the colors list.

CTOP=top-surface-color

specifies a color for the top of the plot surface. By default, the top surface is displayed in the third color in the current colors list.

Featured in: Example 2 on page 1316

DESCRIPTION=*'entry-description'***DES=***'entry-description'*

specifies the description of the catalog entry for the chart. The maximum length is 256 characters. The description does not appear on the chart. By default, the procedure assigns a description of the form PLOT OF $y^*x=z$, where $y^*x=z$ is the request that is specified in the PLOT statement.

GRID

draws reference lines at the major tick marks on all axes.

Featured in: Example 2 on page 1316

NAME=*'entry-name'*

specifies the name of the catalog entry for the graph. The maximum length is 8 characters. The default name is G3D. If you specify DEVICE=ACTIXIMG or DEVICE=JAVAIMG, then the name that you specify will be used for the client image output even in the file exists. For all other devices, if the name duplicates an existing entry name, SAS/GRAPH adds a number to the duplicate name to create a unique entry, for example, G3D1.

NOAXIS**NOAXES**

specifies that a plot have no axes, axis labels, or tick mark values.

NOLABEL

specifies that a plot have no axis labels or tick mark values. Use this option if you want to generate axis labels and tick mark values with an Annotate data set.

ROTATE=*angle-list*

specifies one or more angles at which to rotate the x - y plane about the perpendicular z axis. The units are degrees. The default value is 70. The *angle-list* value is either an explicit list of values, or a starting and an ending value with an interval increment, or a combination of both forms:

n <... n >

n TO n <BY *increment*>

n <... n > TO n <BY *increment* > < n <... n > >

The values specified in the *angle-list* value can be negative or positive and can be larger than 360° . For example, a rotation angle of 45° can also be expressed as

```
rotate=405
```

```
rotate=-315
```

You can specify a sequence of angles to produce separate graphs for each angle. The angles that are specified in the ROTATE= option are paired with any angles that are specified with the TILT= option. If one option contains fewer values than the other, the last value in the shorter list is paired with the remaining values in the longer list.

See also: TILT= option on page 1303

Featured in: Example 2 on page 1316

SIDE

produces a surface graph with a side wall.

Featured in: Example 3 on page 1317

TILT=*angle-list*

specifies one or more angles at which to tilt the graph toward you. The units are degrees and the default value is 70. The *angle-list* value is either an explicit list of values, or a starting and an ending value with an interval increment, or a combination of both forms:

n <... n >

n TO n <BY increment>

n <... n > TO n <BY increment > < n <... n > >

The values that are specified in the *angle-list* value must be 0 through 90.

You can specify a sequence of angles to produce separate graphs for each angle. The angles that are specified in the TILT= option are paired with any angles that are specified with the ROTATE= option. If one option contains fewer values than the other, the last value in the shorter list is paired with the remaining values in the longer list.

See also: ROTATE= option on page 1303

Featured in: Example 3 on page 1317

XTICKNUM=number-of-ticks

YTICKNUM=number-of-ticks

ZTICKNUM=number-of-ticks

specify the number of major tick marks that are located on a plot's x , y , or z axis, respectively. The value must be 2 or greater. The default value is 4 for all three options.

Featured in: Example 2 on page 1316

XYTYPE=1 | 2 | 3

specifies the direction of lines that are used to represent the surface. XYTYPE=1 displays the surface by using lines that represent y axis values. That is, it only draws lines that are parallel to the x axis. XYTYPE=2 displays the surface by using lines that represent x axis values, and draws only lines that are parallel to the y axis. XYTYPE=3 displays the surface by using lines that represent values for both the x and y axes. The default is XYTYPE=3. See Figure 46.5 on page 1305 for an example of the effect of XYTYPE= on the appearance of the surface.

ZMAX=max-value

ZMIN=min-value

specify the maximum and minimum values that are displayed on a plot's z axis. By default, the z axis is defined by the minimum and maximum z values that are in the data set. Defining the ZMIN= and ZMAX= options to be greater than the minimum and maximum values in the data set extends the plot's z axis. Defining the ZMIN= and ZMAX= options to be less than the minimum and maximum values in the data set displays all z values in the range of ZMIN-to-ZMAX. Values that exceed that range are displayed at the values of the ZMIN= or ZMAX= options.

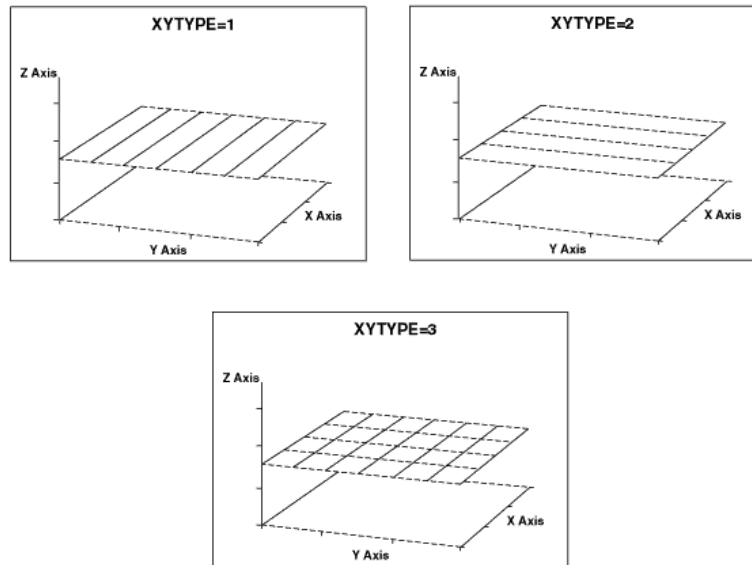
The value of the ZMAX= option must be greater than the value of the ZMIN= option.

Featured in: Example 2 on page 1316

Changing the Surface Appearance

Use the XYTYPE= option to change the appearance of the plot surface. This option lets you select the direction of the lines that form the surface plot. Figure 46.5 on page 1305 shows examples of each type of plot surface.

Figure 46.5 Surface Appearance for Different XYTYPE= Values



SCATTER Statement

Creates three-dimensional scatter plots using values of three numeric variables from the input data set.

Requirements: Exactly one plot request is required.

Global statements: FOOTNOTE, TITLE

Alias: SCAT

Description

The SCATTER statement specifies one plot request that identifies the three numeric variables to plot. This statement automatically

- scales the axes to include the maximum and minimum values for each of the plotted variables x , y , and z .
- divides the range for each variable into three even intervals that are represented by four major tick marks on the axis.
- uses reference lines to mark the major tick marks on the x and y axes.
- rotates the x - y plane 70° around the z axis and tilts it 70° toward you, labeling each axis with the name of the plotted variable or an associated label.
- uses the colors that are defined in the current colors list: axis labels and tick mark labels display in the first color from the colors list, axes in the second color, and data points in the third color.
- represents each data point with a pyramid that is connected to the horizontal plane with a needle.

You can use statement options to modify any of the three plot axes as well as the general appearance of the graph, control the viewing angle, and specify characteristics for reference lines. In addition, if the needles drawn from the data points to the base plane complicate a graph, you can suppress them.

You can use global statements to add text to the graph, and an Annotate data set to enhance the plot.

Syntax

SCATTER *plot-request* *</option(s)>*;

plot-request must be

*y*x=z*

option(s) can be one or more options from any or all of the following categories:

- appearance options:
 - ANNOTATE=*Annotate-data-set*
 - COLOR=*'data-point-color'* | *data-point-color-variable*
 - NONEEDLE
 - ROTATE=*angle-list*
 - SHAPE=*'symbol-name'* | *shape-variable*
 - SIZE=*symbol-size* | *size-variable*
 - TILT=*angle-list*
- axes options:
 - CAXIS=*axis-color*
 - CTEXT=*text-color*
 - GRID
 - NOAXIS | NOAXES
 - NOLABEL
 - XTICKNUM=*number-of-ticks*
 - YTICKNUM=*number-of-ticks*
 - ZMAX=*max-value*
 - ZMIN=*min-value*
 - ZTICKNUM=*number-of-ticks*
- catalog entry description options:
 - DESCRIPTION=*'entry-description'*
 - NAME=*'entry-name'*

Required Arguments

y*x=z

specifies three numeric variables from the input data set:

y

is one of the variables that is plotted on the horizontal (*x-y*) plane.

x

is another of the variables that is plotted on the horizontal (*x-y*) plane.

z

is the variable that is plotted on the vertical (*z*) axis.

The SCATTER statement does not require a full grid of observations for the horizontal variable.

Options

Options in a SCATTER statement affect all graphs that are produced by that statement. You can specify as many options as you want and list them in any order.

ANNOTATE=*Annotate-data-set*

ANNO=*Annotate-data-set*

specifies a data set to annotate plots that are produced by the SCATTER statement.

See also: Chapter 24, “Using Annotate Data Sets,” on page 587

CAXIS=*axis-color*

specifies a color for axis lines and tick marks. By default, axes display in the second color in the colors list.

Featured in: Example 6 on page 1323

COLOR=*'data-point-color'* | *data-point-color-variable*

specifies a color name or a character variable in the input data set whose values are color names. These color values determine the color or colors of the shapes that represent a plot's data points. Color values must be valid color names for the device that is used. By default, plot shapes display in the third color in the current colors list.

Using a list of colors in the value of the *data-point-color-variable* enables you to assign different colors to the shapes to classify data.

Featured in: Example 5 on page 1320

CTEXT=*text-color*

specifies a color for all text on the axes, including tick mark values and axis labels. If you omit this option, a color specification is searched for in this order:

- 1 the CTEXT= option in a GOPTIONS statement
- 2 the default, the first color in the colors list.

DESCRIPTION=*'entry-description'*

DES=*'entry-description'*

specifies the description of the catalog entry for the chart. The maximum length for is 256 characters. The description does not appear on the chart. By default, the procedure assigns a description of the form SCATTER OF $y*x=z$, where $y*x=z$ is the request that is specified in the SCATTER statement.

GRID

draws reference lines at the major tick marks on all axes.

Featured in: Example 5 on page 1320

NAME=*'entry-name'*

specifies the name of the catalog entry for the graph. The maximum length is eight characters. The default name is G3D. If you specify DEVICE=ACTIXIMG or DEVICE=JAVAIMG, then the name that you specify will be used for the client image output even in the file exists. For all other devices, if the name duplicates an existing entry name, SAS/GRAPH adds a number to the duplicate name to create a unique entry, for example, G3D1.

NOAXIS

NOAXES

specifies that a plot have no axes, axis labels, or tick mark values.

NOLABEL

specifies that a plot have no axis labels or tick mark values. Use this option if you want to generate axis labels and tick mark values with an Annotate data set.

NONEEDLE

specifies that a plot have no lines that connect the shapes representing data points to the x - y plane. The NONEEDLE option has no effect when SHAPE='PILLAR' or SHAPE='PRISM'.

Featured in: Example 5 on page 1320

ROTATE=*angle-list*

specifies one or more angles at which to rotate the x - y plane about the perpendicular z axis. The units are degrees and the default value is 70. The *angle-list* value can be a list of values, a starting and an ending value with an interval increment, or a combination of both forms:

n <... n >

n TO n <BY *increment*>

n <... n > TO n <BY *increment* > < n <... n > >

The *angle-list* value(s) can be negative or positive and can be larger than 360° . For example, a rotation angle of 45° can also be expressed

```
rotate=405
rotate=-315
```

You can specify a sequence of angles to produce separate graphs for each angle. The angles that are specified in the ROTATE= option are paired with any angles that are specified with the TILT= option. If one option contains fewer values than the other, the last value in the shorter list is paired with the remaining values in the longer list.

See also: TILT= option on page 1309.

Featured in: Example 6 on page 1323

SHAPE='symbol-name' | *shape-variable*

specifies a symbol name or a character variable whose values are symbol names. Symbols represent a scatter plot's data points. By default, SHAPE='PYRAMID'.

Values for *symbol-name* are

BALLOON

CLUB

CROSS

CUBE

CYLINDER

DIAMOND

FLAG

HEART

PILLAR

POINT

PRISM

PYRAMID

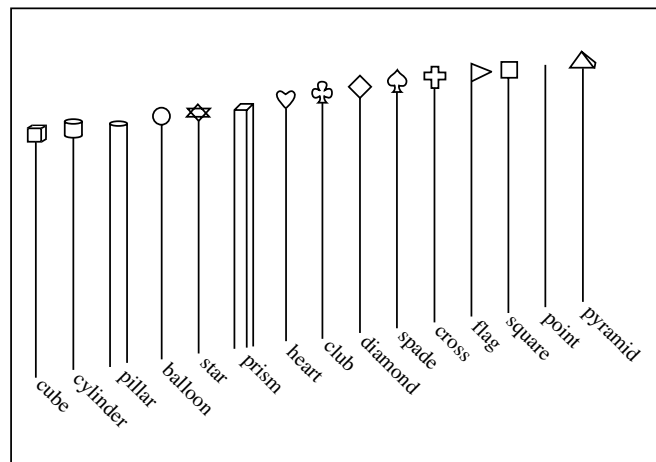
SPADE

SQUARE

STAR.

Figure 46.6 on page 1309 illustrates these symbol types with needles.

Figure 46.6 Scatter Plot Symbols



If you specify `SHAPE='symbol-name'`, all data points are drawn in that shape. For example, the procedure draws all data points as balloons when you specify

```
shape='balloon'
```

If you specify `SHAPE=shape-variable`, the shape of the data point is determined by the value of the shape variable for that observation. For example, the procedure uses the value of the variable `CLASS` for a particular observation as the shape for that data point when you specify

```
shape=class
```

Using a list of values in the variable named in `SHAPE=shape-variable` enables you to assign different shapes to the data points to classify data.

Featured in: Example 5 on page 1320

SIZE=*symbol-size* | *size-variable*

specifies either a constant or a numeric variable, the values of which determine the size of symbol shapes on the scatter plot.

If you specify `SIZE=symbol-size`, all data points are drawn in that size. For example, if you specify `SIZE=3`, the procedure draws all symbol shapes three times the normal size. By default, `SIZE=1.0`. The units are in default symbol size.

If you specify `SIZE=size-variable`, the size of the data point is determined by the value of the size variable for that observation. For example, when you specify `SIZE=CLASS`, the procedure uses the value of the variable `CLASS` for each observation as the size of that data point. If you use a list of sizes as the value of the variable named in `SIZE=size-variable`, you can assign different sizes to the data points to classify data.

Featured in: Example 6 on page 1323

TILT=*angle-list*

specifies one or more angles at which to tilt the graph toward you. The units are degrees and the default value is 70. The value can be a list, a starting and an ending value with an interval increment, or a combination of both forms:

```
n <...n>
```

```
n TO n <BY increment>
```

n $\langle \dots n \rangle$ TO n \langle BY *increment* \rangle $\langle n \langle \dots n \rangle \rangle$

The values that are specified in *angle-list* must be 0 through 90.

You can specify a sequence of angles to produce separate graphs for each angle. The angles that are specified in the TILT= option are paired with any angles that are specified with the ROTATE= option. If one option contains fewer values than the other, the last value in the shorter list is paired with the remaining values in the longer list.

See also: ROTATE= option on page 1308

XTICKNUM=number-of-ticks

YTICKNUM=number-of-ticks

ZTICKNUM=number-of-ticks

specify the number of major tick marks that are located on a plot's x , y , or z axis, respectively. The values must be 2 or greater. For all three options, the default value is 4.

Featured in: Example 6 on page 1323

ZMAX=max-value

ZMIN=min-value

specify the maximum and minimum values that are displayed on a plot's z axis. By default, the z axis is defined by the minimum and maximum z values in the data. You can use the ZMIN= and ZMAX= options to extend the z axis beyond this range. The value that is specified by ZMAX= must be greater than that specified by ZMIN=. If you specify a ZMAX= or ZMIN= value within the actual range of the z variable values, the plot's data values are clipped at the specified level.

Featured in: Example 6 on page 1323

Changing the Appearance of the Points

Use the COLOR=, SHAPE=, and SIZE= options to change the appearance of your scatter plot or to classify data using color, shape, size, or any combination of these features. Figure 46.6 on page 1309 illustrates the shape names that you can specify in the SHAPE= option. For example, to make all of the data points red balloons at twice the normal size, use

```
scatter y*x=z /color='red' shape='balloon' size=2;
```

To size your points according to the values of the variable TYPE in your input data set, use

```
scatter y*x=z / size=type;
```

For an example, see Example 5 on page 1320.

Simulating an Overlaid Scatter Plot

You can approximate an overlaid scatter plot by graphing multiple values for the vertical (z) variables for a single (x, y) position in a single scatter plot. To do this, add a small value to the value of one of the horizontal variables (x or y) to give the observation a slightly different (x, y) position. Thus, you enable the procedure to plot both values of the vertical (z) variable. Represent each different vertical (z) variable with a different symbol, size, or color. The resulting plot appears to be multiple plots overlaid on the same axes.

For example, suppose you want to graph a data set that contains two values for the vertical variable Z for each combination of variables X and Y . You could produce the original data set with a DATA step like this:

```

data planes;
  input x y z shape $;
  datalines;
1 1 1 PRISM
1 2 1 PRISM
1 3 1 PRISM
2 1 1 PRISM
2 2 1 PRISM
2 3 1 PRISM
3 1 1 PRISM
3 2 1 PRISM
3 3 1 PRISM
1 1 2 BALLOON
1 2 2 BALLOON
1 3 2 BALLOON
2 1 2 BALLOON
2 2 2 BALLOON
2 3 2 BALLOON
3 1 2 BALLOON
3 2 2 BALLOON
3 3 2 BALLOON
;

```

The SHAPE variable is assigned a different value for each different Z value for a single combination of X and Y values.

Ordinarily, the SCATTER statement only plots the Z value for the last observation for a single combination of X and Y. However, you can use a DATA step to assign a slightly different x, y position to all observations where Z is greater than 1:

```

data planes2;
  set planes;
  if z > 1 then x = x + .000001;
run;

```

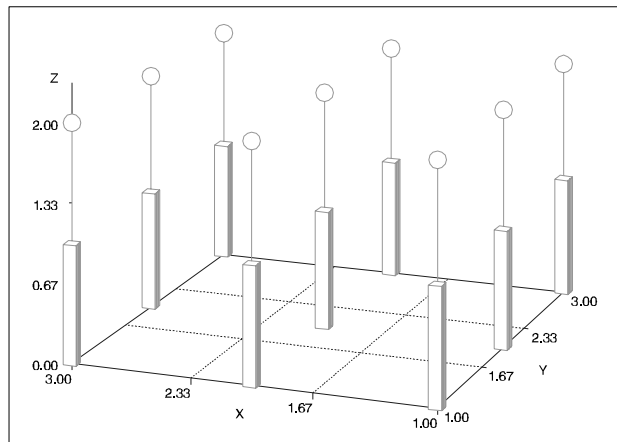
Then you can use a SCATTER statement to produce a plot like the one in Figure 46.7 on page 1312:

```

proc g3d data=planes2;
  scatter x*y=z / zmin=0 shape=shape;
run;
quit;

```

Figure 46.7 Simulated Overlaid Scatter Plot



Reversing Values on an Axis

Although you can use the SCATTER statement's ROTATE option to alter the view of a plot and therefore the general orientation, you cannot use SCATTER statement options to reverse axis values for one of the plot variables. To do this, you can multiply that variable's values by -1 to reverse the values themselves, which has the result of reversing the axis when those values are used to generate a plot. You should then use PROC FORMAT to define a format that displays the variable's values as they exist in the original data.

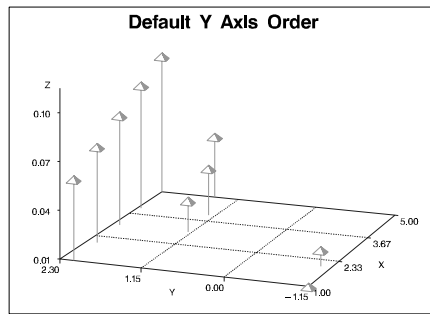
For example, the following code generates the scatter plot shown in Figure 46.8 on page 1313:

```
data original;
  input y x z;
  datalines;
-1.15 1 .01
-1.00 2 .02
 1.20 3 .03
 1.25 4 .04
 1.50 5 .05
 2.10 1 .06
 2.15 2 .07
 2.20 3 .08
 2.25 4 .09
 2.30 5 .10
;

title1 'Default Y Axis Order';

/* default Y axis order */
proc g3d data=original;
  scatter y * x = z;
run;
```

Figure 46.8 Default Y-Axis Order



To reverse the Y axis in the plot that is shown in Figure 46.8 on page 1313, you can write a DATA step like the following to reverse the Y values and, therefore, reverse the Y axis when the values are plotted:

```
data minus_y;
    set original;
    y=-y;
run;
```

The previous code creates the MINUS_Y data set by reading the ORIGINAL data set, and then multiplying the values of variable Y by -1. Although plotting Y values from the MINUS_Y data set would reverse values on the Y axis, it would misrepresent the original data. Such a plot would label the axis with the negative-Y values. You can correct the problem by using PROC FORMAT to display Y values as they are stored in the ORIGINAL data set:

```
proc format;
    picture reverse
        low - < 0 = '09.00'
        0 < - high = '09.00' (prefix='-')
        0 = '09.00';
run;
```

Here, the PICTURE statement defines a picture format named REVERSE, which you can refer to in DATA and PROC steps by using the name followed by a period. A picture format is a template for printing numbers. The '09.00' specifications are *digit selectors* that indicate which digits or columns in the variable values will display in output; columns that do not have a specified digit selector will not be displayed in output. Thus, a picture format for displaying the values of variable Y needs a column for a minus sign, a column for units, and two columns for decimals. The digit selector 0 specifies that no leading zeros will display in a column, and the digit selector 9 specifies that a leading zero will display in a column.

The PICTURE statement defines this new picture format for three data ranges. The lowest value in the data up to but not including zero will display with no prefix, which means negative values will display without a minus sign. All values above (but not including) zero to the highest value in the data will be displayed with the specified prefix, which in this case is a minus sign. Because zero is excluded from both ranges, it is assigned its own picture with no prefix.

You can now assign the REVERSE format to the Y values from the MINUS_Y data set and use Y to generate a scatter plot. The resulting plot displays Y's negative values without a prefix, and its positive values display with a minus sign prefix. This effectively represents Y values as they are stored internally in the ORIGINAL data set, thus correcting the data misrepresentation that results from multiplying Y by -1.

The following code generates the scatter plot shown in Figure 46.9 on page 1314:

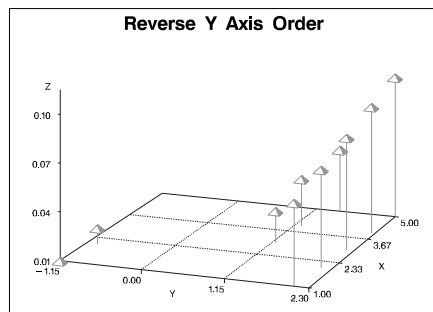
```

title1 'Reverse Y Axis Order';

/* reverses order of default Y axis */
proc g3d data=minus_y;
  format y reverse.;
  scatter y * x = z;
run;
quit;

```

Figure 46.9 Reverse Y-Axis Order



Examples

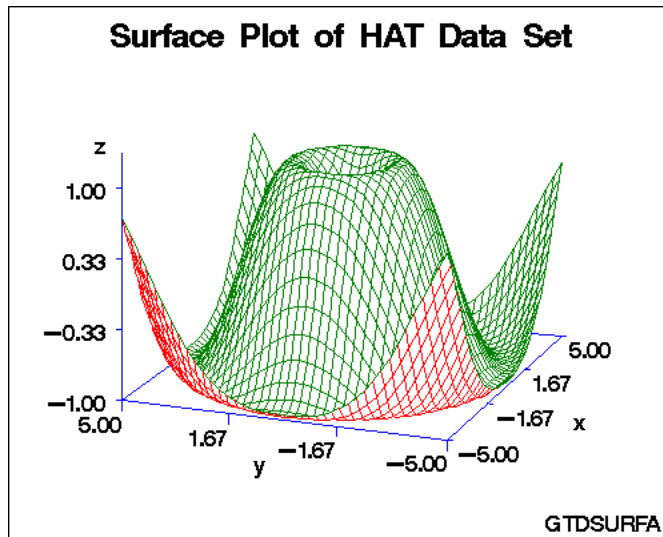
Example 1: Generating a Default Surface Plot

Procedure features:

PLOT statement

Sample library member: GTDSURFA

Figure 46.10 A Surface Plot with Default Option Values



This example shows a surface plot that reveals the shape of a generated data set named HAT. The PLOT statement in this example relies entirely on procedure defaults. The axes are scaled to include all data values and are labeled with the names of the axes variables. The axes major tick marks are divided into three even intervals, and the horizontal plane is rotated 70° around the z axis and tilted 70° toward you. The plot is displayed with the colors that the GOPTIONS statement defines for the colors list.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss ftitle=swissb htitle=6 htext=4;
```

Create the data set. REFLIB.HAT is generated data that produces a symmetric surface pattern, which is useful for illustrating the PLOT statement and its options.

```
data reflib.hat;
  do x=-5 to 5 by 0.25;
    do y=-5 to 5 by 0.25;
      z=sin(sqrt(x*x+y*y));
      output;
    end;
  end;
run;
```

Define title and footnote.

```
title 'Surface Plot of HAT Data Set';
footnote j=r 'GTDSURFA';
```

Generate the surface plot.

```
proc g3d data=reflib.hat;
  plot y*x=z;
run;
quit;
```

Example 2: Rotating a Surface Plot**Procedure Features**

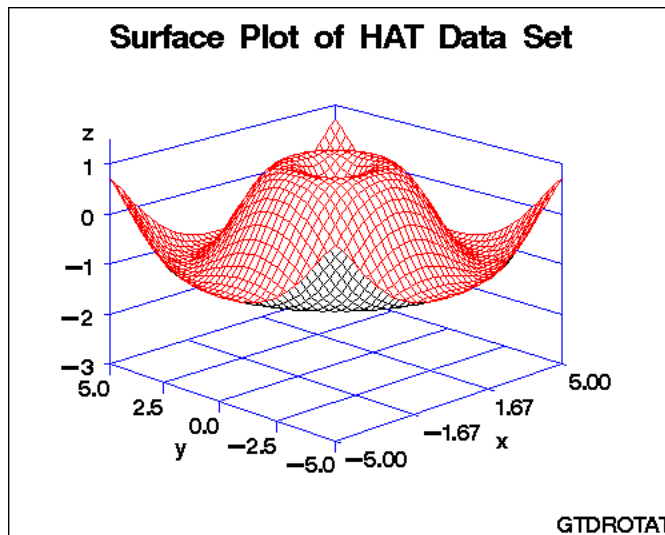
PLOT statement options:

```
CBOTTOM=
CTOP=
GRID
ROTATE=
YTICKNUM=
ZMAX=
ZMIN=
ZTICKNUM=
```

Data set: REFLIB.HAT on page 1315

Sample library member: GTDROTAT

Figure 46.11 A Rotated Surface Plot



This example rotates the surface plot that is shown in Example 4 on page 1318 and enhances its axes by adding reference lines and increasing the number of tick marks on the y and z axes. It also raises the plot above the horizontal x-y plane.

Assign the libref and set the graphics environment.

```

libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss ftitle=swissb htitle=6 htext=4;

```

Define title and footnote.

```

title 'Surface Plot of HAT Data Set';
footnote j=r 'GTDROTAT';

```

Generate the surface plot. GRID draws reference lines for all x, y, and z axis tick marks. ROTATE= specifies a rotation angle of 45°. CTOP= and CBOTTOM= change the colors of the plot's top and bottom surfaces. YTICKNUM= and ZTICKNUM= specify the number of tick marks for the y and z axes. ZMIN= and ZMAX= specify minimum and maximum values for the z axis. Data that exceeds the range of ZMIN-to-ZMAX is displayed at the value of ZMIN or ZMAX. Specifying a ZMIN= value that is below the minimum value in the data effectively raises the plot above the horizontal plane.

```

proc g3d data=reflib.hat;
  plot y*x=z / grid
        rotate=45
        ctop=red
        cbottom=black
        yticknum=5
        zticknum=5
        zmin=-3
        zmax=1;
run;
quit;

```

Example 3: Tilting Surface Plot

Procedure features:

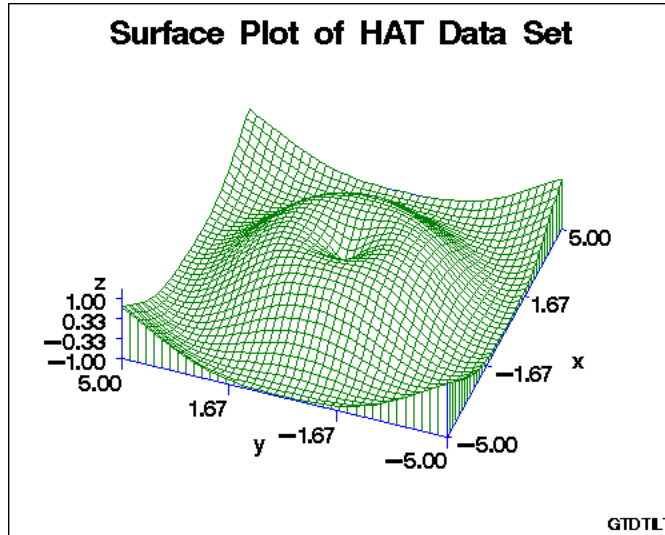
PLOT statement options:

SIDE
TILT=

Data set: REFLIB.HAT on page 1315

Sample library member: GTDTILT

Figure 46.12 A Tilted Surface Plot



This example modifies that shown in Example 1 on page 1314 by tilting the surface plot 15° toward you and adding a side wall.

Assign the libref and set the graphics environment.

```
goptions reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss ftitle=swissb htitle=6 htext=4;
```

Define title and footnote.

```
title 'Surface Plot of HAT Data Set';
footnote j=r 'GTDILT';
```

Generate the surface plot. SIDE draws a side wall for the graph. TILT= specifies a tilt angle of 15° for the plot, which doesn't affect the default rotation of 70° .

```
proc g3d data=work.hat;
  plot y*x=z / side
        tilt=15;
run;
quit;
```

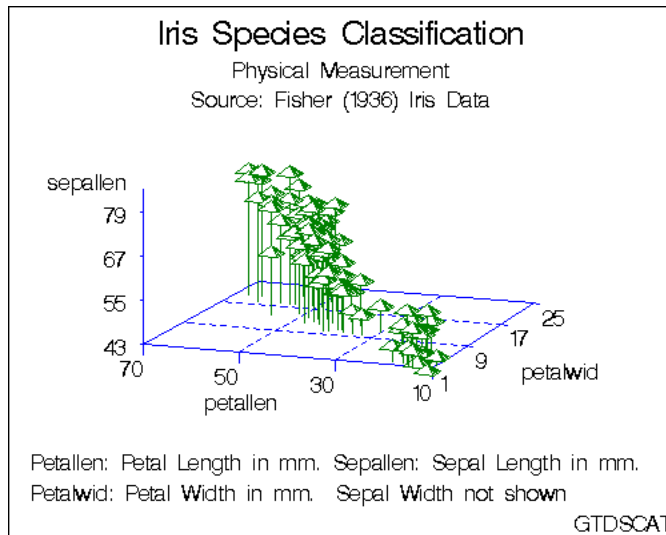
Example 4: Generating a Simple Scatter Plot

Procedure features:

SCATTER statement

Sample library member: GTDSCATR

Figure 46.13 A Scatter Plot with Default Procedure Options



This example shows a scatter plot that examines the results of measuring the petal length, petal width, and sepal length for the flowers of three species of iris. The SCATTER statement in this example relies entirely on procedure defaults, which scale the axes to include all data values, label the axes with the names of the axes variables, divide the axes into three even intervals, rotate the horizontal plane 70° around the z axis and tilt it 70° toward you, and display the plot with the colors that are defined for the colors list. The data points are represented by pyramids, which are connected to the horizontal plane with needles.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
          colors=(black blue green red)
          ftext=swiss ftitle=swissb htitle=6 htext=4;
```

Create data set. REFLIB.IRIS contains petal and sepal measurements for the flowers of three iris species, which are identified by species numbers.

```
data reflib.iris;
  input sepallen sepalwid petallen petalwid spec_no;
  datalines;
50 33 14 02 1
64 28 56 22 3
...more data lines...
63 33 60 25 3
53 37 15 02 1
;
```

Define titles and footnotes.

```

title1 'Iris Species Classification';
title2 'Physical Measurement';
title3 'Source: Fisher (1936) Iris Data';
footnote1 j=l ' Petallen: Petal Length in mm.'
           j=r 'Sepallen: Sepal Length in mm.  ';
footnote2 j=l ' Petalwid: Petal Width in mm.'
           j=r 'Sepal Width not shown          ';
footnote3 j=r 'GTDESCATR';

```

Generate a simple scatter plot.

```

proc g3d data=reflib.iris;
  scatter petallen*petalwid=sepallen;
run;
quit;

```

Example 5: Using Shapes in Scatter Plots

Procedure features:

SCATTER statement options:

```

COLOR=
GRID
NONEEDLE
SHAPE=

```

Other features:

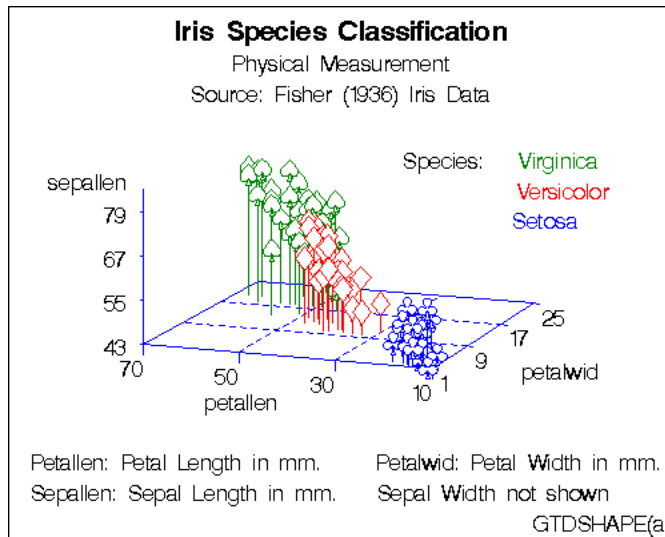
```

DATA step
LABEL statement
NOTE statement

```

Data set: REFLIB.IRIS (see Example 4 on page 1318)**Sample library member:** GTDSHAPE

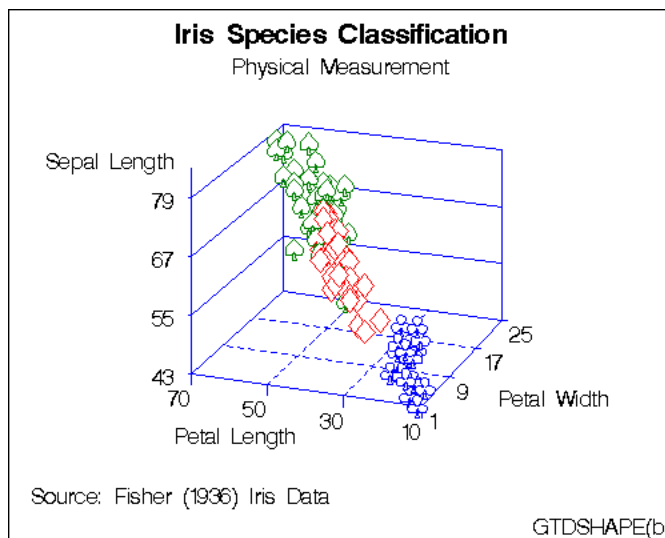
Figure 46.14 A Scatter Plot with Special Shapes



This program modifies that shown in Example 4 on page 1318 to use shape symbols and color to distinguish information for various iris species. It also uses NOTE statements to simulate a plot legend.

The program then generates a second plot to modify the first. As shown by the following output, the second plot request suppresses the needles that connect data points to the horizontal plane, and adds reference lines to make it easier to interpret data values. It also labels the plot axes with descriptive text.

Figure 46.15 A Scatter Plot with Reference Lines and Axis Labels



Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss ftitle=swissb htitle=6 htext=4;
```

Create data set. REFLIB.IRIS2 uses a DATA step to read and modify the REFLIB.IRIS data set. The DATA step adds a variable that identifies the iris species. It also adds two additional variables that store shape and color values for each iris species. These shapes and colors will distinguish iris species in the plot.

```
data reflib.iris2;
  set reflib.iris;
  length species $12. colorval $8. shapeval $8.;
  if spec_no=1 then
    do;
      species='setosa';
      shapeval='club';
      colorval='blue';
    end;
  if spec_no=2 then
    do;
      species='versicolor';
      shapeval='diamond';
      colorval='red';
    end;
  if spec_no=3 then
    do;
      species='virginica';
      shapeval='spade';
      colorval='green';
    end;
run;
```

Define titles and footnotes.

```
title1 'Iris Species Classification';
title2 'Physical Measurement';
title3 'Source: Fisher (1936) Iris Data';
footnote1 j=l ' Petallen: Petal Length in mm.'
          j=r 'Petalwid: Petal Width in mm. ';
footnote2 j=l ' Sepallen: Sepal Length in mm.'
          j=r 'Sepal Width not shown      ';
footnote3 j=r 'GTDSHAPE(a)';
```

Generate the plot. COLOR= specifies the variable that contains color information for the iris species. SHAPE= specifies the variable that contains shape information for the iris species.

```
proc g3d data=reflib.iris2;
  scatter petallen*petalwid=sepallen
        / color=colorval
          shape=shapeval;
```


Create a legend using NOTE statements. The first NOTE statement clears any existing notes. The second NOTE statement identifies the color key used for the different iris species.

```

note;
note j=r 'Species:  ' c=green 'Virginica      '
      j=r c=red 'Versicolor  '
      j=r c=blue 'Setosa      ';
run;

```

Define new title and footnotes.

```

title3;
footnote1 j=l ' Source: Fisher (1936) Iris Data';
footnote2 j=r 'GTDSHAPE(b)';

```

Generate the plot. NONEEDLE suppresses the line drawn from the x-y plane to the plot point. GRID draws reference lines for x, y, and z axis tick marks.

```

proc g3d data=reflib.iris2;
  scatter petallen*petalwid=sepallen
    / noneedle
      grid
      color=colorval
      shape=shapeval;

```

Change the axes labels. To improve axes labels, the LABEL statement associates labels with variable names.

```

label petallen='Petal Length'
      petalwid='Petal Width'
      sepallen='Sepal Length';
run;
quit;

```

Example 6: Rotating a Scatter Plot

Procedure features:

SCATTER statement options

```

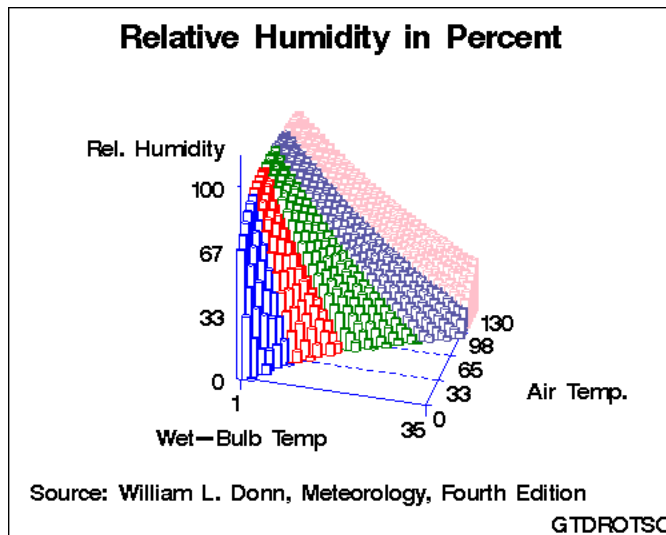
CAXIS=
ROTATE=
SIZE=
XTICKNUM
YTICKNUM=
ZMAX=
ZMIN=
ZTICKNUM=

```

Other features: DATA step

Sample library member: GTDROTSC

Figure 46.16 A Rotated Scatter Plot



This example produces a scatter plot of humidity data. It uses color to distinguish air temperature ranges. The plot is rotated -15° .

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss ftitle=swissb htitle=6 htext=4;
```

Create data set REFLIB.HUMID. The DATA step varies color according to specified air-temperature ranges.

```
data reflib.humid;
  length colorval $ 8.;
  label wtemp='Wet-Bulb Temp';
  label relhum='Rel. Humidity';
  label atemp=' Air Temp.';
  input atemp wtemp relhum;
  if atemp<26 then colorval="blue";
  else if atemp>=26 and atemp<+52 then colorval="red";
  else if atemp>=52 and atemp<+78 then colorval="green";
  else if atemp>=78 and atemp<+104 then colorval="lib";
  else if atemp>104 then colorval="pink ";
  datalines;
```

```

0    1    67
0    2    33
...more data lines...
130  34    29
130  35    28
;

```

Define title and footnotes.

```

title 'Relative Humidity in Percent';
footnote1 j=l ' Source: William L. Donn, Meteorology, Fourth Edition';
footnote2 j=r 'GTDROTSC';

```

Generate the plot. CAXIS= specifies a color for the axis lines and tick marks. ROTATE= specifies a rotation angle for the plot. SIZE= specifies the size of the plot symbols. XTICKNUM=, YTICKNUM=, and ZTICKNUM= specify the number of tick marks for the x, y, and z axes. ZMIN= and ZMAX= specify the minimum and maximum values for the z axis. Z-axis values that exceed the values of the ZMAX= and ZMIN= options will be displayed at the value of ZMAX= or ZMIN=.

```

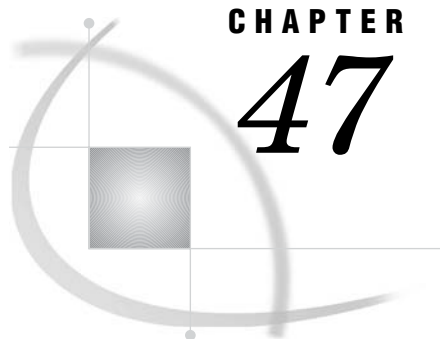
proc g3d data=reflib.humid;
  scatter atemp*wtemp=relhum
    / shape='pillar'
      color=colorval
      caxis=blue
      rotate=-15
      size=.5
      yticknum=5
      xticknum=2
      zticknum=4
      zmin=0
      zmax=100;
run;
quit;

```

References

Fisher, R.A. (1936), "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics*, 7, 179–188.

Watkins, S.L. (1974), "Algorithm 483, Masked Three-Dimensional Plot Program with Rotations (J6)," in *Collected Algorithms from ACM*, New York: Association for Computing Machinery.



CHAPTER

47

The G3GRID Procedure

<i>Overview</i>	1327
<i>Concepts</i>	1329
<i>About the Input Data Set</i>	1329
<i>Multiple Vertical Variables</i>	1329
<i>Horizontal Variables Along a Nonlinear Curve</i>	1329
<i>About the Output Data Set</i>	1329
<i>Interpolation Methods</i>	1329
<i>Default Bivariate Interpolation</i>	1330
<i>Spline Interpolation</i>	1330
<i>Spline Smoothing</i>	1331
<i>Procedure Syntax</i>	1331
<i>PROC G3GRID Statement</i>	1332
<i>GRID Statement</i>	1333
<i>Examples</i>	1336
<i>Example 1: Using the Default Interpolation Method</i>	1336
<i>Example 2: Using Spline Interpolation and a Smoothed Spline</i>	1339
<i>Example 3: Using Partial Spline Interpolation</i>	1342
<i>Example 4: Using Spline Interpolation</i>	1343
<i>References</i>	1346

Overview

The G3GRID procedure processes an existing SAS data set to create a data set that the G3D or GCONTOUR procedure can use to produce three-dimensional surface or contour plots. The procedure creates a data set whose horizontal (x and y) variable values form a complete grid, and it interpolates the value of the vertical (z) variables for each point on the x-y plane.

Using the G3GRID procedure, you can

- Create a rectangular grid of interpolated or smoothed values from irregularly spaced observations for use in a three-dimensional surface or contour plot.
- Complete a rectangular grid of interpolated or smoothed values for an input data set that has an insufficient number of observations to produce a three-dimensional surface or contour plot.
- Interpolate or smooth data for a three-dimensional graph.

The G3GRID procedure does not produce graphics output. Instead, it produces an output data set that you can use as the input data set for the G3D or GCONTOUR procedure.

Figure 47.1 on page 1328 and Figure 47.2 on page 1328 illustrate the effect of the G3GRID procedure on data.

Figure 47.1 on page 1328 shows a collection of data points, where $z=f(x, y)$. These points are randomly distributed and cannot be displayed with a G3D surface plot, although they can be displayed with a scatter plot.

Figure 47.1 Scatter Plot of Data Set Before G3GRID Processing

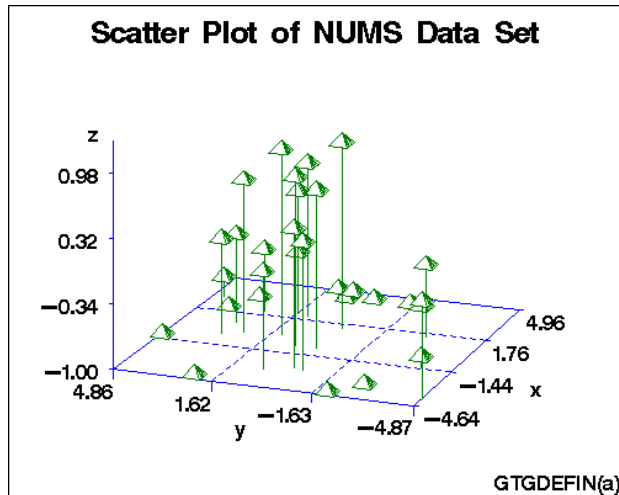
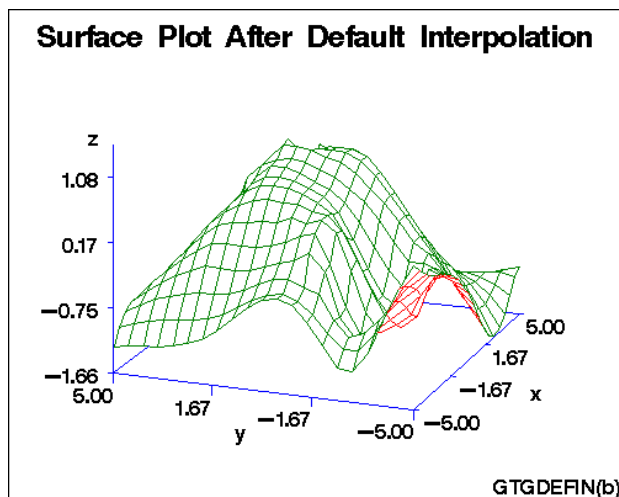


Figure 47.2 on page 1328 shows a surface plot of the data set that is created by a G3GRID interpolation of the original data set shown in Figure 47.1 on page 1328.

Figure 47.2 Surface Plot of Data Set After G3GRID Processing



Note: The evenly distributed horizontal (x, y) data points form a grid for the three-dimensional graph. \triangle

Concepts

About the Input Data Set

The input data set must contain at least three numeric variables:

- two horizontal variables, (x, y)
- one or more vertical variables, z through $z-n$, that will be interpolated or smoothed as if it were a function of the two horizontal variables.

The procedure can process multiple vertical variables for each pair of horizontal variables that you specify. If you specify more than one vertical variable, the G3GRID procedure performs a separate analysis and produces interpolated or smoothed values for each vertical variable. If more than one observation in the input data set has the same values for both horizontal variables, x and y , a warning message is printed, and only the first such point is used in the interpolation.

By default, the interpolation is performed after both variables are similarly scaled because the interpolation methods assume that the scales of x and y are comparable.

Multiple Vertical Variables

In the GRID statement, you can name multiple vertical variables (z through $z-n$) and produce a data set that contains two horizontal variables and multiple vertical variables. You can use the resulting data set to produce plots of the relationships of the two horizontal variables to different vertical variables.

Horizontal Variables Along a Nonlinear Curve

If the points that are generated by the horizontal variables tend to lie along a curve, a poor interpolation or spline may result. In such cases, the vertical variable(s) and one of the horizontal variables should be modeled as a function of the remaining horizontal variable. You can use a scatter plot of the two horizontal variables to help determine the appropriate function.

If the horizontal variable points are collinear, the procedure interpolates the function as constant along lines perpendicular to the line in the plane that is generated by the input data points.

About the Output Data Set

The output data set contains the two horizontal variables, the interpolated or smoothed vertical variables, and the BY variables, if any. If the GRID statement's SMOOTH= option is used, the output data set also contains a variable named `_SMTH_`, with a value equal to that of the smoothing parameter.

You can control both the number of x and y values in the output data set and the values themselves. In addition, you can specify an interpolation method.

Interpolation Methods

The G3GRID procedure can use one of three interpolation methods: bivariate interpolation (the default), spline interpolation, and smoothed spline interpolation.

Default Bivariate Interpolation

Unless you specify the SPLINE option, the G3GRID procedure is an interpolation procedure. That is, it calculates z values for x, y points that are missing from the input grid. The surface that is formed by the interpolated data passes precisely through the data points in the input data set.

This default method of interpolation works best for fairly smooth functions with values given at uniformly distributed points in the plane. If the data points in the input data set are erratic, the default interpolated surface can be erratic.

This default method is a modification of that described by Akima (1978). This method consists of

- 1 dividing the plane into nonoverlapping triangles that use the positions of the available points
- 2 fitting a bivariate fifth degree polynomial within each triangle
- 3 calculating the interpolated values by evaluating the polynomial at each grid point that falls in the triangle.

The coefficients for the polynomial are computed based on

- the values of the function at the vertices of the triangle
- the estimated values for the first and second derivatives of the function at the vertices.

The estimates of the first and second derivatives are computed using the n nearest neighbors of the point, where n is the number specified in the GRID statement's NEAR= option. A Delauney triangulation (Ripley 1981, p. 38) is used for the default method. The coordinates of the triangles are available in an output data set if requested by the OUTTRI= option in the PROC G3GRID statement.

Spline Interpolation

If you specify the SPLINE option, a method is used that produces an interpolation or smoothing that is optimally smooth in a certain sense (Harder and Desmarais 1972, Meinguet 1979). The surface that is generated can be thought of as one that would be formed if a stiff, thin metal plate were forced through or near the given data points. For large data sets, this method is substantially more expensive than the default method.

The function u , formed when you specify the SPLINE option, is determined by letting

$$t_j = (x_j, y_j)$$

$$t = (x, y)$$

and

$$|t - t_j| = \left((x - x_j)^2 + (y - y_j)^2 \right)^{1/2}$$

$$\mathbf{u}(x, y) = \sum_{j=1}^n c_j E(t, t_j) + d_0 + d_1 x + d_2 y$$

where

$$E(s, t) = |s - t| \log(|s - t|).$$

The coefficients c_1, c_2, \dots, c_n and d_1, d_2, d_3 of this polynomial are determined by these equations:

$$(\mathbf{E} + \mathbf{n}\lambda\mathbf{I})\mathbf{c} + \mathbf{T}\mathbf{d} = \mathbf{z}$$

and

$$\mathbf{T}'\mathbf{c} = \mathbf{0}$$

where

E

is the $n \times n$ matrix $E(t_i, t_j)$

I

is the $n \times n$ identity matrix

λ

is the smoothing parameter that is specified in the SMOOTH= option

c

is (c_1, \dots, c_n)

z

is (z_1, \dots, z_n)

d

is (d_1, d_2, d_3)

T

is the $n \times 3$ matrix whose i th row is $(1, x_i, y_i)$.

See Wahba (1979) for more detail.

Spline Smoothing

To produce a smoothed spline, you can use the GRID statement's SMOOTH= option with the SPLINE option. The value or values specified in the SMOOTH= option are substituted for λ in the equation that is described in "Spline Interpolation" on page 1330. A smoothed spline trades closeness to the original data points for smoothness. To find a value that produces the best balance between smoothness and fit to the original data, you can try several values for the SMOOTH= option.

Procedure Syntax

Requirements: Exactly one GRID statement is required.

Reminder: The procedure can include the SAS/GRAPH BY statement.

Supports: Output Delivery System (ODS)

```

PROC G3GRID <DATA=input-data-set>
    <OUT=output-data-set>
    <OUTTRI=output-data-set>;
GRID grid-request <option(s)>;

```

PROC G3GRID Statement

Identifies the input data set. Optionally specifies one or two output data sets.

Requirements: An input data set is required.

Syntax

```

PROC G3GRID <DATA=input-data-set>
    <OUT=output-data-set>
    <OUTTRI=output-data-set>;

```

Options

DATA=*input-data-set*

specifies the SAS data set that contains the variables to process. By default, the procedure uses the most recently created SAS data set.

See also: “SAS Data Sets” on page 29 and “About the Input Data Set” on page 1329

OUT=*output-data-set*

specifies the output data set. The data set contains any BY variables that you specify, the interpolated or smoothed values of the vertical variables (z through $z-n$), and the coordinates for all grid positions on the horizontal (x - y) plane. If you specify smoothing, the output data set also contains a variable named `_SMTH_`, whose value is a smoothing parameter. The observations in this data set are ordered by any variables that you specify with a BY statement. By default, the output of PROC G3GRID creates WORK.DATA1.

Depending on the shape of the original data and the options you use, the output data set may contain values for the vertical (z through $z-n$) values that are outside of the range of the original values in the data set.

Featured in: Example 1 on page 1336

OUTTRI=*output-data-set*

specifies an additional output data set that contains triangular coordinates. The data set will contain any BY variables that you specify, the two horizontal variables giving the horizontal (x - y) plane coordinates of the input points, and a variable named `TRIANGLE` that uses integer values to label the triangles. The observations in this data set are ordered by any variables that you specify with a BY.

The data set contains three observations for each value of the variable `TRIANGLE`. The three observations give the coordinates of the three vertices of the triangle. Points on the convex hull of the input data set of points are also assumed to lie in degenerate triangles whose other vertices are at infinity. The points in the convex hull can be recovered by keeping only those triangles with exactly two missing vertices.

By default, no OUTTRI= data set is produced. OUTTRI= is not valid when you specify the SPLINE option in the GRID statement.

GRID Statement

Specifies the three numeric variables for interpolation or smoothing. Optionally specifies the number of observations (x and y values) in the output data set; output values for the two horizontal variables x,y ; and the interpolation method for the vertical variables.

Requirements: Exactly one grid request is required.

Syntax

GRID *grid-request* </option(s)>;

grid-request must be:

$y*x=z(s)$

grid-request must be

$y*x=z(s)$

option(s) can be one or more options from any or all of the following categories:

- grid options:
 - AXIS1=*ascending-value-list*
 - AXIS2=*ascending-value-list*
 - NAXIS1=*n*
 - NAXIS2=*n*
- interpolation options:
 - JOIN
 - NEAR=*n*
 - NOSCALE
 - PARTIAL
 - SMOOTH=*ascending-value-list*
 - SPLINE

Required Arguments

$y*x=z(s)$

specifies three or more numeric variables from the input data set:

y

is one of the variables that forms the horizontal (x - y) plane.

x

is another of the variables that forms the horizontal (x - y) plane.

$z(s)$

is one or more vertical variables for the interpolation.

Although the GRID statement can specify only two horizontal variables, it can include multiple vertical variables. Separate vertical variables with blanks:

```
grid x*y=z w u v;
```

Options

AXIS1=ascending-value-list

specifies a list of numeric values to assign to the first (y) variable in the grid request for the output data set. Numbers that you specify with this option determine the number of values for y and override a value that you specify with the NAXIS1= option. The *ascending-value-list* must be arranged in ascending order. It can be an explicit list of values, a starting and an ending value with an interval increment, or a combination of both forms:

```
n <...n>
n TO n <BY increment>
n <...n> TO n <BY increment > <n <...n> >
```

Featured in: Example 1 on page 1336 and Example 4 on page 1343

AXIS2=ascending-value-list

specifies a list of numeric values to assign to the second (x) variable in the grid request for the output data set. Numbers that you specify with this option determine the number of values for x and override a value that you specify with the NAXIS2= option. The *ascending-value-list* must be arranged in ascending order. The value can be an explicit list, a starting and an ending value with an interval increment, or a combination of both forms:

```
n <...n>
n TO n <BY increment>
n <...n> TO n <BY increment > <n <...n> >
```

Featured in: Example 1 on page 1336 and Example 4 on page 1343

JOIN

uses a linear interpolation within a set of triangular regions that are formed from the input data set. This interpolation method creates values in the range of the initial values of the vertical variable, but the resulting interpolated surface may not be smooth.

NAXIS1= n

specifies the number of values for the first (y) variable in the grid request for the output data set. You can determine the actual values used for y by taking the minimum and maximum values of y and dividing the range into $n-1$ equal sections. By default, NAXIS1=11.

A value specified with NAXIS1= is ignored if values are also specified with AXIS1=.

NAXIS2= n

specifies the number of values for the second (x) variable in the grid request for the output data set. You can determine the actual values that are used for x by taking the minimum and maximum values of x and dividing the range into $n-1$ equal sections. By default, NAXIS2=11.

A value specified with NAXIS2= is ignored if values are also specified with AXIS2=.

NEAR= n

specifies the number of nearest data points to use for computing the estimates of the first and second derivatives. As NEAR= values become larger, time and computation

costs increase significantly. NEAR= is ignored if you specify SPLINE. The value of n must be greater than or equal to 3. By default, NEAR=3.

If the number of input data points is insufficient for the number that you specify with NEAR=, a smaller number of data points is used.

Featured in: Example 3 on page 1342

NOSCALE

specifies that the x and y variables not be scaled to the same range before interpolation. By default, the interpolation is performed after both variables are similarly scaled because the interpolation methods assume that the scales of x and y are comparable.

PARTIAL

specifies that a spline be used to estimate the derivatives for the biquintic polynomial interpolation. A bivariate spline is fit to the nearest neighbors and used to estimate the needed derivatives. This option produces results that are less smooth than those produced by the SPLINE option and uses fewer computer resources. However, the results produced by PARTIAL are smoother than those that are produced by the default. If you use both PARTIAL and the SPLINE option, PARTIAL is ignored.

Featured in: Example 3 on page 1342

SMOOTH=*ascending-value-list*

specifies a list of numbers for smoothing parameters. Use this option only when you also use the SPLINE option. The *ascending-value-list* must be arranged in ascending order. The value can be an explicit list, a starting and an ending value with an interval increment, or a combination of both forms:

n <... n >

n TO n <BY *increment*>

n <... n > TO n <BY *increment* > < n <... n > >

For each value λ of the smoothing parameter, a function $u(x, y)$ is formed that minimizes

$$\frac{1}{n} \sum_{j=1}^n (\mathbf{u}(x_j, y_j) - z_j)^2 + \lambda \sum_{j=0}^2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} dx dy$$

where n is the number of data points and the pairs (x_j, y_j) are the available points, with corresponding function values z_j (Wahba 1979).

The higher the value of the smoothing parameter, the smoother the resulting interpolation. The lower the smoothing parameter, the closer the resulting surface is to the original data points. A smoothing parameter of 0 produces the same results as the SPLINE option without SMOOTH=.

This procedure repeats for each value of the smoothing parameter. The output data set that you specify in the OUT= option contains the interpolated values, the values of the grid points, and the values of the smoothing parameter in the variable `_SMTH_`. The output data set contains a separate grid for each value of the smoothing parameter.

Featured in: Example 2 on page 1339

SPLINE

specifies the use of a bivariate spline (Harder and Desmarais 1972, Meinguet 1979) to interpolate or to form a smoothed estimate if you also use the SMOOTH= option. This option results in the use of an order n^3 algorithm, where n is the number of

input data points. Consequently, this method can be time-consuming. If you use more than 100 input points, the procedure may use excessive time.

Featured in: Example 2 on page 1339 and Example 4 on page 1343

Controlling Observations in the Output Data Set

By default, the G3GRID procedure produces a data set with 121 observations for combinations of 11 values for each of the horizontal variables, x and y . To create a data set with a different number of observations, use the GRID statement's NAXIS1= or NAXIS2= options to specify the number of the values of y or x , respectively. Or, use the GRID statement's AXIS1= or AXIS2= options to specify the actual values for y or x , respectively.

Table 47.1 on page 1336 shows the number of observations that will be in the output data set if you use any of these options.

Table 47.1 Number of Observations Contained in the Output Data Set

Options Specified	Number of Observations in Output Data Set
None	121
AXIS1=	(number of values for AXIS1=) * 11
AXIS2=	(number of values for AXIS2=) * 11
NAXIS1=	(value of NAXIS1=) * 11
NAXIS2=	(value of NAXIS2=) * 11
AXIS1=, AXIS2=	(number of values for AXIS1=) * (number of values for AXIS2=)
AXIS1=, NAXIS1=	(number of values for AXIS1=) * 11
AXIS1=, NAXIS2=	(number of values for AXIS1=) * (value of NAXIS2=)
AXIS2=, NAXIS1=	(number of values for AXIS2=) * (value of NAXIS1=)
AXIS2=, NAXIS2=	(number of values for AXIS2=) * 11
NAXIS1=, NAXIS2=	(value of NAXIS1=) * (value of NAXIS2=)

If you specify multiple smoothing parameters, the number of observations in the output data set will be the number shown in Table 47.1 on page 1336 multiplied by the number of smoothing values that you specify in the SMOOTH= option. If you use BY-group processing, multiply the number in the table by the number of BY groups.

Depending on the shape of the original data and the options that you specify, the output data set may contain values for the vertical (z) values that are outside of the range of the original values in the data set.

Examples

Example 1: Using the Default Interpolation Method

Procedure features:

G3GRID statement options:

OUT=

GRID statement options:

AXIS1=

AXIS2=

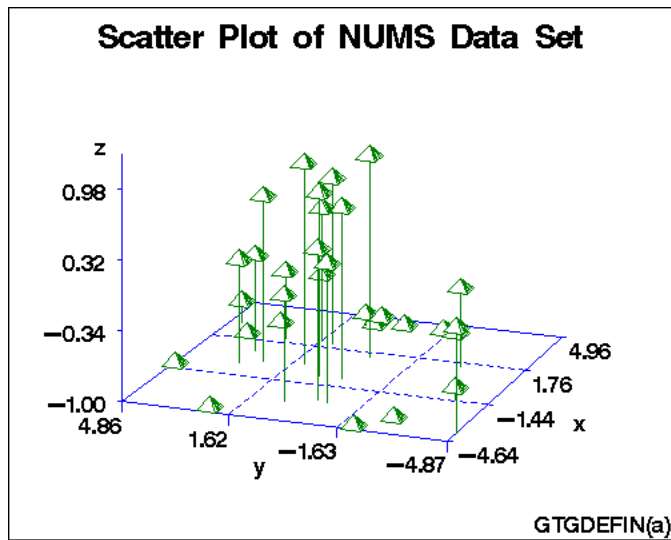
Other features:

DATA step

G3D procedure

Sample library member: GTGDEFIN

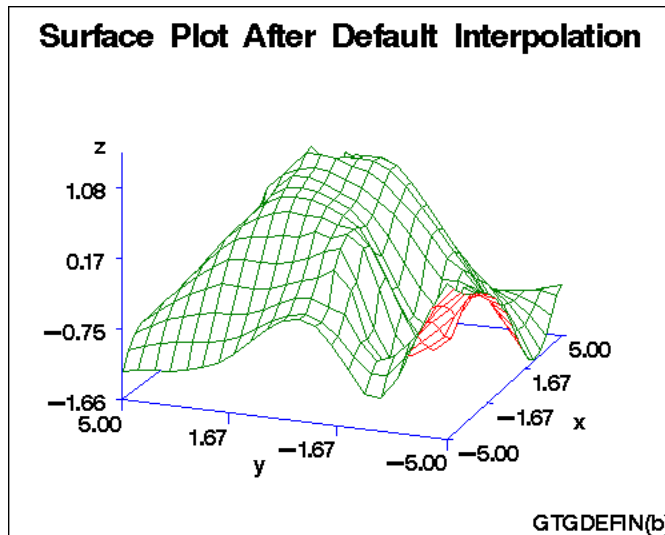
Figure 47.3 A Scatter Plot Showing Data Before Interpolation



This example demonstrates the default interpolation method that is used by the GRID statement. The example first generates a scatter plot of random data to show the concentration of data values before processing with the G3GRID procedure. The original data do not contain enough combinations of x , y , and z values to generate a surface plot with the G3D procedure, or a contour plot with the GCONTOUR procedure.

The example then runs the G3GRID procedure to interpolate additional x , y , and z values. Because no interpolation method is specified, the default interpolation method is used. The resulting output data set is used as input to the G3D procedure, which generates the surface plot shown in the following output.

Figure 47.4 A Surface Plot Generated After Interpolation

**Assign the libref and set the graphics environment.**

```
libname reflib 'SAS-data-library';
options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss ftitle=swissb htitle=6 htext=3;
```

Create data set. REFLIB.NUMS uses a set of randomly sampled points to create the data used in this and all remaining examples in this chapter.

```
data reflib.num;
  keep x y z;
  do i=1 to 30;
    x=10*ranuni(33)-5;
    y=10*ranuni(35)-5;
    z=sin(sqrt(x*x+y*y));
    output;
  end;
run;
```

Define title and footnote.

```
title 'Scatter Plot of NUMS Data Set';
footnote j=r 'GTGDEFIN(a)';
```

Generate the scatter plot.

```
proc g3d data=reflib.num;
  scatter y*x=z;
```



```
run;
```

Process points with PROC G3GRID. OUT= on G3GRID specifies a name for a temporary output data set. GRID specifies the variables Y*X=Z for the output data set. AXIS@@@ 1

```
proc g3grid data=reflib.numms out=default;
  grid y*x=z / axis1=-5 to 5 by .5
             axis2=-5 to 5 by .5;
run;
```

Define new title and footnote.

```
title 'Surface Plot after Default Interpolation';
footnote j=r 'GTGDEFIN(b)';
```

Generate a surface plot. The G3D procedure uses as its input data set the G3GRID procedure's output data set.

```
proc g3d data=default;
  plot y*x=z;
run;
quit;
```

Example 2: Using Spline Interpolation and a Smoothed Spline

Procedure features:

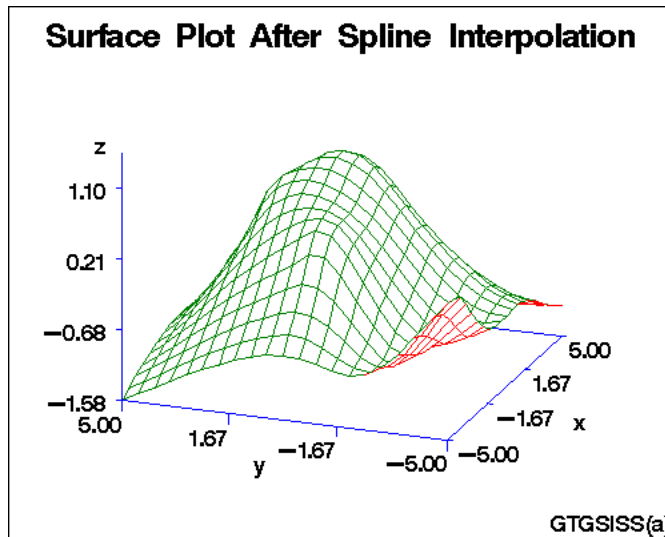
GRID statement options:

SMOOTH=
SPLINE

Data set: REFLIB.NUMMS (see Example 1 on page 1336)

Sample library member: GTGSISS

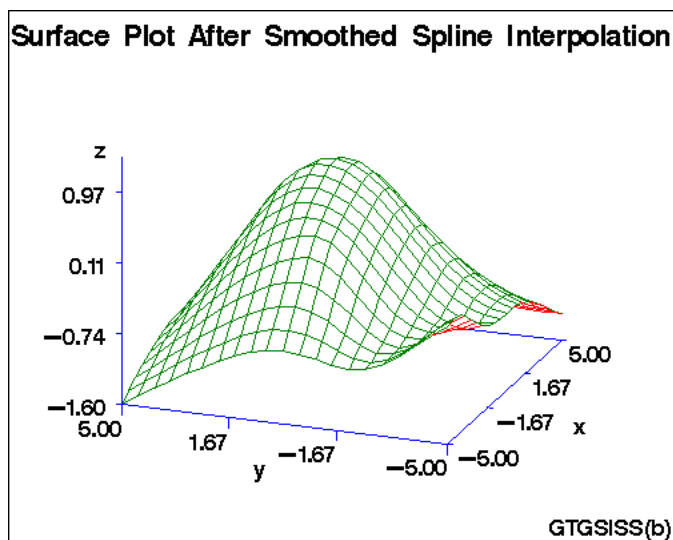
Figure 47.5 A Surface Plot Generated After Spline Interpolation



This example extends Example 1 on page 1336 to specify a spline interpolation method on the GRID statement. The output data set, when used in PROC G3D, generates a smoother surface plot than the surface plot that results from the default interpolation.

This example then specifies a smoothed spline interpolation method on the GRID statement. As shown by the following output, the resulting surface plot is smoother still.

Figure 47.6 A Surface Plot Generated After Smoothed Spline Interpolation



Assign the libref and set the graphics environment.

```
libname rellib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
```

```

colors=(black blue green red)
ftext=swiss ftitle=swissb htitle=5 htext=3;

```

Define title and footnote.

```

title 'Surface Plot After Spline Interpolation';
footnote j=r 'GTGSISS(a)';

```

Process points with PROC G3GRID. SPLINE specifies the bivariate spline method for the data set interpolation.

```

proc g3grid data=reflib.numms out=spline;
  grid y*x=z / spline
          axis1=-5 to 5 by .5
          axis2=-5 to 5 by .5;
run;

```

Generate a surface plot.

```

proc g3d data=spline;
  plot y*x=z ;
run;

```

Define title and footnote for second plot.

```

title 'Surface Plot After Smoothed Spline Interpolation';
footnote j=r 'GTGSISS(b)';

```

Process points with PROC G3GRID. SMOOTH= specifies the smoothing parameter to use during spline interpolation.

```

proc g3grid data=reflib.numms out=smoothed;
  grid y*x=z / spline
          smooth=.05
          axis1=-5 to 5 by .5
          axis2=-5 to 5 by .5;
run;

```

Generate a surface plot.

```

proc g3d data=smoothed;
  plot y*x=z;
run;
quit;

```

Example 3: Using Partial Spline Interpolation

Procedure features:

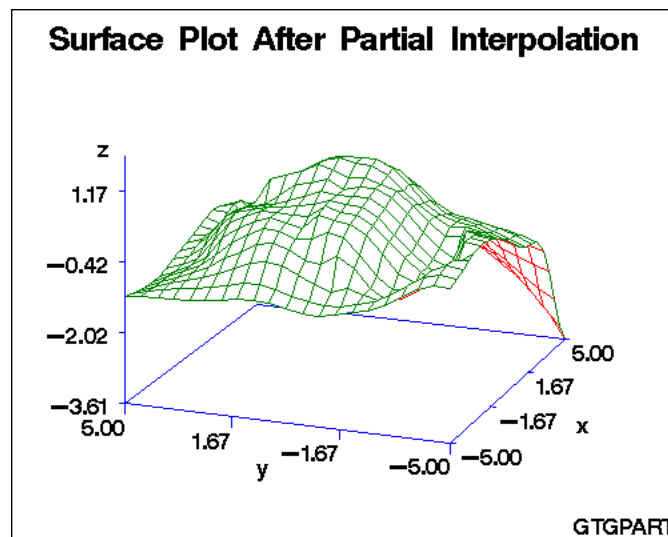
GRID statement options:

NEAR
PARTIAL

Data set: REFLIB.NUMS (see Example 1 on page 1336)

Sample library member: GTGPART

Figure 47.7 A Surface Plot Generated After Partial Interpolation



This example specifies a partial spline interpolation method on the GRID statement, using eight nearest neighbors for computing the estimates of the first and second derivatives. The output data set, when used in PROC G3D, generates a smoother surface plot than the surface plot that results from the default interpolation shown in Example 1 on page 1336, but not as smooth as the surface plot that results from the spline interpolation shown in Example 2 on page 1339.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
options reset=global gunit=pct border cback=white
        colors=(black blue green red)
        ftext=swiss ftitle=swissb htitle=6 htext=3;
```

Define title and footnote.

```
title 'Surface Plot after Partial Interpolation';
footnote j=r 'GTGPART';
```

Process points with PROC G3GRID. PARTIAL specifies that a spline be used to estimate the derivatives for the biquintic polynomial interpolation. NEAR= specifies the number of nearest neighbors to be used for computing the estimates of the first and second derivatives.

```
proc g3grid data=reflib.numms out=partial;
  grid y*x=z / partial
      near=8
      axis1=-5 to 5 by .5
      axis2=-5 to 5 by .5;
run;
```

Generate the surface plot.

```
proc g3d data=partial;
  plot y*x=z;
run;
quit;
```

Example 4: Using Spline Interpolation

Procedure features:

GRID statement options:

AXIS1=
 AXIS2=
 SPLINE

Data set: REFLIB.NUMMS (see Example 1 on page 1336)

Sample library member: GTGSPLIN

Figure 47.8 A Contour Plot Generated After Default Interpolation

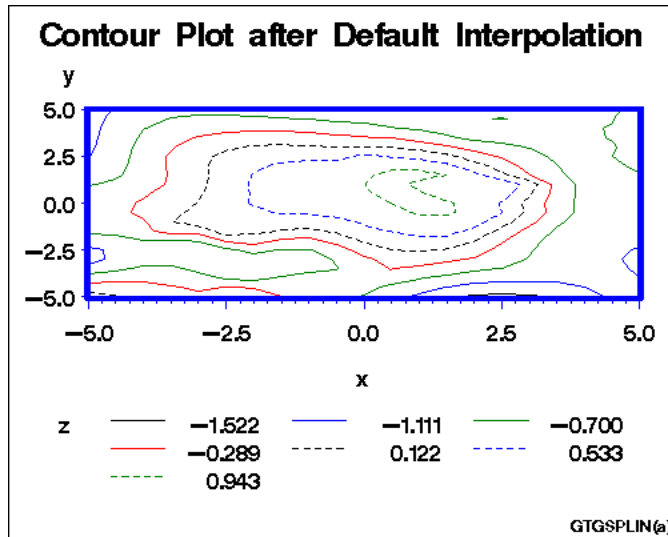
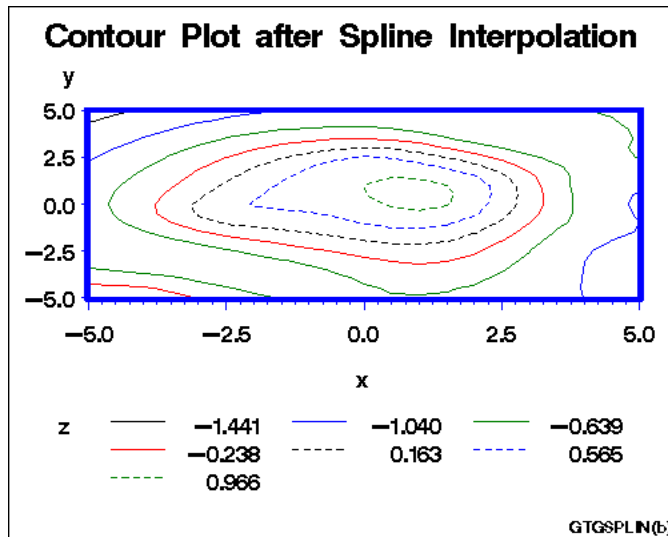


Figure 47.9 A Contour Plot Generated After Spline Interpolation



This example demonstrates the default and spline interpolation methods when used by the GCONTOUR procedure to generate contour plots from the resulting output data sets.

Assign the libref and set the graphics environment.

```
libname reflib 'SAS-data-library';
goptions reset=global gunit=pct border cback=white
         colors=(black blue green red)
         ftext=swiss ftitle=swissb htitle=6 htext=3;
```

Define title and footnote.

```
title 'Contour Plot after Default Interpolation';
footnote j=r 'GTGSPLIN(a)';
```

Define axis characteristics.

```
axis1 width=3;
```

Process points with PROC G3GRID.

```
proc g3grid data=reflib.nums out=numdef;
  grid y*x=z / axis1=-5 to 5 by .5
             axis2=-5 to 5 by .5;
run;
```

Generate the contour after default interpolation.

```
proc gcontour data=numdef;
  plot y*x=z / haxis=axis1 vaxis=axis1;
run;
```

Define new title and footnote.

```
title 'Contour Plot after Spline Interpolation';
footnote j=r 'GTGSPLIN(b)';
```

Process points with PROC G3GRID. SPLINE specifies the bivariate spline method for the data set interpolation.

```
proc g3grid data=reflib.nums out=numspl;
  grid y*x=z / spline
             axis1=-5 to 5 by .5
             axis2=-5 to 5 by .5;
run;
```

Show the contour after spline interpolation.

```
proc gcontour data=numspl;
  plot y*x=z / haxis=axis1 vaxis=axis1;
run;
quit;
```

References

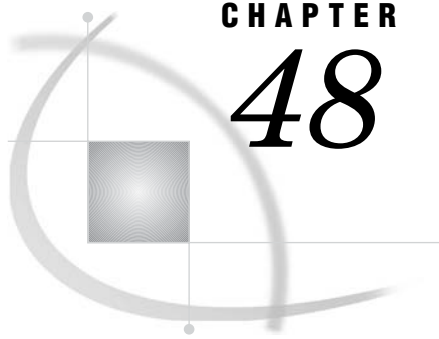
Akima, Hiroshi (1978), "A Method of Bivariate Interpolation and Smooth Surface Fitting for Irregularly Distributed Data Points," *ACM Transaction on Mathematical Software*, 4, 148–159.

Harder, R.L. and Desmarais, R.N. (1972), "Interpolation Using Surface Splines," *Journal of Aircraft*, 9, 189–191.

Meinguet, Jean (1979), "Multivariate Interpolation at Arbitrary Points Made Simple," *Journal of Applied Mathematics and Physics*, 30, 292–304.

Ripley, B.D. (1981), *Spatial Statistics*, New York: John Wiley & Sons, Inc.

Wahba, Grace (1979), "How to Smooth Curves and Surfaces with Splines and Cross-validation," in U.S. Army Research Office Report 79–2, *Proceedings of the 24th Conference on the Design of Experiments*.



CHAPTER

48

The MAPIMPORT Procedure

Overview	1347
Procedure Syntax	1348
PROC MAPIMPORT Statement	1348
Examples	1349
Example 1: Including All Variables from the SHP Shapefile	1349
Example 2: Including Selected Variables from the SHP Shapefile	1350
Example 3: Excluding a Variable from the SHP Shapefile	1350
Example 4: Including Selected Variables from the DBF Shapefile	1350

Overview

The MAPIMPORT procedure enables you to import ESRI shapefiles (spatial data formats) and process the SHP files into SAS/GRAPH traditional map data sets. See “About Traditional Data Sets” on page 999 for more information. The shapefiles file types are described in the following table:

Table 48.1 Shapefiles File Types

File Extension	Description
.dbf	identification information (field-identifier names and values) assigned to specific polygon(s)
.shx	shape information for the polygon(s) that compose the map. <i>Note:</i> These files are used with .shp files and cannot be imported by themselves. △
.shp	combines the shape information for the polygon(s) that compose the map and the identification information (field-identifier names and values) assigned to the specific polygon(s)

Procedure Syntax

Requirements: The name and location of an output data set and the complete path for the input data file.

Reminder: The single quotes surrounding field identifiers are optional when the field identifiers follow the SAS naming convention. Single quotes are required for field identifiers that are non-standard SAS names. When field identifiers placed in single quotes are non-standard SAS names, the field identifiers are converted to a standard SAS name in the traditional map data set. For more information about the standard SAS naming convention, see names in the SAS Language in *SAS Language Reference: Concepts*. For more information on how invalid field identifiers placed in single quotes are renamed, see the SAS System option VALIDVARNAME in *SAS/ACCESS for Relational Databases: Reference*.

PROC MAPIMPORT

```
OUT= traditional-map-data-set
DATAFILE= 'path-to-shapefile'
<CONTENTS>;
<CREATE_ID_>;
<SELECT <'>field-identifier-1<'><...<'>field-identifier-n<'>>>;
<EXCLUDE <'>field-identifier-1<'><...<'>field-identifier-n<'>>>;
<RENAME <'>field-identifier-1<'>=SAS-variable-name-1<...<'>field-identifier-
n<'>=SAS-variable-name-n>>;
```

PROC MAPIMPORT Statement

Identifies the input ESRI SHAPEFILE and converts this map into a SAS/GRAPH traditional map data set.

Requirements: The name and location of an output data set and the complete path for the input data file.

PROC MAPIMPORT

```
OUT= traditional-map-data-set
DATAFILE= 'path-to-shapefile'
<CONTENTS>;
<CREATE_ID_>;
<SELECT <'>field-identifier-1<'><...<'>field-identifier-n<'>>>;
<EXCLUDE <'>field-identifier-1<'><...<'>field-identifier-n<'>>>;
<RENAME <'>field-identifier-1<'>=SAS-variable-name-1<...<'>field-identifier-
n<'>=SAS-variable-name-n>>;
```

Required Arguments

OUT= traditional-map-data-set

specifies the name of the traditional map data set created.

DATAFILE= 'path-to-shapefile'

specifies the path and filename of the shapefile that is read and processed.

Note: By default, all of the fields in a shapefile are included in the traditional map data set. To only include specific fields in the traditional map data set, use the SELECT statement. To exclude specific fields from being in the traditional map data set, use the EXCLUDE statement . △

Optional Argument**CONTENTS**

displays information about the SHAPEFILE, including field identifier names and types.

CREATE_ID_

creates a map ID variable named `_ID_` with a unique value for each polygon in the map. This variable will be created automatically if the .dbf file is missing.

Optional Statements**SELECT *field-identifier-n***

selects only the specified fields in the SHAPEFILE to be included in the traditional map data set.

EXCLUDE *field-identifier-n*

excludes the specified fields in the SHAPEFILE from being in the traditional map data set.

RENAME *field-identifier-n= SAS-variable-name-1*

renames the specified fields in the traditional map data set. By default the field identifiers in the SHAPEFILE will be the SAS variable names in the traditional map data set.

Note: Field identifiers that are invalid SAS variable names must be placed in single quotes. A field identifier placed in single quotes will be automatically renamed to a valid SAS variable name using the SELECT or EXCLUDE statement. To change the field identifier to a specific valid SAS variable name, use the RENAME statement. △

Examples

The following examples use shapefiles with the .shp and .dbf extensions. Replace the shapefiles locations, filenames, and field-identifiers with information from your shapefiles to run these examples.

Example 1: Including All Variables from the SHP Shapefile

In the following example, World30.shp contains polygons that compose a political boundary world map. All the field identifiers in the World30.shp file will be included in the traditional map data set, MYWORLD, created in the SASUSER library.

```
PROC MAPIMPORT OUT=sasuser.myworld DATAFILE='C:\world30.shp';
run;
```

Example 2: Including Selected Variables from the SHP Shapefile

In the following example, the STATES.SHP file contains polygons that compose the political boundaries of a U.S. states map. Only the STATE_FIPS (the state FIPS codes), STATE_NAME (the state name), and STATE_ABBR (the two letter state abbreviation) variables are included in the traditional map data set, MYSTATES, which will be created in the SASUSER library. STATE_FIPS will be renamed FIPS, STATE_NAME will be renamed STATE, and STATE_ABBR will be renamed ABBREV in the MYSTATES map data set.

```
PROC MAPIMPORT OUT=sasuser.mystates DATAFILE='C:\states.shp';
  SELECT STATE_FIPS STATE_NAME STATE_ABBR;
  RENAME STATE_FIPS=FIPS STATE_NAME=STATE STATE_ABBR=ABBREV;
run;
```

Example 3: Excluding a Variable from the SHP Shapefile

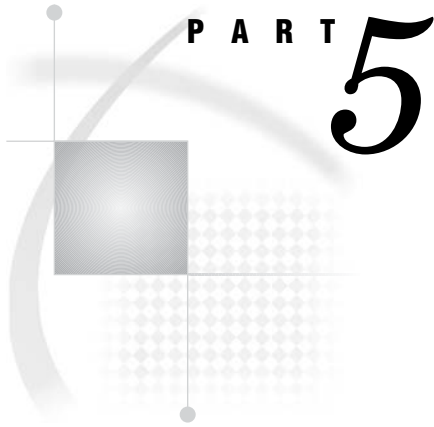
In the following example, the STATES.SHP file contains polygons that compose the political boundaries of a U.S. state map. The variable OTHER is excluded from the traditional map data set, MYSTATES2, created in the SASUSER library.

```
PROC MAPIMPORT OUT=sasuser.mystates2 DATAFILE='C:\states.shp';
  EXCLUDE OTHER;
run;
```

Example 4: Including Selected Variables from the DBF Shapefile

In the following example, the STATES.DBF file contains the identification information (field-identifier names and values) applied to the U.S. states polygon map. Only the STATE_FIPS (the state FIPS codes), STATE_NAME (the state names), and STATE_ABBR (the two letter state abbreviations) variables are included in the traditional map data set, MYDATA, which will be created in the SASUSER library. STATE_FIPS will be renamed FIPS, STATE_NAME will be renamed STATE, and STATE_ABBR will be renamed ABBREV in the MYDATA map data set.

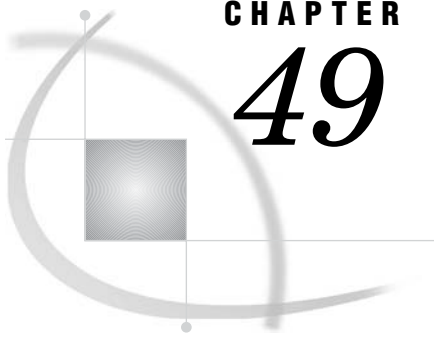
```
PROC MAPIMPORT OUT=sasuser.mydata DATAFILE='C:\states.dbf';
  SELECT STATE_FIPS STATE_NAME STATE_ABBR;
  RENAME STATE_FIPS=FIPS STATE_NAME=STATE STATE_ABBR=ABBREV;
run;
```



The Data Step Graphics Interface

Chapter 49 **The DATA Step Graphics Interface** 1353

Chapter 50 **DATA Step Graphics Interface Dictionary** 1401



CHAPTER

49

The DATA Step Graphics Interface

<i>Overview</i>	1354
<i>Syntax</i>	1355
<i>Requirements</i>	1356
<i>Applications of the DATA Step Graphics Interface</i>	1356
<i>Enhancing Existing Graphs</i>	1356
<i>Creating Custom Graphs</i>	1356
<i>Using the DATA Step Graphics Interface</i>	1357
<i>Summary of Use</i>	1357
<i>Producing and Storing DSGI Graphs</i>	1357
<i>Structure of DSGI Data Sets</i>	1358
<i>Using SAS/GRAPH Global Statements with DSGI</i>	1358
<i>Operating States</i>	1359
<i>The Current Window System</i>	1359
<i>Debugging DSGI Programs</i>	1360
<i>DSGI Graphics Summary</i>	1360
<i>DSGI Functions</i>	1360
<i>DSGI Routines</i>	1364
<i>Creating Simple Graphics with DSGI</i>	1367
<i>Setting Attributes for Graphics Elements</i>	1368
<i>How Operating States Control the Order of DSGI Statements</i>	1370
<i>Functions That Change the Operating State</i>	1370
<i>Order of Functions and Routines</i>	1371
<i>Bundling Attributes</i>	1373
<i>Attributes That Can Be Bundled for Each Graphics Primitive</i>	1373
<i>Assigning Attributes to a Bundle</i>	1374
<i>Selecting a Bundle</i>	1375
<i>Defining Multiple Bundles for a Graphics Primitive</i>	1375
<i>How DSGI Selects the Value of an Attribute to Use</i>	1375
<i>Disassociating an Attribute from a Bundle</i>	1376
<i>Using Viewports and Windows</i>	1376
<i>Defining Viewports</i>	1377
<i>Clipping around Viewports</i>	1377
<i>Defining Windows</i>	1377
<i>Activating Transformations</i>	1378
<i>Inserting Existing Graphs into DSGI Graphics Output</i>	1379
<i>Generating Multiple Graphics Output in One DATA Step</i>	1380
<i>Processing DSGI Statements in Loops</i>	1380
<i>Examples</i>	1381
<i>Vertically Angling Text</i>	1381
<i>Changing the Reading Direction of the Text</i>	1384
<i>Using Viewports in DSGI</i>	1385

<i>Scaling Graphs by Using Windows</i>	1388
<i>Enlarging an Area of a Graph by Using Windows</i>	1391
<i>Using GASK Routines in DSGI</i>	1394
<i>Generating a Drill-down Graph Using DSGI</i>	1395
<i>See Also</i>	1399

Overview

The DATA Step Graphics Interface (DSGI) enables you to create graphics output within the DATA step or from within an SCL application. Through DSGI, you can call the graphics routines used by SAS/GRAPH software to generate an entire custom graph or to add features to an existing graph. You can use DSGI to write a custom graphics application in conjunction with all the power of the programming statements accessible by the DATA step.

DSGI provides many of the same features as the Annotate facility, but it also has many advantages over the Annotate facility.

- You can use DSGI functions and routines through SCL.
- You can save disk space. DSGI graphics can be generated through the DATA step without creating an output data set. The graphics output is stored as a catalog entry in the catalog you select and, optionally, is displayed after the DATA step is submitted.
- DSGI generates graphics faster than the Annotate facility. With the Annotate facility, you must first create a data set and then submit a PROC step to display the graphics output. In DSGI, you eliminate the PROC step because the graphics output is generated after the DATA step.
- DSGI supports viewports and windows, which enable you to specify the dimensions, position, and scale of the graphics output. They also allow you to include multiple graphs in the same graphics output.

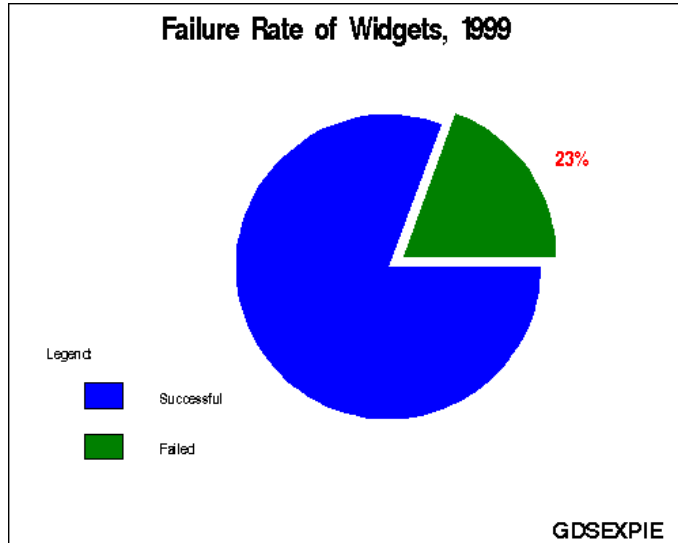
You should consider using the Annotate facility for enhancing procedure output and using DSGI for creating custom graphics without using a graphics procedure.

DSGI is based upon the Graphics Kernel System (GKS) standard, although it does not follow a strict interpretation, nor is it implemented on a particular level of GKS. GKS was used to provide a recognizable interface to the user. Because of its modularity, the standard allows for enhancements to DSGI without the side effect of converting programs between versions of SAS/GRAPH software.

This chapter explains the concepts used to create graphics output with DSGI. The discussion provides an overview of the functions and routines used in DSGI. For complete details of each function and routine, see Chapter 50, “DATA Step Graphics Interface Dictionary,” on page 1401.

Display 49.1 on page 1355 shows a pie chart that was created entirely with DSGI functions. Display 49.2 on page 1355 is an example of a text slide that was created with DSGI statements.

Display 49.1 Exploded Pie Chart Generated with the DSGI



Display 49.2 Text Slide Created Using the DSGI

Production Information for Widgets Assembly
JAN 1999 – JUN 1999

MONTH	TEMP	SPEED	PERCENT FAILED
JAN	100	90	4.950
FEB	110	90	4.792
MAR	120	90	5.042
APR	130	90	5.700
MAY	140	90	6.766
JUN	150	90	8.240

GDSSLIDE

Syntax

DSGI uses GASK routines and functions to draw graphics elements. These statements have the following syntax:

```
CALL GASK(operator, arguments);
return-code-variable=function-name (operator, arguments);
```

where

arguments are the additional required variables or values for the routine or function.

<i>return-code-variable</i>	is an arbitrary name and can be any numeric variable name. It will hold the return code upon execution of the function.
<i>function-name</i>	is the DSGI command you want to execute and must be one of the following: GDRAW, GINIT, GPRINT, GRAPH, GSET, or GTERM.
<i>operator</i>	is a character string that names the function you either want to submit or for which you want the current settings. When used with functions, <i>operator</i> can take different values depending on <i>function-name</i> .

Requirements

When using DSGI statements, the following formats for arguments must be used:

- All x and y coordinates are expressed in units of the current window system. (See “The Current Window System” on page 1359 for details.)
- The arguments used with DSGI functions can be expressed as either constants or variables. The arguments used with GASK routines must be variable names since values are returned through them. See Chapter 50, “DATA Step Graphics Interface Dictionary,” on page 1401 for a complete explanation of each argument used with DSGI functions and routines.
- All arguments that are character constants must be enclosed in either single or double quotation marks.

Applications of the DATA Step Graphics Interface

With the DATA Step Graphics Interface you can

- enhance existing graphs
- create custom graphs.

Enhancing Existing Graphs

You can use DSGI to enhance graphs that were previously generated by using SAS/GRAPH procedures. You can add text and other graphics elements. You can also alter the appearance of the existing graph by scaling or reducing it. To enhance a graph produced by a SAS/GRAPH graphics procedure, you must insert the existing graph into graphics output being generated with DSGI.

To insert a graph, you must provide DSGI with the following information:

- the catalog in which the existing graph is located
- the name of the existing graph
- the coordinates of the place in the graphics output where you want to insert the existing graph
- a square coordinate system ((0,0) to (100,100))
- the statements to draw enhancements to the existing graph.

The coordinates that DSGI uses to position existing graphs, enhancements to that graph, or graphics elements are based on units of percent of the window system currently defined. See “Using Viewports and Windows” on page 1376.

Creating Custom Graphs

You can produce custom graphs with DSGI without using a data set to produce the graphics output. DSGI enables you to generate

- arcs
- bars
- ellipses
- elliptical arcs
- lines
- markers
- pie slices
- polygons (filled areas)
- text.

To create custom graphs, you must provide the system with the following information:

- DSGI statements to draw graphics elements
- the coordinates of the graphics elements in the output.

In addition, you can specify the color, pattern, size, style, and position of these graphics elements.

Using the DATA Step Graphics Interface

The following sections provide general information about using DSGI, including general steps for using DSGI, how to produce and store graphs, how the data sets used with DSGI are structured, how SAS/GRAPH global statements can be used with DSGI, and how to debug DSGI programs. The sections also explain some of the basic concepts of DSGI, including information about operating states and windowing systems.

Summary of Use

To generate graphics output using DSGI, you generally follow these steps:

- 1 On a grid that matches the dimensions of the graphics output, sketch the output you want to produce.
- 2 Determine the coordinates of each graphics element.
- 3 In the DATA step, write the program to generate the graphics output. The basic steps are to
 - a initialize DSGI
 - b open a graphics segment
 - c generate graphics elements
 - d close the graphics segment
 - e end DSGI.
- 4 Submit the DATA step with a final RUN statement to display the output.

Note: The DISPLAY graphics option must be in effect for the graphics output to be displayed. See Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261 for more information about the DISPLAY graphics option. △

Producing and Storing DSGI Graphs

When you create or enhance graphs with DSGI, the DSGI graphics are displayed and stored as part of the graphics output. When you execute the DATA step, DSGI creates a catalog entry using the name from the GRAPH('CLEAR', . . .)function.

By default, DSGI uses the name DSGI if you have not specified a name with the `GRAPH('CLEAR', . . .)` function. By default, the catalog entry is stored in `WORK.GSEG` unless you specify another catalog with the `GSET('CATALOG', . . .)` function.

If you generate another graph using a name that matches an existing catalog entry in the current catalog, DSGI uses the default naming conventions for the catalog entry. See “Names and Descriptions of Catalog Entries” on page 55 for a description of the conventions used to name catalog entries.

If you want to store your output in a permanent library or in a different temporary catalog, you must use the `GSET('CATALOG', . . .)` function. This function allows you to specify the libref and catalog name for the output catalog. Before you use the `GSET('CATALOG', . . .)` function, you must allocate the libref using a `LIBNAME` statement.

You can redisplay DSGI graphics output stored in catalog entries using the `GREPLAY` procedure or the `GRAPH` window.

Structure of DSGI Data Sets

The DSGI `DATA` step is usually not written to produce an output data set. Unlike data sets created by the Annotate facility, which contain observations for each graphics element drawn, DSGI does not usually create an observation for each graphics primitive. Only variables created in the `DATA` step are written to the output data set.

You can output as many observations to the data set as you want. To output these values, you must use the `OUTPUT` statement. You can also use any other valid SAS `DATA` step statements in a DSGI `DATA` step. See *SAS Language Reference: Dictionary* for information about the statements used in the `DATA` step.

Using SAS/GRAPH Global Statements with DSGI

You can use some SAS/GRAPH global statements with DSGI programs. DSGI recognizes `FOOTNOTE`, `GOPTIONS`, and `TITLE` statements; however, it ignores `AXIS`, `LEGEND`, `NOTE`, `PATTERN`, and `SYMBOL` statements.

`FOOTNOTE` and `TITLE` statements affect DSGI graphics output the same way as they affect other SAS/GRAPH procedure output. When `TITLE` and `FOOTNOTE` statements are used, the output from DSGI statements is placed in the procedure output area. See “Placement of Graphic Elements in the Graphics Output Area” on page 39 for an explanation of how space in graphics output is allocated to titles and footnotes.

Some DSGI functions override the graphics options. The following table lists the DSGI functions that directly override graphics options. For details about the graphics options, see Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261.

DSGI Function	Graphics Option That Is Overridden
<code>GSET('CBACK', . . .)</code>	<code>CBACK=</code>
<code>GSET('COLREP', . . .)</code>	<code>COLORS=</code>
<code>GSET('DEVICE', . . .)</code>	<code>DEVICE=</code>
<code>GSET('HPOS', . . .)</code>	<code>HPOS=</code>
<code>GSET('HSIZE', . . .)</code>	<code>HSIZE=</code>
<code>GSET('VPOS', . . .)</code>	<code>VPOS=</code>
<code>GSET('VSIZE', . . .)</code>	<code>VSIZE=</code>

DSGI Function	Graphics Option That Is Overridden
GSET('TEXCOLOR', . . .)	CTEXT=
GSET('TEXTFONT', . . .)	FTEXT=
GSET('TEXHEIGHT', . . .)	HTEXT=

Operating States

The operating state of DSGI determines which functions and routines may be issued at any point in the DATA step. You can only submit a function or routine when the operating state is appropriate for it. See “How Operating States Control the Order of DSGI Statements” on page 1370 for a discussion of how functions and routines should be ordered within the operating states.

The operating states defined by DSGI are

GKCL	facility closed, the initial state of DSGI. No graphical resources have been allocated.
GKOP	facility open. When DSGI is open, you may check the settings of the attributes.
SGOP	segment open. At this point, graphics output primitives may be generated.
WSAC	workstation active. When the workstation is active, it can receive DSGI statements.
WSOP	workstation open. In this implementation, the graphics catalog, either the default or the one specified through the GSET('CATALOG', . . .)command, is opened or created.

Refer to individual functions and routines in Chapter 50, “DATA Step Graphics Interface Dictionary,” on page 1401 for the operating states from which that function or routine can be issued.

The Current Window System

When DSGI draws graphics, it evaluates x and y coordinates in terms of the *current window system*, either a window you have defined or the default window system. Unless you define and activate a different window, DSGI uses the default window system.

The default window system assigns two arbitrary systems of units to the x and y axes. The default window guarantees a range of 0 through 100 in one direction (usually the y direction) and at least 0 through 100 in the other (usually the x direction). The ranges depend on the dimensions of your device. You can use the GASK('WINDOW', . . .)routine to determine the dimensions of your default window system.

You can define the x and y ranges to be any numeric range. For example, you can use -1000 to $+2000$ on the x axis and 30 to 35 on the y axis. The units used are arbitrary.

Debugging DSGI Programs

When DSGI encounters an error in a program, it flags the statement in the SAS log and displays a description of the error. (To receive SAS System messages, GSET('MESSAGE', . . .) must be ON.) The description provides you with an explanation of the error. The description may also provide a return code. If you get a return code, you can refer to “Return Codes for DSGI Routines and Functions” on page 1501 for a description of the error and why it might have occurred.

Some of the most common errors in DSGI programs are

- syntax errors
- an invalid number of arguments for the function or routine
- a function or routine being executed in an operating state that is not correct for the function or routine.

DSGI Graphics Summary

The following sections summarize the functions and routines you can use to create graphics output with DSGI.

DSGI Functions

DSGI provides functions that

- initialize and terminate DSGI
- generate graphics elements
- control the appearance of graphics elements by setting attributes
- control the overall appearance of the graphics output
- perform management operations for the catalog
- control messages issued by DSGI.

Table 49.1 on page 1360 summarizes the types of operations available and the functions used to invoke them. Refer to Chapter 50, “DATA Step Graphics Interface Dictionary,” on page 1401 for details about each function.

Table 49.1 DATA Step Graphics Interface Functions

DSGI Operations	Associated Function	Function Description
Bundling Attributes (valid values for xxx are FIL, LIN, MAR, and TEX)		
	GSET('ASF', . . .)	sets the aspect source flag of an attribute
	GSET('xxxINDEX', . . .)	selects the bundle of attributes to use
	GSET('xxxREP', . . .)	assigns attributes to a bundle

Setting Attributes That Affect Graphics Elements

DSGI Operations	Associated Function	Function Description
color index	GSET('COLREF', . . .)	assigns a color name to color index
fill area	GSET('FILCOLOR', . . .)	selects the color of the fill area
	GSET('FILSTYLE', . . .)	selects the pattern when FILTYPE is HATCH or PATTERN
	GSET('FILTYPE', . . .)	specifies the type of interior for the fill area
	GSET('HTML', . . .)	specifies the HTML string to invoke when an affected DSGI graphic element in a web page is clicked
line	GSET('LINCOLOR', . . .)	selects the color of the line
	GSET('LINTYPE', . . .)	sets the type of line
	GSET('LINWIDTH', . . .)	specifies the width of the line
marker	GSET('MARCOLOR', . . .)	selects the color of the marker
	GSET('MARSIZE', . . .)	determines the size of the marker
	GSET('MARTYPE', . . .)	sets the type of marker drawn
text	GSET('TEXALIGN', . . .)	specifies horizontal and vertical alignment of text
	GSET('TEXCOLOR', . . .)	selects the color of the text
	GSET('TEXFONT', . . .)	sets the font for the text
	GSET('TEXHEIGHT', . . .)	selects the height of the text
	GSET('TEXPATH', . . .)	determines reading direction of text
	GSET('TEXUP', . . .)	selects the angle of text
Setting Attributes That Affect Entire Graph		
	GSET('ASPECT', . . .)	sets the aspect ratio

DSGI Operations	Associated Function	Function Description
	GSET('CATALOG', . . .)	selects the catalog to use
	GSET('CBACK', . . .)	selects the background color
	GSET('DEVICE', . . .)	specifies the output device
	GSET('HPOS', . . .)	sets the number of columns in the graphics output area
	GSET('HSIZE', . . .)	sets the width of the graphics output area in units of inches
	GSET('VPOS', . . .)	sets the number of rows in the graphics output area
	GSET('VSIZE', . . .)	sets the height of the graphics output area in units of inches
Managing Catalogs		
	GRAPH('COPY', . . .)	copies a graph to another entry within the same catalog
	GRAPH('DELETE', . . .)	deletes a graph
	GRAPH('INSERT', . . .)	inserts a previously created graph into the currently open segment
	GRAPH('RENAME', . . .)	renames a graph
Drawing Graphics Elements		
arc	GDRAW('ARC', . . .)	draws a circular arc
bar	GDRAW('BAR', . . .)	draws a rectangle that can be filled
ellipse	GDRAW('ELLIPSE', . . .)	draws an oblong circle that can be filled
elliptical arc	GDRAW('ELLARC', . . .)	draws an elliptical arc
fill area	GDRAW('FILL', . . .)	draws a polygon that can be filled

DSGI Operations	Associated Function	Function Description
line	GDRAW('LINE', . . .)	draws a single line, a series of connected lines, or a dot
marker	GDRAW('MARK', . . .)	draws one or more symbols
pie	GDRAW('PIE', . . .)	draws a pie slice that can be filled
text	GDRAW('TEXT', . . .)	draws a character string
Initializing DSGI		
	GINIT()	initializes DSGI
	GRAPH('CLEAR', . . .)	opens a segment to receive graphics primitives
Handling Messages		
	GDRAW('MESSAGE', . . .)	prints a message in the SAS log
	GPRINT(<i>code</i>)	prints the description of a DSGI error code
	GSET('MESSAGE', . . .)	turns message logging on or off
Ending DSGI		
	GRAPH('UPDATE', . . .)	closes the currently open segment and, optionally, displays it
	GTERM()	ends DSGI
Activating Transformations		
	GET('TRANSNO', . . .)	selects the transformation number of the viewport or window to use
Defining Viewports		
	GSET('CLIP', . . .)	turns clipping on or off
	GSET('VIEWPORT', . . .)	sets the coordinates of the viewport and assigns it a transformation number

DSGI Operations	Associated Function	Function Description
Defining Windows		
	GSET('WINDOW', . . .)	sets the coordinates of the window and assigns it a transformation number

DSGI Routines

DSGI routines return the values set by some of the DSGI functions. Table 49.2 on page 1364 summarizes the types of values that the GASK routines can check. Refer to Chapter 50, “DATA Step Graphics Interface Dictionary,” on page 1401 for details about each routine.

Table 49.2 DATA Step Graphics Interface Routines

DSGI Operations	Associated Routine	Routine Description
Checking Attribute Bundles (valid values for xxx are FIL, LIN, MAR, and TEX)		
	GASK('ASK', . . .)	returns the aspect source flag of the attribute
	GASK('xxxINDEX', . . .)	returns the index of the active bundle
	GASK('xxxREP', . . .)	returns the attributes assigned to the bundle
Checking Attribute Settings		
color index	GASK('COLINDEX', . . .)	returns the color indices that currently have colors assigned to them
	GASK('COLREP', . . .)	returns the color name assigned to the color index
fill area	GASK('FILCOLOR', . . .)	returns the color of the fill area
	GASK('FILSTYLE', . . .)	returns the index of the pattern when the FILTYPE is HATCH or PATTERN
	GASK('FILTYPE', . . .)	returns the index of the type of interior

DSGI Operations	Associated Routine	Routine Description
	GASK('HTML', . . .)	finds the HTML string that is in effect when one of the following graphic elements is drawn: bar, ellipse, fill, mark, pie, and text.
line	GASK('LINCOLOR', . . .)	returns the color index of the color of the line
	GASK('LINTYPE', . . .)	returns the index of the type of line
	GASK('LINWIDTH', . . .)	returns the width of the line
marker	GASK('MARCOLOR', . . .)	returns the color index of the color of markers
	GASK('MARSIZE', . . .)	returns the size of markers
	GASK('MARTYPE', . . .)	returns the index of the type of marker drawn
text	GASK('TEXALIGN', . . .)	returns the horizontal and vertical alignment of text
	GASK('TEXCOLOR', . . .)	returns the color index of the color of text
	GASK('TEXEXTENT', . . .)	returns the coordinates of text extent rectangle and the text concatenation point of the character string
	GASK('TEXFONT', . . .)	returns the text font
	GASK('TEXHEIGHT', . . .)	returns the height of text
	GASK('TEXPATH', . . .)	returns the reading direction of text
	GASK('TEXUP', . . .)	returns the character up vector in x vector and y vector
Checking Attributes That Affect Entire Graph		
	GASK('ASPECT', . . .)	returns the aspect ratio
	GASK('CATALOG', . . .)	returns the current catalog
	GASK('CBACK', . . .)	returns the background color
	GASK('DEVICE', . . .)	returns the current output device

DSGI Operations	Associated Routine	Routine Description
	GASK('HPOS', . . .)	returns the number of columns in the graphics output area
	GASK('HSIZE', . . .)	returns the width of the graphics output area in units of inches
	GASK('MAXDISP', . . .)	returns the dimensions of maximum display area for the device in meters and pixels
	GASK('VPOS', . . .)	returns the number of rows in the graphics output area
	GASK('VSIZE', . . .)	returns the height of the graphics output area in units of inches
Querying Catalogs		
	GASK('GRAPHLIST', . . .)	returns the names of graphs in the current catalog
	GASK('NUMGRAPH', . . .)	returns the number of graphs in the current catalog
	GASK('OPENGRAPH', . . .)	returns the name of the currently open graph
Checking System Status		
	GASK('STATE', . . .)	returns the current operating state
	GASK('WSACTIVE', . . .)	returns whether or not the workstation is active
	GASK('WSOPEN', . . .)	returns whether or not the workstation is open
Checking Transformation Definitions		
	GASK('TRANS', . . .)	returns the coordinates of the viewport and window associated with the transformation
	GASK('TRANSNO', . . .)	returns the active transformation number
Checking Viewport Definitions		
	GASK('CLIP', . . .)	returns the status of clipping

DSGI		
Operations	Associated Routine	Routine Description
	GASK('VIEWPORT', . . .)	returns the coordinates of the viewport assigned to the transformation number
Checking Window Definitions		
	GASK('WINDOW', . . .)	returns the coordinates of the window assigned to the transformation number

Creating Simple Graphics with DSGI

Within any DSGI program, you need to follow these basic steps:

1 Initialize DSGI.

The function that initializes DSGI is GINIT(). GINIT() loads the graphics sublibrary, opens a workstation, and activates a workstation.

2 Open a graphics segment.

Before you can submit graphics primitives, you must submit the GRAPH('CLEAR', . . .) function. GRAPH('CLEAR', . . .) opens a graphic segment so that graphics primitives can be submitted.

3 Generate graphics elements.

DSGI can generate arcs, bars, ellipses, elliptical arcs, lines, markers, pie slices, polygons (fill areas), and text. These graphics elements are all produced with the GDRAW function using their associated operator names.

GDRAW functions can only be submitted when a graphics segment is open. Therefore, they must be submitted between the GRAPH('CLEAR', . . .) and GRAPH('UPDATE', . . .) functions.

4 Close the graphics segment.

Once the attribute and graphics statements have been entered, you must submit statements to close the graphics segment and output the graph. The GRAPH('UPDATE', . . .) function closes the graphic segment currently open and, optionally, displays the graphics output.

5 End DSGI.

The GTERM() function ends DSGI by deactivating and closing the workstation, and closing the graphics sublibrary. It frees any memory allocated by DSGI.

Note: You must execute a RUN statement at the end of the DATA step to display the output.

Figure 49.1 on page 1368 outlines the basic steps and shows the functions used to initiate steps 1, 2, 4, and 5. Step 3 can consist of many types of functions. The GDRAW('LINE', . . .)function is used as an example.

Figure 49.1 Basic Steps Used in Creating DSGI Graphics Output

```

data dsname;
.
.
.

/* Step 1 -initialize DSGI */
rc=ginit();
.
.
.

/* Step 2 -open graphics segment */
rc=graph('clear');
.
.
.

/* Step 3 -generate graphics elements */
rc=gdraw('line' ,2, 30, 50, 70, 50);
.
.
.

/* Step 4 -close graphics segment and display output */
rc=graph('update');
.
.
.

/* Step 1 -end DSGI */
rc=gterm();
.
.
.
run;

```

Notice that there are two pairs of functions that work together within a DSGI DATA step (shown by a and b in Figure 49.1 on page 1368). The first pair, GINIT() and GTERM(), begin and end DSGI. Within the first pair, the second pair, GRAPH('CLEAR', . . .)and GRAPH('UPDATE', . . .)begin and end a graphics segment. You can repeat these pairs within a single DATA step to produce multiple graphics output; however, the relative positions of these functions must be maintained within a DATA step. See “Generating Multiple Graphics Output in One DATA Step” on page 1380 for more information about producing multiple graphics outputs from one DATA step.

The order of these steps is controlled by DSGI operating states. Before any DSGI function or routine can be submitted, the operating state in which that function or routine can be submitted must be active. See “How Operating States Control the Order of DSGI Statements” on page 1370.

Setting Attributes for Graphics Elements

The appearance of the graphics elements is determined by the settings of the attributes. Attributes control such aspects as height of text; text font; and color, size, and width of the graphics element. In addition, the HTML attribute determines whether the element provides a link to another graphic or web page. Attributes are set and reset with GSET functions. GASK routines return the current setting of the attribute specified.

Each graphics primitive is associated with a particular set of attributes. Its appearance or linking capability can only be altered by that set of attributes. Table 49.3 on page 1369 lists the operators used with GDRAW functions to generate graphics elements and the attributes that control them.

Table 49.3 Graphics Output Primitive Functions and Associated Attributes

Graphics Output Primitive	Functions	Associated Attributes
Arc	GDRAW('ARC', . . .)	HTML, LINCOLOR, LININDEX, LINREP, LINTYPE, LINWIDTH
Bar	GDRAW('BAR', . . .)	FILCOLOR, FILINDEX, FILREP, FILSTYLE, FILTYPE, HTML
Ellipse	GDRAW('ELLIPSE', . . .)	FILCOLOR, FILINDEX, FILREP, FILSTYLE, FILTYPE, HTML
Elliptical Arc	GDRAW('ELLARC', . . .)	HTML, LINCOLOR, LININDEX, LINREP, LINTYPE, LINWIDTH
Fill Area	GDRAW('FILL', . . .)	FILCOLOR, FILINDEX, FILREP, FILSTYLE, FILTYPE, HTML
Line	GDRAW('LINE', . . .)	HTML, LINCOLOR, LININDEX, LINREP, LINTYPE, LINWIDTH
Marker	GDRAW('MARK', . . .)	HTML, MARCOLOR, MARINDEX, MARREP, MARSIZE, MARTYPE
Pie	GDRAW('PIE', . . .)	FILCOLOR, FILINDEX, FILREP, FILSTYLE, FILTYPE, HTML
Text	GDRAW('TEXT', . . .)	HTML, TEXALIGN, TEXCOLOR, TEXFONT, TEXHEIGHT, TEXINDEX, TEXPATH, TEXREP, TEXUP

Attribute functions must precede the graphics primitive they control. Once an attribute is set, it controls any associated graphics primitives that follow. If you want to change the setting, you can issue another `GSET(attribute, . . .)` function with the new setting.

If you do not set an attribute before you submit a graphics primitive, DSGI uses the default value for the attribute. Refer to Chapter 50, “DATA Step Graphics Interface Dictionary,” on page 1401 for the default values used for each attribute.

How Operating States Control the Order of DSGI Statements

Each DSGI function and routine can only be submitted when certain operating states are active. This restriction affects the order of functions and routines within the DATA step. Generally, the operating states within a DATA step follow this order:

GKCL \rightarrow WSAC \rightarrow SGOP \rightarrow WSAC \rightarrow GKCL

Functions That Change the Operating State

The functions described earlier in steps 1, 2, 4, and 5 actually control the changes to the operating state. For example, the GINIT() function must be submitted when the operating state is GKCL, the initial state of DSGI. GINIT() then changes the operating state to WSAC. The GRAPH('CLEAR', . . .) function must be submitted when the operating state is WSAC and before any graphics primitives are submitted. The reason it precedes graphics primitives is that it changes the operating state to SGOP, the operating state in which you can submit graphics primitives. The following list shows the change in the operating state due to specific functions:

GINIT()	GKCL \rightarrow WSAC
GRAPH('CLEAR', . . .)	WSAC \rightarrow SGOP
GRAPH('UPDATE', . . .)	SGOP \rightarrow WSAC
GTERM()	WSAC \rightarrow GKCL

Because these functions change the operating state, you must order all other functions and routines so that the change in operating state is appropriate for the functions and routines that follow. The following program statements show how the operating state changes from step to step in a typical DSGI program. They also summarize the functions and routines that can be submitted under each operating state. The functions that change the operating state are included as actual statements. Refer to “Operating States” on page 1402 for the operating states from which functions and routines can be submitted.

```
data dsname;

    /* GKCL - initial state of DSGI; can execute:          */
    /* 1. GSET functions that set attributes              */
    /*    that affect the entire graphics output          */
    /* 2. some catalog management functions               */
    /*    (some GRAPH functions)                          */

    /* Step 1 - initialize DSGI                            */
rc=ginit();

    /* WSAC - workstation is active; can execute:         */
    /* 1. most GASK routines                               */
    /* 2. some catalog management functions               */
    /*    (some GRAPH functions)                          */
```



```

/* 3. GSET functions that set attributes */
/*    and bundles, viewports, windows, */
/*    transformations, and message logging */

/* Step 2 - open a graphics segment */
rc=graph('clear', 'text');

/* SGOP - segment open; can execute: */
/* 1. any GASK routine */
/* 2. any GDRAW function */
/* 3. some catalog management functions */
/*    (some GRAPH functions) */
/* 4. GSET functions that set attributes */
/*    and bundles, viewports, windows, */
/*    transformations, and message logging */

/* Step 3 - execute graphics primitives */
rc = gdraw('line', 2, 30,50,50,50);

/* Step 4 - close the graphics segment */
rc=graph('update');

/* WSAC - workstation is active; can execute: */
/* 1. most GASK routines */
/* 2. some catalog management functions */
/*    (some GRAPH functions) */
/* 3. GSET functions that set attributes */
/*    and bundles, viewports, windows, */
/*    transformations, and message logging */

/* Step 5 - end DSGI */
rc=gterm();

/* GKCL - initial state of DSGI */
run;

```

Order of Functions and Routines

Functions and routines within each operating state can technically be submitted in any order; however, once an attribute is set, it remains in effect until the end of the DATA step or until you change its value. If you are producing multiple graphics output within the same DATA step, the attributes for one output affect the ones that follow. Attributes are not reset until after the GTERM() function is submitted.

Notice that you can set attributes for the graphics primitives in several places. As long as the functions that set the attributes are executed before the graphics primitives, they will affect the graphics output. If you execute them after a graphics primitive, the primitive is not affected. See “Setting Attributes for Graphics Elements” on page 1368.

The following program statements illustrate a more complex DSGI program that produces Display 49.3 on page 1373 when submitted. Notice that all attributes for a graphics primitive are executed before the graphics primitive. In addition, the GINIT() and GTERM() pairing and the GRAPH('CLEAR') and GRAPH('UPDATE') pairing are maintained within the DATA step. Refer to “Operating States” on page 1402 for the operating states in which each function and routine can be submitted.

```
/* set the graphics environment */
goptions reset=global gunit=pct border
        hsize=7 in vsize=5 in
        targetdevice=pscolor;

/* execute a DATA step with DSGI */
data dsname;
    /* initialize SAS/GRAPH software */
    /* to accept DSGI statements */
    rc=ginit();
    rc=graph('clear');

    /* assign colors to color index */
    rc=gset('colrep', 1, 'blue');
    rc=gset('colrep', 2, 'red');

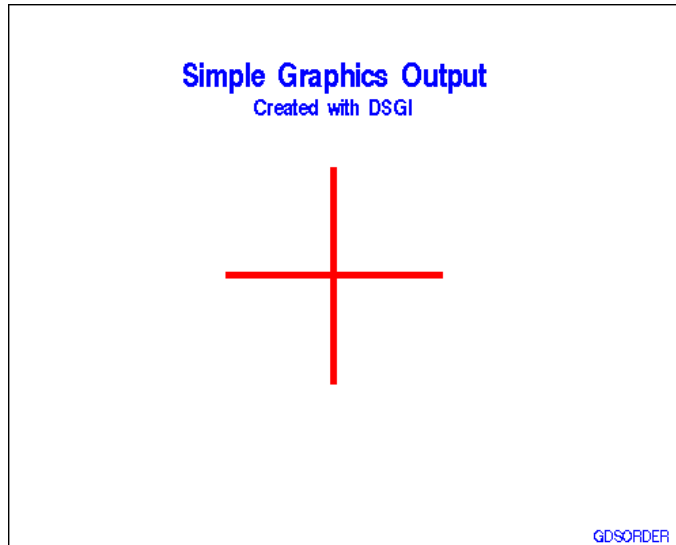
    /* define and display titles */
    rc=gset('texcolor', 1);
    rc=gset('texfont', 'swissb');
    rc=gset('texheight', 6);
    rc=gdraw('text', 45, 93, 'Simple Graphics Output');

    /* change the height and */
    /* display second title */
    rc=gset('texheight', 4);
    rc=gdraw('text', 58, 85, 'Created with DSGI');

    /* define and display footnotes */
    /* using same text font and */
    /* color as defined for titles */
    rc=gset('texheight', 3);
    rc=gdraw('text', 125, 1, 'GDSORDER ');

    /* define and draw bar */
    rc=gset('lincolor', 2);
    rc=gset('linwidth', 5);
    rc=gdraw('line', 2, 72, 72, 30, 70);
    rc=gdraw('line', 2, 52, 92, 50, 50);

    /* display graph and end DSGI */
    rc=graph('update');
    rc=gterm();
run;
```

Display 49.3 Simple Graphics Output Generated with DSGI

Bundling Attributes

DSGI allows you to bundle attributes. As a result, you can select a group of attribute values rather than having to select each one individually. This feature is useful if you use the same attribute settings over and over within the same DATA step.

To use an attribute bundle, you assign the values of the attributes to a bundle index. When you want to use those attributes for a graphics primitive, you select the bundle rather than set each attribute separately.

Attributes That Can Be Bundled for Each Graphics Primitive

Each graphics primitive has a group of attributes associated with it that can be bundled. Only the attributes in that group can be assigned to the bundle. Table 49.4 on page 1373 shows the attributes that can be bundled for each graphics primitive.

Note: You do not have to use attribute bundles for all graphics primitives if you use a bundle for one. You can define bundles for some graphics primitives and set the attributes individually for others. △

However, if the other graphics primitives are associated with the same attributes you have bundled and you do not want to use the same values, you can use other bundles to set the attributes, or you can set the attributes back to 'INDIVIDUAL'.

Table 49.4 Attributes That Can Be Bundled for Each Graphics Primitive

Graphics Output Primitive	Associated Attributes That Can Be Bundled
GDRAW('ARC', . . .)	LINCOLOR, LINTYPE, LINWIDTH
GDRAW('BAR', . . .)	FILCOLOR, FILSTYLE, FILTYPE
GDRAW('ELLARC', . . .)	LINCOLOR, LINTYPE, LINWIDTH

Graphics Output Primitive	Associated Attributes That Can Be Bundled
GDRAW('ELLIPSE', . . .)	FILCOLOR, FILSTYLE, FILTYPE
GDRAW('FILL', . . .)	FILCOLOR, FILSTYLE, FILTYPE
GDRAW('LINE', . . .)	LINCOLOR, LINTYPE, LINWIDTH
GDRAW('MARK', . . .)	MARCOLOR, MARSIZE, MARTYPE
GDRAW('PIE', . . .)	FILCOLOR, FILSTYLE, FILTYPE
GDRAW('TEXT', . . .)	TEXCOLOR, TEXTFONT

Assigning Attributes to a Bundle

To assign values of attributes to a bundle, you must

- assign the values to a numeric bundle index with the GSET('xxx REP', . . .)function. Each set of attributes that can be bundled uses a separate GSET('xxx REP', . . .)function, where *xxx* is the appropriate prefix for the set of attributes to be bundled. Valid values for *xxx* are FIL, LIN, MAR, and TEX.
- set the aspect source flag (ASF) of the attributes to 'BUNDLED' before you use the bundled attributes. You can use the GSET('ASF', . . .)function to set the ASF of an attribute. You need to execute a GSET('ASF', . . .)function for each attribute in the bundle.

The following example assigns the text attributes, color, and font, to the bundle indexed by the number 1. As shown in the GSET('TEXREP', . . .)function, the color for the bundle is green, the second color in the COLOR= graphics option. The font for the bundle is the 'ZAPF' font. (See "COLREP" on page 1467 for an explanation of how colors are used in DSGI.)

```
goptions colors=(red green blue);

data dsname;
.
.   /* other DATA step statements */
.
.   /* associate the bundle with the index 1 */
rc=gset('texrep', 1, 2, 'zapf');
.
.   /* more statements */
.
.   /* assign the text attributes to a bundle */
rc=gset('asf', 'texcolor', 'bundled');
rc=gset('asf', 'texfont', 'bundled');

.
.   /* draw the text */
rc=gdraw('text', 50, 50, 'Today is the day.');
```

The bundled attributes are used when an associated GDRAW function is executed. If the ASF of an attribute is not set to 'BUNDLED' at the time a GDRAW function is executed, DSGI searches for a value to use in the following order:

- 1 the current value of the attribute
- 2 the default value of the attribute.

Selecting a Bundle

Once you have issued the GSET('ASF', . . .) and GSET('xxx REP', . . .) functions, you can issue the GSET('xxx INDEX', . . .) function to select the bundle. The following statement selects the bundle defined in the previous example:

```
/* invoke the bundle of text attributes */
rc=gset('texindex', 1);
```

The 1 in this example corresponds to the index number specified in the GSET('TEXREP', . . .) function.

Defining Multiple Bundles for a Graphics Primitive

You can set up more than one bundle for graphics primitives by issuing another GSET('xxx REP', . . .) function with a different index number. If you wanted to add a second attribute bundle for text to the previous example, you could issue the following statement:

```
/* define another attribute bundle for text */
rc=gset('texrep', 2, 3, 'swiss');
```

When you activate the second bundle, the graphics primitives for the text that follows will use the third color, blue, and the SWISS font.

Note: When using a new bundle, you do not need to reissue the GSET('ASF', . . .) functions for the attributes that will be bundled. Once the ASF of an attribute has been set, the setting remains in effect until it is changed. \triangle

How DSGI Selects the Value of an Attribute to Use

Attributes that are bundled override any of the same attributes that are individually set. For example, you assign the line color green, the type 1, and the width 5 to a line bundle with the following statements:

```
goptions colors=(red green blue);
rc=gset('asf', 'lincolor', 'bundled');
rc=gset('asf', 'linwidth', 'bundled');
rc=gset('asf', 'lintype', 'bundled');
rc=gset('linrep', 3, 2, 5, 1);
```

In subsequent statements, you activate the bundle, select other attributes for the line, and then draw a line:

```
/* activate the bundle */
rc=gset('linindex', 3);

/* select other attributes for the line */
rc=gset('lincolor', 3);
rc=gset('linwidth', 10);
rc=gset('lintype', 4);

/* draw a line from point (30,50) to (70,50) */
rc=gdraw('line', 2, 30, 70, 50, 50);
```

The color, type, and width associated with the line bundle are used rather than the attributes set just before the GDRAW('LINE', . . .) function was executed. The line that

is drawn is green (the second color from the colors list of the COLORS= graphics option), 5 units wide, and solid (line type 1).

During processing, DSGI chooses the value of an attribute using the following logic:

- 1 Get the index of the active line bundle.
- 2 Check the ASF of the LINCOLOR attribute. If the ASF is 'INDIVIDUAL', the value selected with GSET('LINCOLOR', . . .) is used; otherwise, the LINCOLOR associated with the bundle index is used.
- 3 Check the ASF of the LINTYPE attribute. If the ASF is 'INDIVIDUAL', the value selected with GSET('LINTYPE', . . .) is used; otherwise, the LINTYPE associated with the bundle index is used.
- 4 Check the ASF of the LINWIDTH attribute. If the ASF is 'INDIVIDUAL', the value selected with GSET('LINWIDTH', . . .) is used; otherwise, the LINWIDTH associated with the bundle index is used.
- 5 Draw the line using the appropriate color, type, and width for the line.

Disassociating an Attribute from a Bundle

To disassociate an attribute from a bundle, use the GSET('ASF', . . .) function to reset the ASF of the attribute to 'INDIVIDUAL'. The following program statements demonstrate how to disassociate the attributes from the text bundle:

```
/* disassociate an attribute from a bundle */
rc=gset('asf', 'texcolor', 'individual');
rc=gset('asf', 'texfont', 'individual');
```

Using Viewports and Windows

In DSGI, you can define viewports and windows. Viewports enable you to subdivide the graphics output area and insert existing graphs or draw graphics elements in smaller sections of the graphics output area. Windows define the coordinate system within a viewport and enable you to scale the graph or graphics elements drawn within the viewport.

The default viewport is defined as (0,0) to (1,1) with 1 being 100 percent of the graphics output area. If you do not define a viewport, graphics elements or graphs are drawn using the default.

The default window is defined so that a rectangle drawn from window coordinates (0,0) to (100,100) is square and fills the display in one dimension. The actual dimensions of the default window are device dependent. Use the GASK('WINDOW', . . .) routine to find the exact dimensions of your default window. You can define a window without defining a viewport. The coordinate system of the window is used with the default viewport.

If you define a viewport, you can position it anywhere in the graphics output area. You can define multiple viewports within the graphics output area so that more than one existing graph, part of a graph, or more than one graphics element can be inserted into the graphics output.

Transformations activate both a viewport and the associated window. DSGI maintains 21 (0 through 20) transformations. By default, transformation 0 is active. Transformation 0 always uses the entire graphics output area for the viewport and maps the window coordinates to fill the viewport. The definition of the viewport and window of transformation 0 may not be changed.

By default, the viewports and windows of all the other transformations (1 through 20) are set to the defaults for viewports and windows. If you want to define a different viewport or window, you must select a transformation number between 1 and 20.

You generally follow these steps when defining viewports or windows:

- Define the viewport or window.
- Activate the transformation so that the viewport or window is used for the output.

These steps can be submitted in any order; however, if you use a transformation you have not defined, the default viewport and window are used. Once you activate a transformation, the graphics elements drawn by the subsequent DSGI functions are drawn in the viewport and window associated with that transformation.

Defining Viewports

You can define a viewport with the `GSET('VIEWPORT', n , . . .)` function, where n is the transformation number of the viewport you are defining. You can also use this function to define multiple viewports, each containing a portion of the graphics output area. You can then place a separate graph, part of a graph, or graphics elements within each viewport.

The following program statements divide the graphics output area into four subareas:

```
/* define the first viewport, indexed by 1 */
rc=gset('viewport', 1, .05, .05, .45, .45);

      /* define the second viewport, indexed by 2 */
rc=gset('viewport', 2, .55, .05, .95, .45);

      /* define the third viewport, indexed by 3 */
rc=gset('viewport', 3, .55, .55, .95, .95);

      /* define the fourth viewport, indexed by 4 */
rc=gset('viewport', 4, .05, .55, .45, .95);
```

Once you define the viewports, you can insert existing graphs or draw graphics elements in each viewport by activating the transformation of that viewport.

Clipping around Viewports

When you use viewports, you also may need to use the clipping feature. Even though you have defined the dimensions of your viewport, it is possible for graphics elements to display past its boundaries. If the graphics elements are too large to fit into the dimensions you have defined, portions of the graphics elements actually display outside of the viewport. To ensure that only the portions of the graphics elements that fit within the dimensions of the viewport display, turn the clipping feature on by using the `GSET('CLIP', . . .)` function. For details, see "CLIP" on page 1467.

Defining Windows

You can define a window by using the `GSET('WINDOW', n , . . .)` function, where n is the transformation number of the window you are defining. If you are defining a window for a viewport you have also defined, n must match the transformation number of the viewport.

You can scale the x and y axes differently for a window. The following program statements scale the axes for each of the four viewports defined earlier in "Defining Viewports":

```
/* define the window for viewport 1 */
rc=gset('window', 1, 0, 50, 20, 100);
```

```

    /* define the window for viewport 2 */
rc=gset('window', 2, 0, 40, 20, 90);

    /* define the window for viewport 3 */
rc=gset('window', 3, 10, 25, 45, 100);

    /* define the window for viewport 4 */
rc=gset('window', 4, 0, 0, 100, 100);

```

See “Scaling Graphs by Using Windows” on page 1388 for an example of using windows to scale graphs.

Note: When you define a window for a viewport, the transformation numbers in the GSET(‘VIEWPORT’, . . .)and GSET(‘WINDOW’, . . .)functions must match in order for DSGI to activate them simultaneously. \triangle

Activating Transformations

Once you have defined a viewport or window, you must activate the transformation in order for DSGI to use the viewport or window. To activate the transformation, use the GSET(‘TRANSNO’, n , . . .)function where n has the same value as n in GSET(‘VIEWPORT’, n , . . .)or GSET(‘WINDOW’, n , . . .).

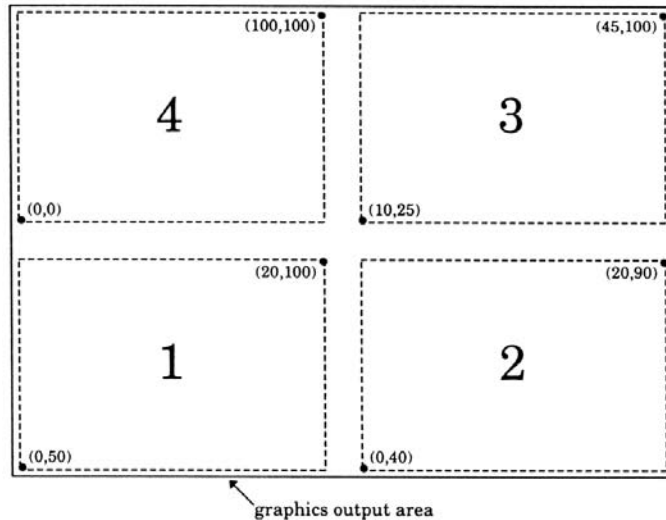
The following program statements illustrate how to activate the viewports and windows defined in the previous examples:

```

/* define the viewports */
.
.
.
/* define the windows */
.
.
.
/* activate the first transformation */
gset('transno', 1);
.
. /* graphics primitive functions follow */
.
/* activate the second transformation */
gset('transno', 2);
.
. /* graphics primitive functions follow */
.
/* activate the third transformation */
gset('transno', 3);
.
. /* graphics primitive functions follow */
.
/* activate the fourth transformation */
gset('transno', 4);
.
. /* graphics primitive functions follow */
.

```

When you activate these transformations, your display is logically divided into four subareas as shown in Figure 49.2 on page 1379.

Figure 49.2 Graphics Output Area Divided into Four Logical Transformations

If you want to use the default viewport and window after selecting different ones, execute the `GSET('TRANSNO', 0)` function to reselect the default transformation for DSGI.

Inserting Existing Graphs into DSGI Graphics Output

You can insert existing graphs into graphics output you are creating. The graph you insert must be in the same catalog in which you are currently working. Follow these steps to insert an existing graph:

- 1 Use the `GSET('CATALOG', . . .)` function to set the output catalog to the catalog that contains the existing graph.

Note: Unless you are using the WORK library, you must have previously defined the libref in a LIBNAME statement or window when using `GSET('CATALOG', . . .)`. Δ

- 2 Define a viewport with the dimensions and position of the place in the graphics output where you want to insert the existing graph. `GSET('VIEWPORT', n , . . .)` defines a viewport and `GSET('WINDOW', n , . . .)` defines a window.
- 3 Define a window as (0,0) to (100,100) so that the inserted graph is not distorted. The graph must have a square area defined to avoid the distortion. If your device does not have a square graphics output area, the window defaults to the units of the device rather than (0,0) to (100,100) and may distort the graph.
- 4 Activate the transformation number n , as defined in the viewport function, and possibly in the window function, using `GSET('TRANSNO', n , . . .)`.
- 5 Use the `GRAPH('INSERT', . . .)` function with the name of the existing graph.

The following program statements provide an example of including an existing graph in the graphics output being created. The name of the existing graph is 'MAP'. 'LOCAL' points to the library containing the catalog 'MAPCTLG'. The coordinates of the viewport are percentages of the graphics output area. **SAS-data-library** refers to a permanent SAS data library.

Example Code 49.1 Graphics Output Area Divided into Four Logical Transformations

```
libname local 'SAS-data-library';
```

```

.
.
.
      /* select the output catalog to the */
      /* catalog that contains 'map' */
rc=gset('catalog', 'local', 'mapctlg');
.
.
.

      /* define the viewport to contain the */
      /* existing graph */
rc=gset('viewport', 1, .25, .45, .75, .9);
rc=gset('window', 1, 0, 0, 100, 100);

      /* set the transformation number to the one */
      /* defined in the viewport function */
rc=gset('transno', 1);

      /* insert the existing graph */
rc=graph('insert', 'map');

```

These statements put the existing graph 'MAP' in the upper half of the graphics output.

Generating Multiple Graphics Output in One DATA Step

You can produce more than one graphics output within the same DATA step. All statements between the GRAPH('CLEAR', . . .)and GRAPH('UPDATE', . . .)functions will produce one graphics output.

Each time the GRAPH('UPDATE', . . .)function is executed, a graph is displayed. After the GTERM() function is executed, no more graphs are displayed for the DATA step. The GINIT() function must be executed again to produce more graphs.

CAUTION:

Be careful using global SAS/GRAPH statements when you are producing multiple output from within the DATA step. Δ

If you use global SAS/GRAPH statements when producing multiple output from one DATA step, the last definition of the statements is used for all displays.

Processing DSGI Statements in Loops

You can process DSGI statements in loops to draw a graphics element multiple times in one graphics output or to produce multiple output. If you use loops, you must maintain the GRAPH('CLEAR', . . .)and GRAPH('UPDATE', . . .)pairing within the GINIT() and GTERM() pairing. (See Figure 49.1 on page 1368.) The following program statements illustrate how you can use DSGI statements to produce multiple graphics output for different output devices:

```

data _null_;
  length d1-d5 $ 8;
  input d1-d5;
  array devices{5} d1-d5;

```

```

.
.
.
do j=1 to 5;
  rc=gset('device', devices{j});
  .
  .
  rc=ginit();
  .
  .
  do i=1 to 5;
    rc=graph('clear');
    rc=gset('filcolor', i);
    rc=gdraw('bar', 45, 45, 65, 65);
    rc=graph('update');
  end;
  .
  .
  rc=gterm();
end;
cards;
tek4105 hp7475 ps qms800 ibm3279
;
run;

```

The inner loop produces five graphs for each device. Each graphics output produced by the inner loop consists of a bar. The bar uses a different color for each graph. The outer loop produces all of the graphs for five different devices. A total of 25 graphs is generated by these loops.

Examples

The following examples show different applications for DSGI and illustrate some of its features such as defining viewports and windows, inserting existing graphs, angling text, using GASK routines, enlarging a segment of a graph, and scaling a graph.

These examples use some additional graphics options that may not be used in other examples in this book. Because the dimensions of the default window vary across devices, the TARGETDEVICE=, HSIZE=, and VSIZE= graphics options are used to make the programs more portable. The COLORS= graphics option provides a standard colors list.

Refer to Chapter 50, “DATA Step Graphics Interface Dictionary,” on page 1401 for a complete description of each of the functions used in the examples.

Vertically Angling Text

This example generates a pie chart with text that changes its angle as you rotate around the pie. DSGI positions the text by aligning it differently depending on its location on the pie. In addition, DSGI changes the angle of the text so that it aligns with the spokes of the pie.

This example illustrates how global statements can be used with DSGI. In this example, FOOTNOTE and TITLE statements create the footnotes and title for the graph. The GOPTIONS statement defines general aspects of the graph. The COLORS=

graphics option provides a colors list from which the colors referenced in GSET('xxx COLOR', . . .)functions are selected.

The following program statements produce Display 49.4 on page 1383:

```

/* set the graphics environment */
goptions reset=global gunit=pct border
        ftext=swissb htitle=6 htext=3
        colors=(black blue green red)
        hsize=7 in vsize=5 in
        targetdevice=pscolor;

/* define the footnote and title */
footnotel j=r 'GDSVTEXT ';
titlel 'Text Up Vector';

/* execute DATA step with DSGI */
data vector;

        /* prepare SAS/GRAPH software */
        /* to accept DSGI statements */
rc=ginit();
rc=graph('clear');

        /* define and display arc */
        /* with intersecting lines */
rc=gset('lincolor', 2);
rc=gset('linwidth', 5);
rc=gdraw('arc', 84, 50, 35, 0, 360);
rc=gdraw('line', 2, 49, 119, 51, 51);
rc=gdraw('line', 2, 84, 84, 15, 85);

        /* define height of text */
rc=gset('texheight', 5);

        /* mark 360 degrees on the arc */
        /* using default align */
rc=gdraw('text', 121, 50, '0');

        /* set text to align to the right and */
        /* mark 180 degrees on the arc */
rc=gset('texalign', 'right', 'normal');
rc=gdraw('text', 47, 50, '180');

        /* set text to align to the center and */
        /* mark 90 and 270 degrees on the arc */
rc=gset('texalign', 'center', 'normal');
rc=gdraw('text', 84, 87, '90');
rc=gdraw('text', 84, 9, '270');

        /* reset texalign to normal and */
        /* display coordinate values or quadrant */
rc=gset('texalign', 'normal', 'normal');
rc=gdraw('text', 85, 52, '(0.0, +1.0)');

        /* rotate text using TEXUP and */

```

```

/* display coordinate values or quadrant */
rc=gset('texup', 1.0, 0.0);
rc=gdraw('text', 85, 49, '(+1.0, 0.0)');

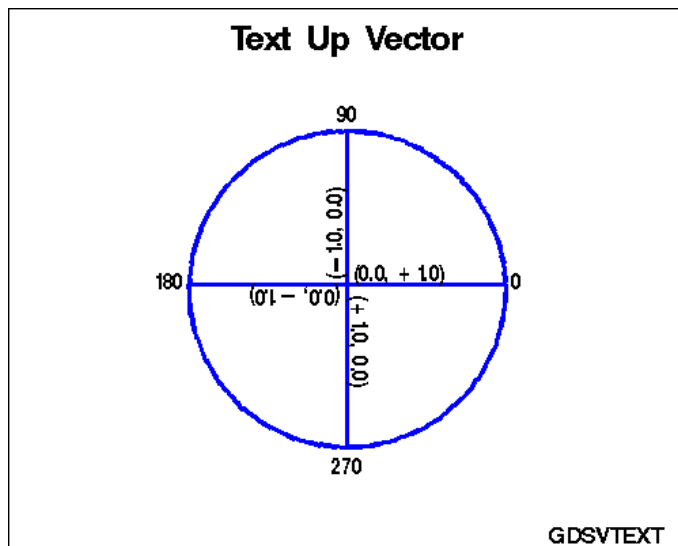
/* rotate text using TEXUP and          */
/* display coordinate values or quadrant */
rc=gset('texup', 0.0, -1.0);
rc=gdraw('text', 83, 50, '(0.0, -1.0)');

/* rotate text using TEXUP and          */
/* display coordinate values or quadrant */
rc=gset('texup', -1.0, 0.0);
rc=gdraw('text', 83, 52, '(-1.0, 0.0)');

/* display graph and end DSGI */
rc=graph('update');
rc=gterm();
run;

```

Display 49.4 Text Angled with the GSET('TEXUP', ...) Function



This example illustrates the following features:

- The `COLORS=` graphics option provides a colors table to be used with the `GSET('LINCOLOR', . . .)` function.
- The `HSIZE=` graphics option provides a standard width for the graphics output area.
- The `VSIZE=` graphics option provides a standard height for the graphics output area.
- The `TARGETDEVICE=` graphics option selects the standard color PostScript driver to use as the target device.
- The `GINIT()` function begins DSGI.
- The `GRAPH('CLEAR')` function sets the graphics environment. Because the function does not specify a name for the catalog entry, DSGI will use the default name 'DSGI'.

- The GSET('TEXHEIGHT', . . .), GSET('LINCOLOR', . . .), and GSET('LINWIDTH', . . .) functions set attributes of the graphics primitives. The COLORS= graphics option provides a colors table for the GSET('LINCOLOR', 2) function to reference. In this example, the color indexed by 2 is used to draw lines. Since no other colors table is explicitly defined with GSET('COLREP', . . .) functions, DSGI looks at the colors list and chooses the color indexed by 2 (the second color in the list) to draw the lines.
- The GDRAW('ARC', . . .) function draws an empty pie chart. The arguments of the GDRAW('ARC', . . .) function provide the coordinates of the starting point, the radius, and the beginning and ending angles of the arc.
- The GDRAW('LINE', . . .) function draws a line. It provides the type of line, the coordinates of the beginning point, and the coordinates of the ending point.
- The GDRAW('TEXT', . . .) function draws the text. It sets the coordinates of the starting point of the text string as well as the text string to be written.
- The GSET('TEXALIGN', . . .) function aligns text to the center, left, or right of the starting point specified in the GDRAW('TEXT', . . .) function.
- The GSET('TEXUP', . . .) function determines the angle at which the text is to be written.
- The GRAPH('UPDATE', . . .) function closes the graphics segment.
- The GTERM() function ends DSGI.

Changing the Reading Direction of the Text

This example changes the reading direction of text. Notice that the data set name is `_NULL_`. No data set is created as a result of this DATA step; however, the graphics output is generated. The following program statements produce Display 49.5 on page 1385:

```

/* set the graphics environment */
goptions reset=global gunit=pct border
        ftext=swissb htitle=6 htext=3
        colors=(black blue green red)
        hsize=7 in vsize=5 in
        targetdevice=pscolor;

/* define the footnote and title */
footnotel j=r 'GSDIREC ';
titlel 'Text Path';

/* execute DATA step with DSGI */
data _null_;

        /* prepare SAS/GRAPH software */
        /* to accept DSGI statements */
        rc=ginit();
        rc=graph('clear');

        /* define height of text */
        rc=gset('texheight', 5);

        /* display first text */
        rc=gdraw('text', 105, 50, 'Right');

```

```

        /* change text path so that text reads from */
        /* right to left and display next text      */
rc=gset('texpath', 'left');
rc=gdraw('text', 65, 50, 'Left');

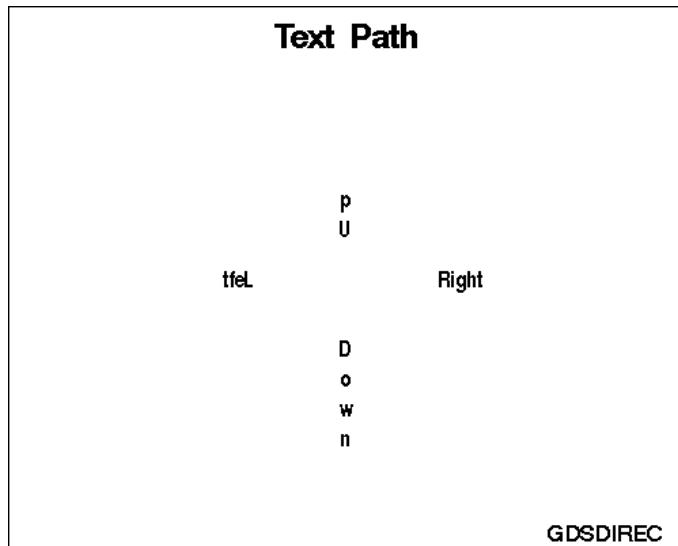
        /* change text path so that text reads up */
        /* the display and display next text      */
rc=gset('texpath', 'up');
rc=gdraw('text', 85, 60, 'Up');

        /* change text path so that text reads down */
        /* the display and display next text      */
rc=gset('texpath', 'down');
rc=gdraw('text', 85, 40, 'Down');

        /* display the graph and end DSGI */
rc=graph('update');
rc=gterm();
run;

```

Display 49.5 Reading Direction of the Text Changed with the GSET('TEXPATH', ...) Function



Features not explained earlier in "Vertically Angling Text" are described here:

- DATA_NULL_ causes the DATA step to be executed, but no data set is created.
- The GSET('TEXPATH', . . .)function changes the direction in which the text reads.

Using Viewports in DSGI

This example uses the GCHART procedure to generate a graph, defines a viewport in which to display it, and inserts the GCHART graph into the graphics output being created by DSGI. Display 49.6 on page 1387 shows the pie chart created by the GCHART procedure. Display 49.7 on page 1388 shows the same pie chart after it has been inserted into a DSGI graph.

```

/* set the graphics environment */
goptions reset=global gunit=pct border
        ftext=swissb htitle=6 htext=4
        colors=(black blue green red)
        hsize=7 in vsize=7 in
        targetdevice=pscolor;

/* create data set TOTALS */
data totals;
  length dept $ 7 site $ 8;
  do year=1996 to 1999;
    do dept='Parts','Repairs','Tools';
      do site='New York','Atlanta','Chicago','Seattle';
        sales=ranuni(97531)*10000+2000;
        output;
      end;
    end;
  end;
run;

/* define the footnote */
footnotel h=3 j=r 'GDSVWPTS ';

/* generate pie chart from TOTALS */
/* and create catalog entry PIE */
proc gchart data=totals;
  format sales dollar8.;
  pie site
    / type=sum
      sumvar=sales
      midpoints='New York' 'Chicago' 'Atlanta' 'Seattle'
      fill=solid
      cfill=green
      coutline=blue
      angle=45
      percent=inside
      value=inside
      slice=outside
      noheading
      name='GDSVWPTS';
run;

/* define the titles */
title1 'Total Sales';
title2 'For Period 1996-1999';

/* execute DATA step with DSGI */
data piein;

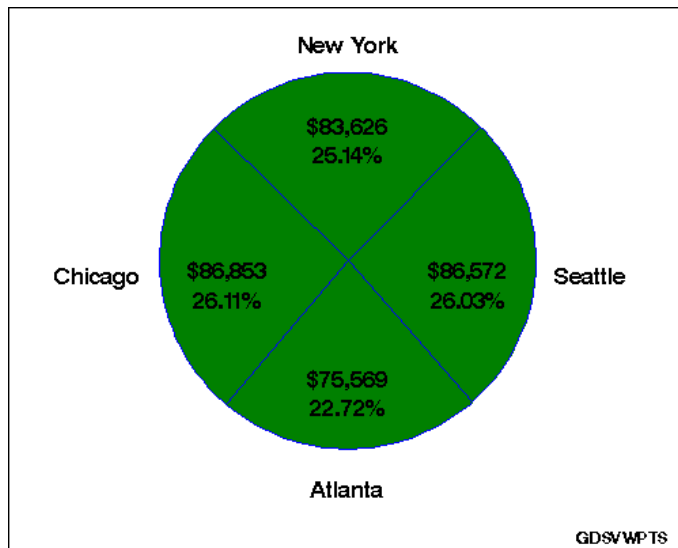
  /* prepare SAS/GRAPH software */
  /* to accept DSGI statements */
  rc=ginit();
  rc=graph('clear');

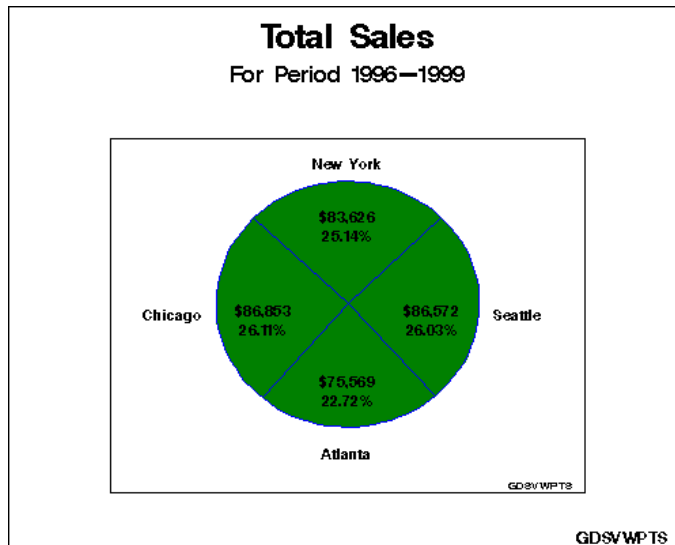
```



```
/* define and activate viewport for inserted graph */  
rc=gset('viewport', 1, .15, .05, .85, .90);  
rc=gset('window', 1, 0, 0, 100, 100);  
rc=gset('transno', 1);  
  
/* insert graph created from GCHART procedure */  
rc=graph('insert', 'GDSVWPTS');  
  
/* display graph and end DSGI */  
rc=graph('update');  
rc=gterm();  
run;
```

Display 49.6 Pie Chart Produced with the GCHART Procedure



Display 49.7 Pie Chart Inserted into DSGI Graph by Using a Viewport

Features not explained in previous examples are described here:

- A graph can be created by another SAS/GRAPH procedure and inserted into DSGI graphics output. In this case, the NAME= option in the PIE statement of the GCHART procedure names the graph, 'GDSVWPTS', to be inserted.
- The GSET('VIEWPORT', . . .)function defines the section of the graphics output area into which GDSVWPTS is inserted. The dimensional ratio of the viewport should match that of the entire graphics output area so that the inserted graph is not distorted.
- The GSET('WINDOW', . . .)function defines the coordinate system to be used within the viewport. In this example, the coordinates (0,0) to (100,100) are used. These coordinates provide a square area to insert the graph and preserve the aspect ratio of the GCHART graph.
- The GSET('TRANSNO', . . .)function activates the transformation for the defined viewport and window.
- The GRAPH('INSERT', . . .)function inserts the existing graph, 'GDSVWPTS', into the one being created with DSGI. If no viewport has been explicitly defined, DSGI inserts the graph into the default viewport, which is the entire graphics output area.

Scaling Graphs by Using Windows

This example uses the GPLOT procedure to generate a plot of AMOUNT*MONTH and store the graph in a permanent catalog. DSGI then scales the graph by defining a window in another DSGI graph and inserting the GPLOT graph into that window. Display 49.8 on page 1390 shows the plot as it is displayed with the GPLOT procedure. Display 49.9 on page 1391 shows how the same plot is displayed when the x axis is scaled from 15 to 95 and the y axis is scaled from 15 to 75.

```

/* set the graphics environment */
goptions reset=global gunit=pct border
        ftext=swissb htitle=6 htext=3
        colors=(black blue green red)
        hsize=7 in vsize=5 in

```

```

        targetdevice=pscolor;

        /* create data set EARN, which holds month */
        /* and amount of earnings for that month */
data earn;
    input month amount;
    datalines;
1 2.1
2 3
3 5
4 6.4
5 9
6 7.2
7 6
8 9.8
9 4.4
10 2.5
11 5.75
12 4.35
;
run;

        /* define the footnote for the first graph */
footnotel j=r 'GDSSCALE(a) ';

        /* define axis and symbol characteristics */
axis1 label=(color=green 'Millions of Dollars')
    order=(1 to 10 by 1)
    value=(color=green);
axis2 label=(color=green 'Months')
    order=(1 to 12 by 1)
    value=(color=green Tick=1 'Jan' Tick=2 'Feb' Tick=3 'Mar'
        Tick=4 'Apr' Tick=5 'May' Tick=6 'Jun'
        Tick=7 'Jul' Tick=8 'Aug' Tick=9 'Sep'
        Tick=10 'Oct' Tick=11 'Nov' Tick=12 'Dec');

symbol value=M font=special height=8 interpol=join
    color=blue width=3;

        /* generate a plot of AMOUNT * MONTH, */
        /* and store in member GDSSCALE */
proc gplot data=earn;
    plot amount*month
        / haxis=axis2
        vaxis=axis1
        name='GDSSCALE';
run;

        /* define the footnote and titles for */
        /* second graph, which will scale output */
footnotel j=r 'GDSSCALE(b) ';
title1 'XYZ Corporation Annual Earnings';
title2 h=4 'Fiscal Year 1999';

```

```

/* execute DATA step with DSGI using */
/* catalog entry created in previous */
/* plot, but do not create a data set */
/* (determined by specifying _NULL_) */
data _null_;

    /* prepare SAS/GRAPH software */
    /* to accept DSGI statements */
    rc=ginit();
    rc=graph('clear');

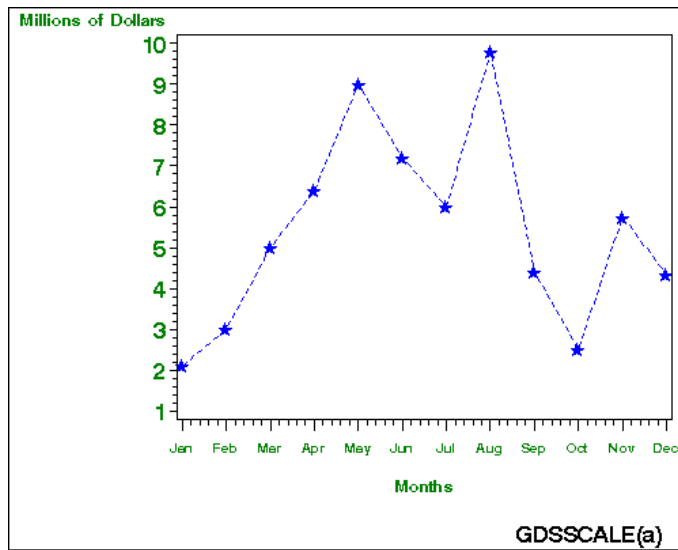
    /* define viewport and window for inserted graph */
    rc=gset('viewport', 1, .20, .30, .90, .75);
    rc=gset('window', 1, 15, 15, 95, 75);
    rc=gset('transno', 1);

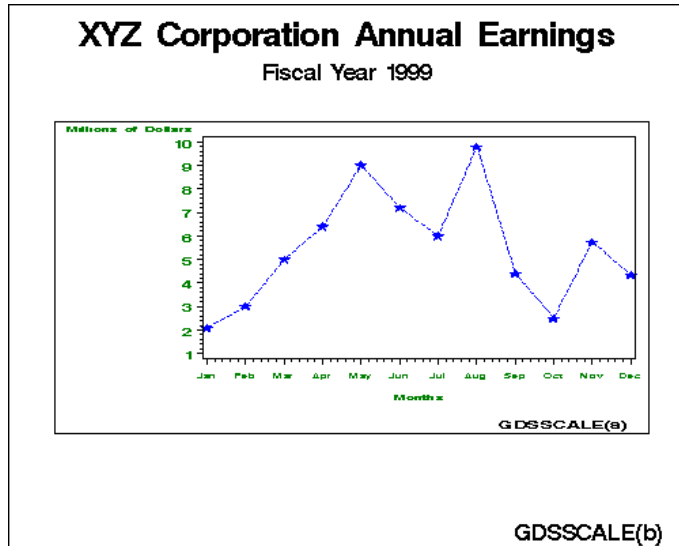
    /* insert graph previously created */
    rc=graph('insert', 'GDSSCALE');

    /* display graph and end DSGI */
    rc=graph('update');
    rc=gterm();
run;

```

Display 49.8 Plot Produced with the GPLOT Procedure



Display 49.9 Plot Scaled by Using a Window in DSGI

One feature not explained in previous examples is described here:

- The GSET('WINDOW', . . .)function scales the plot with respect to the viewport that is defined. The x axis is scaled from 15 to 95, and the y axis is scaled from 15 to 75. If no viewport were explicitly defined, the window coordinates would be mapped to the default viewport, the entire graphics output area.

Enlarging an Area of a Graph by Using Windows

This example illustrates how you can enlarge a section of a graph by using windows. In the first DATA step, the program statements generate graphics output that contains four pie charts. The second DATA step defines a window that enlarges the bottom-left quadrant of the graphics output and inserts 'GDSENLAR' into that window. The following program statements produce Display 49.10 on page 1393 from the first DATA step, and Display 49.11 on page 1393 from the second DATA step:

```

/* set the graphics environment */
goptions reset=global gunit=pct border
        ftext=swissb htext=3
        colors=(black blue green red)
        hsize=7 in vsize=5 in
        targetdevice=pscolor;

/* define the footnote for the first graph */
footnotel j=r 'GDSENLAR(a) ';

/* execute DATA step with DSGI */
data plot;

/* prepare SAS/GRAPH software */
/* to accept DSGI statements */
rc=ginit();
rc=graph('clear', 'GDSENLAR');

/* define and draw first pie chart */

```

```

rc=gset('filcolor', 4);
rc=gset('filtype', 'solid');
rc=gdraw('pie', 30, 75, 22, 0, 360);

    /* define and draw second pie chart */
rc=gset('filcolor', 1);
rc=gset('filtype', 'solid');
rc=gdraw('pie', 30, 25, 22, 0, 360);

    /* define and draw third pie chart */
rc=gset('filcolor', 3);
rc=gset('filtype', 'solid');
rc=gdraw('pie', 90, 75, 22, 0, 360);

    /* define and draw fourth pie chart */
rc=gset('filcolor', 2);
rc=gset('filtype', 'solid');
rc=gdraw('pie', 90, 25, 22, 0, 360);

    /* display graph and end DSGI */
rc=graph('update');
rc=gterm();
run;

    /* define the footnote for the second graph */
footnotel j=r 'GDSENLAR(b) ';

    /* execute DATA step with DSGI */
    /* that zooms in on a section of */
    /* the previous graph */
data zoom;

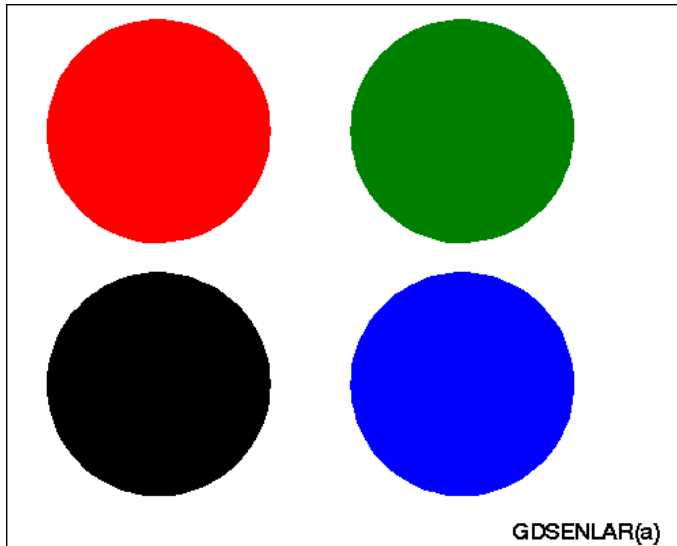
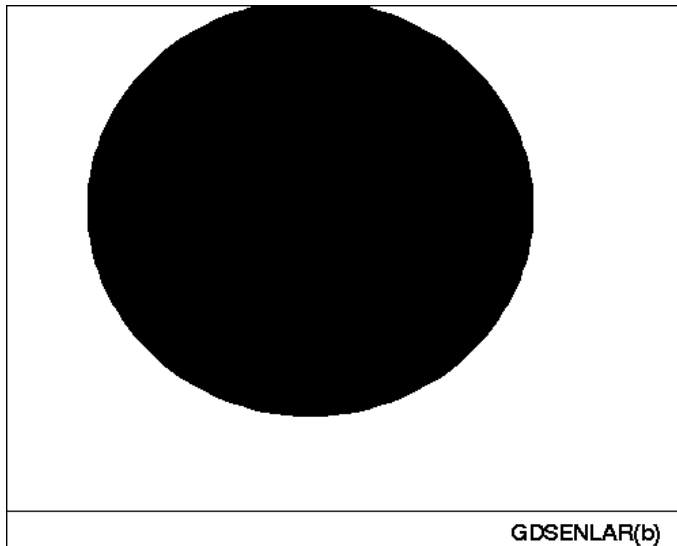
    /* prepare SAS/GRAPH software */
    /* to accept DSGI statements */
rc=ginit();
rc=graph('clear');

    /* define and activate a window */
    /* that will enlarge the lower left */
    /* quadrant of the graph */
rc=gset('window', 1, 0, 0, 50, 50);
rc=gset('transno', 1);

    /* insert the previous graph into */
    /* window 1 */
rc=graph('insert', 'GDSENLAR');

    /* display graph and end DSGI */
rc=graph('update');
rc=gterm();
run;

```

Display 49.10 Four Pie Charts Generated with DSGI**Display 49.11** Area of the Graph Enlarged by Using Windows

Features not explained in previous examples are described here:

- The `GSET('WINDOW', . . .)` function defines a window into which the graph is inserted. In this example, no viewport is defined, so the window coordinates map to the default viewport, which is the entire graphics output area. The result of using the default viewport is that only the portion of the graph enclosed by the coordinates of the window is displayed.
- The `GRAPH('INSERT', . . .)` function inserts a graph that was previously generated with DSGI. If you want to insert output created by DSGI, the output to be inserted must be closed.

Using GASK Routines in DSGI

This example illustrates how to invoke GASK routines and how to display the returned values in the SAS log and write them to a data set.

This example assigns a predefined color to color index 2 and then invokes a GASK routine to get the name of the color associated with color index 2. The value returned from the GASK call is displayed in the log and written to a data set. Output 49.1 shows how the value appears in the log. Output 49.2 shows how the value appears in the data set in the OUTPUT window.

```

/* execute DATA step with DSGI */
data routine;

    /* declare character variables used */
    /* in GASK subroutines          */
    length color $ 8;

    /* prepare SAS/GRAPH software */
    /* to accept DSGI statements  */
    rc=ginit();
    rc=graph('clear');

    /* set color for color index 2 */
    rc=gset('colrep', 2, 'orange');

    /* check color associated with color index 2 and */
    /* display the value in the LOG window          */
    call gask('colrep', 2, color, rc);
    put 'Current FILCOLOR =' color;
    output;

    /* end DSGI */
    rc=graph('update');
    rc=gterm();
run;

/* display the contents of ROUTINE */
proc print data=routine;
run;

```


Output 49.1 Checking the Color Associated with a Particular Color Index

```

3      /* execute DATA step with DSGI */
4      data routine;
5
6          /* declare character variables used */
7          /* in GASK subroutines          */
8          length color $ 8;
9
10         /* prepare SAS/GRAPH software */
11         /* to accept DSGI statements */
12         rc=ginit();
13         rc=graph('clear');
14
15         /* set color for color index 2 */
16         rc=gset('colrep', 2, 'orange');
17
18         /* check color associated with color index 2 and */
19         /* display the value in the LOG window          */
20         call gask('colrep', 2, color, rc);
21         put 'Current FILCOLOR =' color;
22         output;
23
24         /* end DSGI */
25         rc=graph('update');
26         rc=gterm();
27     run;

```

Current FILCOLOR =ORANGE

Output 49.2 Writing the Value of an Attribute to a Data Set

The SAS System	13:50 Tuesday, December 22, 1998	1
Obs	color	rc
1	ORANGE	0

Features not explained in previous examples are described here:

- The GSET('COLREP', . . .)function assigns the predefined color 'ORANGE' to the color index 2.
- GASK routines check the current value of an attribute. In this example, the GASK('COLREP', . . .)function returns the color associated with color index 2.
- A PUT statement displays the value of the COLOR argument in the log.
- An OUTPUT statement writes the value of COLOR to the ROUTINE data set.
- The GRAPH('UPDATE') function closes the graphics segment.
- The PRINT procedure displays the contents of the ROUTINE data set.

Generating a Drill-down Graph Using DSGI

This example uses ODS processing with DSGI to generate a drill-down graph. To get the drill-down capability, you use the GSET('HTML',...) function to specify a URL that points to the location of the target output. This HTML string can be used with the following graphic element types drawn in the code *after* the string is set: BAR, ELLIPSE, FILL, MARK, PIE, and TEXT. The example uses a PIE element type.

Note: The example assumes users will access the output through a file system rather than across the Web, so the HTML string uses a file specification rather than a

full URL. For information on bringing SAS/GRAPH output to the Web, see Chapter 9, “Introducing SAS/GRAPH Output for the Web,” on page 369. \triangle

This example also includes a FILENAME statement to allocate an aggregate storage location for the HTML and GIF files produced by the code. You should replace the term *path-to-Web-server* with the location where you want to store the files.

In the example, the ODS HTML statement is used to create a body file named *dsgi.htm*. When file *dsgi.htm* is viewed in a Web browser, it displays a solid pie chart, as shown in Display 49.12 on page 1397. To drill down to the graph shown in Display 49.13 on page 1398, click anywhere in the pie chart. This example uses PROC GSLIDE to create the simple graphic that is used for the target output:

```

/* This is the only line you have to */
/* change to run the program. Specify */
/* a location in your file system.    */
*filename odsout 'path-to-Web-server';

/* close the listing destination */
ods listing close;

/* set the graphics environment */
goptions reset=global gunit=pct noborder
         ftitle=swissb htitle=6
         ftext=swiss htext=3
         colors=(black blue)
         hsize=5 in vsize=5 in
         device=gif;

/* define title and footnote for graph */
title1 'Drill-down Graph';
footnotel j=1 ' Click in pie chart'
          j=r 'GSDRILL  ';

ods html body='dsgi.htm'
        path=odsout;

/* execute DATA step with DSGI */
data _null_;
  /* prepare SAS/GRAPH software */
  /* to accept DSGI statements */
  rc=ginit();
  rc=graph('clear');
  /* set a value for the html variable */
  rc=gset('html', 'href="blue.htm"');

  /* define and draw a pie chart */
  rc=gset('filcolor', 2);
  rc=gset('filtype', 'solid');
  rc=gdraw('pie', 55, 50, 22, 0, 360);

  /* generate graph and end DSGI */
  rc=graph('update');
  rc=gterm();
run;

goptions ftext=centb ctext=blue;

```

```
/* open a new body file for the */
/* target output                */
ods html body='blue.htm'
      path=odsout;

title1;
footnotel;
proc gslide wframe=4
      cframe=blue
      name='blue';
      note height=20;
      note height=10
          justify=center
          'Blue Sky';
run;
quit;

ods html close;
ods listing;
```

Display 49.12 Drill-down Graph Generated with DSGI

Drill-down Graph



Click in pie chart

GDSDRILL

Display 49.13 Target Output for Drill-down Graph

Features not explained in previous examples are described here:

- `FILENAME` allocates a storage location for the HTML and GIF files that are produced by the program.
- To conserve system resources, `ODS LISTING CLOSE` closes the Listing destination.
- On the `GOPTIONS` statement, `DEVICE=GIF` tells SAS/GRAPH to generate a GIF file for each `GRSEG` that is created in the code. The GIF files are needed to display the graphics output in a Web browser.
- On the first `ODS HTML` statement, `BODY=` specifies a name for the file that will reference the pie chart that is generated with `DSGI`. `PATH=` specifies the output location that was allocated by the `FILENAME` statement.
- In the `DATA` step, the presence of the `GSET('HTML',...)` function causes SAS/GRAPH to create the pie chart as a drill-down graph. The HTML string `'href="blue.htm"'` will be used as the value for the `HREF` attribute in the image map that SAS/GRAPH creates for the drill-down capability. The image map will be created in the body file `dsgi.htm`, because that is the file that references the pie chart. (The target output file `blue.htm` does not exist yet, but it will be created by the `GSLIDE` procedure later in the program.)
- The second `ODS HTML` file specifies a new body file. Thus, the first body file `dsgi.htm` is closed, and the new body file `blue.htm` is opened. File `blue.htm` is the file that is identified as the target output by the `HREF` value on the `GSET('HTML',...)` function.
- `PROC GSLIDE` produces the graphic that is used as the target output for the drill-down graph.
- `ODS HTML CLOSE` closes the HTML destination, and `ODS LISTING` opens the Listing destination for subsequent output during the SAS session.

See Also

“Storing Graphics Output in SAS Catalogs” on page 53
for an explanation of graphics catalogs and catalog entries

Chapter 8, “Graphics Options and Device Parameters Dictionary,” on page 261
for complete information about graphics options

“TITLE, FOOTNOTE, and NOTE Statements” on page 210
for details of using the TITLE and FOOTNOTE statements

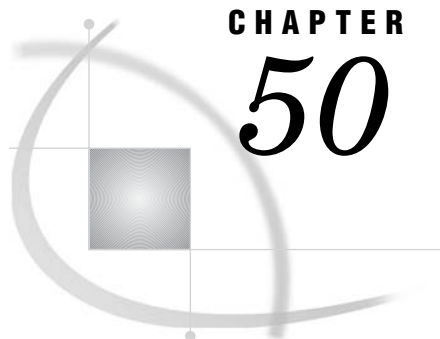
“GOPTIONS Statement” on page 146
for details of using the GOPTIONS statement

Chapter 24, “Using Annotate Data Sets,” on page 587
for an explanation of the Annotate facility

Chapter 50, “DATA Step Graphics Interface Dictionary,” on page 1401
for complete information on the functions and routines used with DSGI

SAS Language Reference: Dictionary

for information about additional functions and statements that can be used in the DATA step



CHAPTER

50

DATA Step Graphics Interface Dictionary

Overview	1401
Operating States	1402
Utility Functions	1402
GASK Routines	1404
GDRAW Functions	1446
GRAPH Functions	1457
GSET Functions	1462
Return Codes for DSGI Routines and Functions	1501
See Also	1502
References	1503

Overview

This chapter contains detailed descriptions of each command used in the DATA Step Graphics Interface (DSGI).

The following commands are associated with DSGI:

- 1 utility functions
 - GINIT
 - GPRINT
 - GTERM
- 2 GASK routines
- 3 GDRAW functions
- 4 GRAPH functions
- 5 GSET functions

Each routine or function is followed by an alphabetical listing of the operators used with it. For each operator, this chapter provides the statement syntax, other argument definitions, and notes about using the functions and routines, operating states, and return codes. Operating states are summarized in “Operating States” on page 1359.

The syntax for all routines and functions contains the argument *return-code-variable*. This argument must be a numeric variable name and can be a different variable name for each routine.

The *return-code-variable* argument is used to debug DSGI programs. It contains the return code of the routine or function call. If the return code is any value other than 0, the routine or function did not execute properly.

Each routine and function has a different set of possible return codes. The return codes are listed in the heading for the routine or function. Refer to “Return Codes for DSGI Routines and Functions” on page 1501 for an explanation of the return codes.

Operating States

This list summarizes the operating states in DSGI. For a detailed discussion of operating states, see “Operating States” on page 1359.

GKCL	facility closed, initial state of DSGI.
GKOP	facility open. DSGI is open. You may check the settings of attributes.
SGOP	segment open. Graphics output can be generated.
WSAC	workstation active. You can issue DSGI statements.
WSOP	workstation open. The graphics catalog is opened or created.

Utility Functions

Utility functions enable you to initialize a session for DSGI, print error messages, and terminate the session.

GINIT

Initializes DSGI

Operating States: GKCL

Return Codes: 0, 1, 26, 301, 307

Resulting Operating State: WSAC

Syntax

return-code-variable=GINIT();

Description

The GINIT function performs three functions: it readies the library that contains SAS/GRAPH graphics routines, it opens a workstation, and it activates it. A workstation is a Graphics Kernel Standard (GKS) concept. GKS allows for multiple workstations to be open at the same time; however, for DSGI applications, you always use exactly one workstation. This function moves the operating state from GKCL to WSAC.

See Also

“GTERM” on page 1403

GPRINT

Prints the specified interface error message

Operating States: All

Return Codes: 0

Syntax

return-code-variable=GPRINT(*code*);

Description

The GPRINT function displays the message that corresponds to the error code entered. You can use this routine if you have disabled automatic error logging but still want to display the message associated with a return code you have received.

Argument Definitions

code numeric constant or numeric variable name; should be the value of a return code received from some previous function.

See Also

“MESSAGE” on page 1485

GTERM

Terminates DSGI

Operating States: WSAC

Return Codes: 0, 3

Resulting Operating State: GKCL

Syntax

return-code-variable=GTERM();

Description

The GTERM function performs three functions: it deactivates the workstation, closes the workstation, and closes the library that contains SAS/GRAPH routines. This function should be issued to free memory allocated by DSGI. This function moves the operating state from WSAC to GKCL.

See Also

“GINIT” on page 1402

GASK Routines

When you use GASK routines, remember the following:

- All arguments are required.
- Most arguments are expressed as variable names. You can use any valid SAS variable name.
- If character arguments are expressed as character strings, they must be enclosed in quotation marks.
- All character variable names used as arguments *must* be declared in a previous LENGTH statement.
- GASK routines do not change the operating state.
- PUT statements display a value returned by a routine in the SAS log.
- OUTPUT statements write a value that is returned by a routine to a data set.

GASK routines enable you to check these current attribute settings:

ASF
ASPECT
CATALOG
CBACK
CLIP
COLINDEX
COLREP
DEVICE
FILCOLOR
FILINDEX
FILREP
FILSTYLE
FILTYPE
GRAPHLIST
HPOS
HSIZE
HTML
LINCOLOR
LININDEX
LINREP
LINTYPE
LINWIDTH
MARCOLOR
MARINDEX

MARREP
 MARSIZE
 MARTYPE
 MAXDISP
 NUMGRAPH
 OPENGRAPH
 PATREP
 STATE
 TEXALIGN
 TEXCOLOR
 TEXEXTENT
 TEXTFONT
 TEXHEIGHT
 TEXINDEX
 TEXPATH
 TEXREP
 TEXUP
 TRANS
 TRANSNO
 VIEWPORT
 VPOS
 VSIZE
 WINDOW
 WSACTIVE
 WSOPEN

ASF

Finds whether an aspect source flag is bundled or separate

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('ASF', *attribute*, *status*, *return-code-variable*);

Description

The GASK('ASF', . . .) routine returns the aspect source flag (ASF) of a particular attribute. Possible ASF values are BUNDLED (associated with a bundle index) and

INDIVIDUAL (separate from a bundle index). GASK('ASF', . . .) returns the default value INDIVIDUAL if you have not set the ASF for an attribute.

Argument Definitions

<i>attribute</i>	character string enclosed in quotes or character variable name with one of the following values: <ul style="list-style-type: none"> <input type="checkbox"/> FILCOLOR <input type="checkbox"/> FILSTYLE <input type="checkbox"/> FILTYPE <input type="checkbox"/> LINCOLOR <input type="checkbox"/> LINTYPE <input type="checkbox"/> LINWIDTH <input type="checkbox"/> MARCOLOR <input type="checkbox"/> MARSIZE <input type="checkbox"/> MARTYPE <input type="checkbox"/> TEXCOLOR <input type="checkbox"/> TEXTFONT.
<i>status</i>	character variable name; returns either the value BUNDLED or INDIVIDUAL.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

"ASF" on page 1463
 "FILCOLOR" on page 1469
 "FILSTYLE" on page 1471
 "FILTYPE" on page 1473
 "LINCOLOR" on page 1476
 "LINTYPE" on page 1479
 "LINWIDTH" on page 1479
 "MARCOLOR" on page 1480
 "MARSIZE" on page 1483
 "MARTYPE" on page 1483
 "TEXCOLOR" on page 1488
 "TEXTFONT" on page 1489

ASPECT

Finds the aspect ratio

Operating States: All

Return Codes: 0

Syntax

CALL GASK('ASPECT', *aspect*, *return-code-variable*);

Description

The GASK('ASPECT', . . .)routine returns the current aspect ratio used to draw graphics output. GASK('ASPECT', . . .)searches for the current aspect ratio in the following order:

- 1 the aspect ratio set with the GSET('ASPECT', . . .)function
- 2 the ASPECT= graphics option
- 3 the device's default aspect ratio found in the device entry. For more information on device entries, see Chapter 31, "The GDEVICE Procedure," on page 915.

Argument Definitions

aspect numeric variable name; returns the aspect ratio.*return-code-variable* numeric variable name; returns the return code of the routine call.

See Also

ASPECT= graphics option (see "ASPECT" on page 262)

"ASPECT" on page 1465

CATALOG

Finds the catalog for the graphs**Operating States:** All**Return Codes:** 0

Syntax

CALL GASK('CATALOG', *libref*, *memname*, *return-code-variable*);

Description

The GASK('CATALOG', . . .)routine returns the libref and the name of the current output catalog. GASK('CATALOG', . . .)returns the default catalog, WORK.GSEG, if no other catalog has been specified with the GSET('CATALOG', . . .)function.

Argument Definitions

<i>libref</i>	character variable name; returns the libref of the library in which the current catalog is stored.
<i>memname</i>	character variable name; returns the name of the current output catalog.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

- “CATALOG” on page 1465
- “NUMGRAPH” on page 1427
- “OPENGRAPH” on page 1428

CBACK

Finds the background color

Operating States: All

Return Codes: 0

Syntax

CALL GASK('CBACK', *cback*, *return-code-variable*);

Description

The GASK('CBACK', . . .)routine returns the current background color. GASK('CBACK', . . .)searches for the current background color in the following order:

- 1 the background color selected with the GSET('CBACK', . . .)function
- 2 the CBACK= graphics option
- 3 the default background color for the device found in the device entry. For more information about device entries, see Chapter 31, “The GDEVICE Procedure,” on page 915.

Argument Definitions

<i>cback</i>	character variable name; returns the background color name.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

- CBACK= graphics option (see “CBACK” on page 266)

“CBACK” on page 1466

CLIP

Finds whether clipping is on or off

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 55, 56

Syntax

```
CALL GASK('CLIP', status);
```

Description

The GASK('CLIP', . . .) routine checks whether clipping outside of viewports is enabled or disabled. One of the two following messages is displayed when this routine is called:

NOTE: Clipping is ON.

or

NOTE: Clipping is OFF.

Clipping is OFF by default.

Argument Definitions

status numeric variable name; returns the current setting, 55 (ON) or 56 (OFF), for clipping.

See Also

“CLIP” on page 1467

COLINDEX

Finds the color indexes that have colors associated with them

Operating States: SGOP

Return Codes: 0, 4, 86, 87

Syntax

```
CALL GASK('COLINDEX', n, index-array, return-code-variable);
```

Description

The GASK('COLINDEX', . . .)routine returns the color indexes that currently have colors assigned to them.

Argument Definitions

<i>n</i>	numeric constant or numeric variable name; tells how many color indexes you want returned. If <i>n</i> is expressed as a variable, the variable must be initialized. The variable returns the number of colors currently assigned. If <i>n</i> is expressed as a constant, it will not return this value.
<i>index-array</i>	list of numeric variables into which the used color index numbers are returned. The list of variable names can be members of an array or OF argument lists (where the arguments are variables). If you are using an array, <i>index-array</i> must have been declared as an array. The dimension of the array is determined by the number of color indexes you want returned. Refer to the discussion of ARRAY in <i>SAS Language Reference: Dictionary</i> for more information about OF argument lists.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“COLREP” on page 1410

“COLREP” on page 1467

COLREP

Finds the color name associated with a color index

Operating States: SGOP

Return Codes: 0, 4, 86, 87

Syntax

CALL GASK('COLREP', *color-index*, *color*, *return-code-variable*);

Description

The GASK('COLREP', . . .)routine returns the predefined SAS color name associated with a color index. GASK('COLREP', . . .)searches for the current color assigned to a color index in the following order:

- 1 the color selected by the GSET('COLREP', . . .)function.
- 2 the COLORS= graphics option. If *color-index* is 2, the routine returns the second color from the colors list of the COLORS= graphics option.

- 3 the device's default colors list found in the device entry. If *color-index* is 2, the routine returns the second color from the default colors list.

See “SAS Color Names and RGB Values” on page 99 for a list of SAS predefined color names.

Argument Definitions

<i>color-index</i>	numeric constant; indicates the color index for which you want to check the color. Valid values are 1 to 256, inclusive.
<i>color</i>	character variable name; returns the color name associated with <i>color-index</i> .
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“COLINDEX” on page 1409

“COLREP” on page 1467

DEVICE

Finds the output graphics device

Operating States: All

Return Codes: 0

Syntax

CALL GASK('DEVICE', *device*, *return-code-variable*);

Description

The GASK('DEVICE', . . .)routine returns the current device driver. This routine returns the device driver set by one of the following methods:

- the GSET('DEVICE', . . .)function
- the DEVICE= graphics option
- the device driver you entered in the DEVICE prompt window
- the device driver you entered in the OPTIONS window.

There is no default value for a device driver. To use DSGI, you must specify a device driver. For more information about setting device drivers, see “Selecting a Device Driver” on page 43.

Argument Definitions

<i>device</i>	character variable name; returns the name of the device driver.
---------------	---

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

DEVICE= graphics option (see “DEVICE” on page 279)
 “DEVICE” on page 1468

FILCOLOR

Finds the color index of the color to be used to draw fill areas

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('FILCOLOR', *color-index*, *return-code-variable*);

Description

The GASK('FILCOLOR', . . .)routine returns the current fill color. If a GSET('FILCOLOR', . . .)function has not been previously submitted, GASK('FILCOLOR', . . .)returns the default value, 1. The color index returned corresponds to a color specification in the following order:

- 1 the color assigned to a color name with the GSET('COLREP', . . .)function
- 2 the *n*th color in the colors list of the COLORS= graphics option
- 3 the *n*th color in the device's default colors list found in the device entry.

Argument Definitions

color-index numeric variable name; returns the color index of the fill color currently selected.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

COLORS= graphics option (see “COLORS” on page 272)
 “COLREP” on page 1410
 “COLREP” on page 1467
 “FILCOLOR” on page 1469

FILINDEX

Finds the bundle of fill area attributes that is active

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('FILINDEX', *index*, *return-code-variable*);

Description

The GASK('FILINDEX', . . .)routine asks which fill bundle is active. If no fill bundles have been previously defined with GSET('FILREP', . . .)or activated with GSET('FILINDEX', . . .), GASK('FILINDEX', . . .)returns the default value, 1.

Argument Definitions

<i>index</i>	numeric variable name; returns the index of the fill bundle currently selected.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“FILREP” on page 1413
 “FILREP” on page 1470
 “FILINDEX” on page 1470

FILREP

Finds the fill area attributes associated with a bundle index

Operating States: GKOP, WSOP, WSAC, SGOP

Return Codes: 0, 8, 75, 76

Syntax

CALL GASK ('FILREP', *index*, *color-index*, *interior*, *style-index*, *return-code-variable*);

Description

The GASK('FILREP', . . .)routine returns the color, type of interior, and fill pattern associated with a specific fill bundle. If the bundle indicated by *index* has not been

previously defined with the GSET('FILREP', . . .)function, DSGI issues the following error message:

```
ERROR: A representation for the specified fill area index has
not been defined on this workstation.
```

Argument Definitions

<i>index</i>	numeric constant or numeric variable name; indicates the fill bundle to check. Valid values are 1 to 20, inclusive. If <i>index</i> is expressed as a variable, the variable must be initialized to a value between 1 and 20.
<i>color-index</i>	numeric variable name; returns the color index of the fill color associated with the bundle. The color index that is returned corresponds to a color specification in the following order: <ol style="list-style-type: none"> 1 a color index assigned to a color name with the GSET('COLREP', . . .)function 2 the <i>n</i>th color in the colors list of the COLORS= graphics option 3 the <i>n</i>th color in the device's default colors list found in the device entry.
<i>interior</i>	character variable name; returns the style of the interior associated with the bundle index – that is, one of the following values: <ul style="list-style-type: none"> □ HATCH □ HOLLOW □ PATTERN □ SOLID.
<i>style-index</i>	numeric variable name; returns the index of the fill pattern associated with the bundle. See the "FILSTYLE" on page 1471 for the fill patterns represented by <i>style-index</i> .
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

COLORS= graphics option (see "COLORS" on page 272)
 "FILINDEX" on page 1413
 "COLREP" on page 1467
 "FILREP" on page 1470
 "FILSTYLE" on page 1471

FILSTYLE

Finds the style of the fill area when FILTYPE is PATTERN or HATCH

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('FILSTYLE', *style-index*, *return-code-variable*);

Description

The GASK('FILSTYLE', . . .)routine returns the current fill style of the interior when FILTYPE is PATTERN or HATCH. If no fill style has been previously selected with the GSET('FILSTYLE', . . .)function, GASK('FILSTYLE', . . .)returns the default value, 1.

Argument Definitions

<i>style-index</i>	numeric variable name; returns the index of the fill pattern associated with the bundle. See the "FILSTYLE" on page 1471 for the interior styles represented by <i>style-index</i> .
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

- "FILTYPE" on page 1415
- "FILSTYLE" on page 1471
- "FILTYPE" on page 1473

FILTYPE

Finds the type of the interior of the fill area

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('FILTYPE', *interior*, *return-code-variable*);

Description

The GASK('FILTYPE', . . .)routine returns the current fill type. If no fill type has been previously selected with the GSET('FILTYPE', . . .)function, GASK('FILTYPE', . . .)returns the default value, HOLLOW.

Argument Definitions

<i>interior</i>	character variable name; returns the fill type that is active, that is, one of the following values:
-----------------	--

- HATCH
- HOLLOW
- PATTERN
- SOLID.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

“FILSTYLE” on page 1414

“FILTYPE” on page 1415

GRAPHLIST

Finds the names of segments in the current catalog

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('GRAPHLIST', *n*, *name-array*, *return-code-variable*);

Description

The GASK('GRAPHLIST', . . .)routine lists the first *n* names of the graphs that are in the current catalog. If a catalog has not been previously specified with the GRAPH('CATALOG', . . .)function, the routine returns names from the default catalog, WORK.GSEG.

The names returned are any of the following:

- those specified in the GRAPH('CLEAR', . . .)function
- if the name is omitted from the GRAPH('CLEAR' . . .)function, some form of DSGI: for example, DSGI, DSGI1, or DSGI2.
- the name specified in the NAME= option of a graphics procedure
- graphs previously created by other graphics procedures and already in the catalog.

Argument Definitions

n numeric variable name; tells the maximum number of graph names you want returned. If you express *n* as a variable, the variable must be initialized to the maximum number of graph names you want returned.

name-array list of character variable names into which the graph names will be returned. The list of variable names can be members of an array or OF argument lists (where the arguments are variables). If you are using an array, *name-array* must be declared as an array. The

dimension of the array is determined by the number of color indexes you want returned. See the discussion for ARRAY in *SAS Language Reference: Dictionary* for more information about OF argument lists.

return-code-variable

numeric variable names; returns the return code of the routine call.

See Also

“CLEAR” on page 1457

HPOS

Finds the number of columns

Operating States: All

Return Codes: 0

Syntax

CALL GASK('HPOS', *hpos*, *return-code-variable*);

Description

The GASK('HPOS', . . .)routine returns the number of columns currently in the graphics output area. GASK('HPOS', . . .)searches for the current number of columns in the following order:

- 1 the value selected in the GSET('HPOS', . . .)function
- 2 the value of the HPOS= graphics option
- 3 the device's default HPOS value found in the device entry.

Argument Definitions

hpos numeric variable name; returns the number of columns in the graphics output area.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

“HSIZE” on page 1418

“HPOS” on page 1474

HPOS= graphics option (see “HPOS” on page 315)

HSIZE

Finds the horizontal dimension of the graphics output area

Operating States: All

Return Codes: 0

Syntax

CALL GASK('HSIZE', *hsize*, *return-code-variable*);

Description

The GASK('HSIZE', . . .)routine returns the current horizontal dimension, in inches, of the graphics output area. GASK('HSIZE', . . .)searches for the current horizontal dimension in the following order:

- 1 the value selected in the GSET('HSIZE', . . .)function
- 2 the value of the HSIZE= graphics option
- 3 the device's default HSIZE found in the device entry.

Argument Definitions

hsize numeric variable name; the size of the graphics output area in the *x* dimension (in inches).

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

"HPOS" on page 1417

"HSIZE" on page 1475

HSIZE= graphics option (see "HSIZE" on page 315)

HTML

Finds the HTML string that is in effect when one of the following graphic elements is drawn: bar, ellipse, fill, mark, pie, and text.

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('HTML', *string*, *return-code-variable*);

Description

The GASK('HTML', . . .)routine returns the current HTML string. If a GSET('HTML', . . .)function has not been previously submitted, GASK('HTML', . . .)returns the default value, null.

Argument Definitions

string the HTML string invoked when an affected DSGI graphic element in a web page is clicked.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

“BAR” on page 1448
 “ELLIPSE” on page 1450
 “FILL” on page 1451
 “MARK” on page 1453
 “PIE” on page 1455
 “TEXT” on page 1456
 “HTML” on page 1475

LINCOLOR

Finds the current setting of the color to be used to draw lines

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('LINCOLOR', *color-index*, *return-code-variable*);

Description

The GASK('LINCOLOR', . . .)routine returns the current line color. If a GSET('LINCOLOR', . . .)function has not been previously submitted, GASK('LINCOLOR', . . .)returns the default value, 1. The color index returned corresponds to a color specification in the following order:

- 1 the color specified in a GSET('COLREP', . . .)function
- 2 the *n*th color in the colors list of the COLORS= graphics option
- 3 the *n*th color in the device's default colors list.

Argument Definitions

color-index numeric variable name; returns the color index of the current line color.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

COLORS= graphics option (see “COLORS” on page 272)

“COLREP” on page 1410

“COLREP” on page 1467

“LINCOLOR” on page 1476

LININDEX

Finds the index of the bundle of line attributes

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('LININDEX', *index*, *return-code-variable*);

Description

The GASK('LININDEX', . . .)routine returns the current line bundle. If no line bundles have been previously defined with GSET('LINREP', . . .)or activated with GSET('LININDEX', . . .), GASK('LININDEX', . . .)returns the default value, 1.

Argument Definitions

index numeric variable name; returns the index of the current line bundle.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

“LINREP” on page 1420

“LININDEX” on page 1477

“LINREP” on page 1478

LINREP

Finds the bundle of line attributes associated with an index

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 60, 61

Syntax

CALL GASK ('LINREP', *index*, *color-index*, *width*, *type*, *return-code-variable*);

Description

The GASK('LINREP', . . .)routine returns the color, width, and line type associated with a specific line bundle. If the bundle indicated by *index* has not been previously defined with the GSET('LINREP', . . .)function, DSGI issues the following error message:

```
ERROR: A representation for the specified line type index has
       not been defined on this workstation.
```

Argument Definitions

<i>index</i>	numeric constant or numeric variable name; indicates the fill bundle to check. Valid values are 1 to 20, inclusive. If <i>index</i> is expressed as a variable, the variable must be initialized to a value between 1 and 20.
<i>color-index</i>	numeric variable name; returns the color index of the fill color associated with the bundle. The color index returned corresponds to a color specification in the following order: <ol style="list-style-type: none"> 1 a color index assigned with the GSET('COLREP', . . .)function 2 the <i>n</i>th color in the colors list of the COLORS= graphics option 3 the <i>n</i>th color in the device's default colors list.
<i>width</i>	numeric variable name; returns the line width (in pixels) associated with the bundle.
<i>type</i>	numeric variable name; returns the index of the line type associated with the bundle. Refer to Figure 7.22 on page 208 for representations of the line types.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

COLORS= graphics option (see "COLORS" on page 272)
 "COLREP" on page 1410
 "LININDEX" on page 1420
 "COLREP" on page 1467
 "LINREP" on page 1478

LINTYPE

Finds the line type

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('LINTYPE', *type*, *return-code-variable*);

Description

The GASK('LINTYPE', . . .)routine returns the current line type. If no line type was previously selected with the GSET('LINTYPE', . . .)function, GASK('LINTYPE', . . .)returns the default value, 1.

Argument Definitions

type numeric variable name; returns the index of the line type currently selected. Refer to Figure 7.22 on page 208 for representations of the line types.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

“LINTYPE” on page 1479

LINWIDTH

Finds the line thickness

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('LINWIDTH', *width*, *return-code-variable*);

Description

The GASK('LINWIDTH', . . .)routine returns the current line width. If a line width has not been previously selected with the GSET('LINWIDTH', . . .)function, GASK('LINWIDTH', . . .)returns the default value, 1.

Argument Definitions

width numeric variable name; returns the current line width (in units of pixels).

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

“LINWIDTH” on page 1479

MARCOLOR

Finds the color index of the color to be used to draw markers

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('MARCOLOR', *color-index*, *return-code-variable*);

Description

The GASK('MARCOLOR', . . .)routine returns the current marker color. If a GSET('MARCOLOR', . . .)function has not been previously submitted, GASK('MARCOLOR', . . .)returns the default value, 1. The color index returned corresponds to a color specification in the following order:

- 1 the color selected in a GSET('COLREP', . . .)function
- 2 the *n*th color in the colors list of the COLORS= graphics option
- 3 the *n*th color in the device's default colors list.

Argument Definitions

color-index numeric variable name; returns the color index of the current marker color.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

COLORS= graphics option (see “COLORS” on page 272)

“COLREP” on page 1410

“COLREP” on page 1467

“MARCOLOR” on page 1480

MARINDEX

Finds the index of the bundle of marker attributes currently selected

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('MARINDEX', *index*, *return-code-variable*);

Description

The GASK('MARINDEX', . . .)routine returns the current marker bundle. If no marker bundles have been previously defined with GSET('MARREP', . . .)or activated with GSET('MARINDEX', . . .), GASK('MARINDEX', . . .)returns the default value, 1.

Argument Definitions

<i>index</i>	numeric variable name; returns the index of the marker bundle currently selected.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“MARREP” on page 1424

“MARINDEX” on page 1481

“MARREP” on page 1482

MARREP

Finds the bundle of marker attributes associated with an index

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 64, 65

Syntax

CALL GASK('MARREP', *index*, *color-index*, *size*, *type*, *return-code-variable*);

Description

The GASK('MARREP' . . .)routine returns the color, size, and type of marker associated with a specific marker bundle. If the bundle indicated by *index* has not been

previously defined with the GSET('MARREP', . . .)function, DSGI issues the following error message:

```
ERROR: A representation for the specified marker index has
       not been defined on this workstation.
```

Argument Definitions

<i>index</i>	numeric constant or numeric variable name; indicates the index of the fill bundle to check. Valid values are 1 to 20, inclusive. If <i>index</i> is expressed as a variable, the variable must be initialized to a value between 1 and 20.
<i>color-index</i>	numeric variable name; returns the color index of the fill color associated with the bundle. The color index returned corresponds to a color specification in the following order: <ol style="list-style-type: none"> 1 a color index assigned with the GSET('COLREP', . . .)function 2 the <i>n</i>th color in the colors list of the COLORS= graphics option 3 the <i>n</i>th color in the device's default colors list.
<i>size</i>	numeric variable name; returns the marker size in units of the current window system.
<i>type</i>	numeric variable name; the index of the marker type associated with the bundle. See the "MARTYPE" on page 1483 for an explanation of the marker indexes.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

COLORS= graphics option (see "COLORS" on page 272)
 "COLREP" on page 1410
 "COLREP" on page 1467
 "MARINDEX" on page 1481
 "MARREP" on page 1482
 "MARTYPE" on page 1483

MARSIZE

Finds the size of markers

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('MARSIZE', *size*, *return-code-variable*);

Description

The GASK('MARSIZE', . . .)routine returns the current marker size. If no marker size has been previously selected with the GSET('MARSIZE', . . .)function, GASK('MARSIZE', . . .)returns the default value, 1.

Argument Definitions

<i>size</i>	numeric variable name; returns the marker size in units of the current window system.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“MARSIZE” on page 1483

MARTYPE

Finds the kind of markers

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('MARTYPE', *type*, *return-code-variable*);

Description

The GASK('MARTYPE', . . .)routine returns the current marker type. If no marker type has been previously selected with the GSET('MARTYPE', . . .)function, GASK('MARTYPE', . . .)returns the default value, 1.

Argument Definitions

<i>type</i>	numeric variable name; returns the index of the marker type currently selected. See the function “MARTYPE” on page 1483 for an explanation of the indexes for markers.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“MARTYPE” on page 1483

MAXDISP

Finds the maximum display area size

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK ('MAXDISP', *units*, *x-dim*, *y-dim*, *x-pixels*, *y-pixels*, *return-code-variable*);

Description

The GASK('MAXDISP', . . .)routine returns the dimensions of the maximum display area for the device. This routine is useful when you need to know the maximum display area in order to determine the aspect ratio or to scale a graph.

There is a difference between the maximum display size returned when the operating state is not SGOP and when it is SGOP. The full addressable display area is returned when the operating state is not SGOP, and the display area minus room for titles and footnotes is returned when the operating state is SGOP.

Argument Definitions

<i>units</i>	numeric variable name; returns a 1 to show that <i>x-dim</i> and <i>y-dim</i> are in meters.
<i>x-dim</i>	numeric variable name; returns the dimension, in meters, in the <i>x</i> direction.
<i>y-dim</i>	numeric variable name; returns the dimension, in meters, in the <i>y</i> direction.
<i>x-pixels</i>	numeric variable name; returns the number of pixels in the <i>x</i> direction.
<i>y-pixels</i>	numeric variable name; returns the number of pixels in the <i>y</i> direction.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“HSIZE” on page 1418

“VSIZE” on page 1443

“HSIZE” on page 1475

“VSIZE” on page 1499

NUMGRAPH

Finds the number of graphs in the current catalog

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('NUMGRAPH', *n*, *return-code-variable*);

Description

The GASK('NUMGRAPH', . . .)routine returns how many graphs are in the current catalog. The catalog checked is the catalog selected in the GSET('CATALOG', . . .)function, if specified; otherwise, it is the default catalog, WORK.GSEG.

Argument Definitions

<i>n</i>	numeric variable name; returns the number of graphs in the current catalog.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“CATALOG” on page 1407

“CATALOG” on page 1465

OPENGRAPH

Finds the name of the segment currently open

Operating States: SGOP

Return Codes: 0, 4

Syntax

CALL GASK('OPENGRAPH', *name*, *return-code-variable*);

Description

The GASK('OPENGRAPH', . . .)routine returns the name of the graph that is currently open.

The name returned is one of the following:

- the name specified in the GRAPH('CLEAR', . . .)function
- if the name is omitted from the GRAPH('CLEAR', . . .)function, some form of DSGI: for example, DSGI, DSGI1, and DSGI2.

Argument Definitions

<i>name</i>	character variable name; returns the name of the graph that is currently open.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“CLEAR” on page 1457

PATREP

Finds the pattern name assigned to a style index

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 79

Syntax

CALL GASK('PATREP', *index*, *pattern-name*, *hatch-name*, *return-code-variable*);

Description

The GASK('PATREP', . . .)routine returns the pattern name assigned to a style index.

Argument Definitions

<i>index</i>	numeric variable name; returns the index of the pattern currently selected.
<i>pattern-name</i>	character variable name; returns the name of the pattern at the specified index.
<i>hatch-name</i>	character variable name; returns the name of the hatch at the specified index.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“PATREP” on page 1485

STATE

Finds the current operating state of DSGI

Operating States: All

Return Codes: 0

Syntax

CALL GASK('STATE', *status*);

Description

The GASK('STATE', . . .) routine returns the current operating state of DSGI.

Argument Definitions

status character variable name; returns one of the following values:

- GKCL
- GKOP
- SGOP
- WSAC
- WSOP.

See Also

“WSACTIVE” on page 1445

“WSOPEN” on page 1445

TEXALIGN

Finds the horizontal and vertical alignment of the text string

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('TEXALIGN', *halign*, *valign*, *return-code-variable*);

Description

The GASK('TEXALIGN', . . .)routine returns the current horizontal and vertical text alignment. If no values have been previously selected with the GSET('TEXALIGN', . . .)function, GASK('TEXALIGN', . . .)returns the default value NORMAL for both *halign* and *valign*.

Argument Definitions

<i>halign</i>	character variable name; indicates the horizontal alignment set by the GSET('TEXALIGN', . . .)function; returns one of the following values: <ul style="list-style-type: none"> <input type="checkbox"/> CENTER <input type="checkbox"/> LEFT <input type="checkbox"/> NORMAL <input type="checkbox"/> RIGHT.
<i>valign</i>	character variable name; indicates the vertical alignment set by the GSET('TEXALIGN', . . .)function; returns one of the following values: <ul style="list-style-type: none"> <input type="checkbox"/> BASE <input type="checkbox"/> BOTTOM <input type="checkbox"/> HALF <input type="checkbox"/> NORMAL <input type="checkbox"/> TOP.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“TEXPATH” on page 1436

“TEXUP” on page 1438

“TEXALIGN” on page 1486

TEXCOLOR

Finds the color index of the color currently selected to draw text strings

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('TEXCOLOR', *color-index*, *return-code-variable*);

Description

The `GASK('TEXCOLOR', . . .)` routine returns the current text color. If a `GSET('TEXCOLOR', . . .)` function has not been previously submitted, `GASK('TEXCOLOR', . . .)` returns the default value, 1. The color index returned corresponds to a color specification in the following order:

- 1 the color specified in a `GSET('COLREP', . . .)` function
- 2 the *n*th color in the colors list of the `COLORS=` graphics option
- 3 the *n*th color in the device's default colors list.

Argument Definitions

<i>color-index</i>	numeric variable name; returns the color index of the color used to draw text.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

COLORS= graphics option (see “COLORS” on page 272)
 “COLREP” on page 1410
 “COLREP” on page 1467
 “TEXCOLOR” on page 1488

TEXEXTENT

Finds the text extent rectangle and concatenation point for a specified text string

Operating States: SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

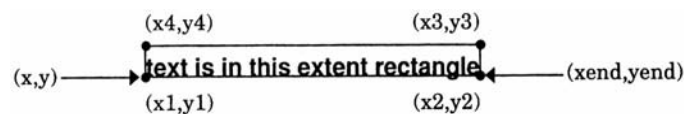
CALL GASK ('TEXEXTENT', *x*, *y*, *string*, *x-end*, *y-end*, *x1*, *x2*, *x3*, *x4*, *y1*, *y2*, *y3*, *y4*, *return-code-variable*);

Description

The GASK('TEXEXTENT', . . .)routine returns the text extent rectangle and text concatenation point for a specified text string. All text extent coordinates returned are in units of the current window system. If no text string is specified for *string*, GASK('TEXEXTENT', . . .)does not return values for the other arguments.

The text attributes and bundles affect the values returned by this query. See Figure 50.1 on page 1433 for a diagram of the text extent rectangle (in the figure, *x,y* is always the place where the text string starts).

Figure 50.1 Text Extent Diagram



Argument Definitions

- x* numeric variable name; *x* coordinates are in units based on the current window system; returns *x* coordinate after justification. The variable used to specify *x* must be initialized.
- y* numeric variable name; *y* coordinates are in units based on the current window system; returns *y* coordinate after justification. The variable used to specify *y* must be initialized.

<i>string</i>	character string enclosed in single quotation marks or a character variable name; a set of characters for which the text extent rectangle and text concatenation point are calculated.
<i>x-end</i>	numeric variable name; returns the <i>x</i> coordinate of the point at which the next text string may be concatenated.
<i>y-end</i>	numeric variable name; returns the <i>y</i> coordinate of the point at which the next text string may be concatenated.
<i>x1, x2, x3, x4,</i> <i>y1, y2, y3, y4</i>	numeric variable names; return the text extent rectangles of the text strings as shown in Figure 50.1 on page 1433.
<i>return-code-</i> <i>variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“WINDOW” on page 1444

“TEXT” on page 1456

TEXFONT

Finds the font used to draw text strings

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('TEXFONT', *font*, *return-code-variable*);

Description

The GASK('TEXFONT' . . .)routine returns the current text font. GASK('TEXFONT', . . .)searches for the current font in the following order:

- 1 the value selected in the GSET('TEXFONT', . . .)function, if specified
- 2 the value of the FTEXT= graphics option, if specified
- 3 the device's default hardware font if the device supports a hardware font
- 4 the SIMULATE font.

Argument Definitions

<i>font</i>	character variable name; returns the font name.
<i>return-code-</i> <i>variable</i>	numeric variable name; returns the return code of the routine call.

See Also

FTEXT= graphics options in (see “FTEXT” on page 294)
 “TEXTFONT” on page 1489

TEXHEIGHT

Finds the character height of the text strings

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('TEXHEIGHT', *height*, *return-code-variable*);

Description

The GASK('TEXHEIGHT', . . .)routine returns the current text height.
 GASK('TEXHEIGHT', . . .)searches for the current text height in the following order:

- 1 the value selected in the GSET('TEXHEIGHT', . . .)function, if specified
- 2 the value of the HTEXT= graphics option, if specified
- 3 the default text height, 1.

Argument Definitions

<i>height</i>	numeric variable name; returns the character height in units of the current window system.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“TEXHEIGHT” on page 1490

HTEXT= graphics options (see “HTEXT” on page 316)

TEXINDEX

Finds the index of the bundle of text attributes currently selected

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('TEXINDEX', *index*, *return-code-variable*);

Description

The GASK('TEXINDEX', . . .)routine returns the current text bundle. If no text bundles have been previously defined with GSET('TEXREP', . . .)or activated with GSET('TEXINDEX', . . .), GASK('TEXINDEX', . . .)returns the default value, 1.

Argument Definitions

index numeric variable name; returns the text bundle index.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

“TEXREP” on page 1437

“TEXREP” on page 1492

“TEXINDEX” on page 1490

TEXPATH

Finds the direction of the text string

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

```
CALL GASK('TEXPATH', path, return-code-variable);
```

Description

The GASK('TEXPATH', . . .)routine returns the current text path (reading direction). If TEXPATH has not been previously selected with the GSET('TEXPATH', . . .)function, GASK('TEXPATH', . . .)returns the default value, RIGHT. See the "TEXPATH" on page 1491 for an illustration of text paths.

Argument Definitions

path character variable name; returns one of the following values:

- DOWN
- LEFT
- RIGHT
- UP.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

"TEXALIGN" on page 1430

"TEXUP" on page 1438

"TEXPATH" on page 1491

TEXREP

Finds the attribute settings associated with a text bundle

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 68, 69

Syntax

```
CALL GASK('TEXREP', index, color-index, font, return-code-variable);
```

Description

The GASK('TEXREP', . . .)routine returns the color and font associated with a specific text bundle. If the bundle indicated by *index* has not been previously defined with the GSET('TEXREP', . . .)function, DSGI issues the following error message:

```
ERROR: A representation for the specified text index has
not been defined on this workstation.
```

Argument Definitions

<i>index</i>	numeric constant or numeric variable name; indicates the fill bundle to check. Valid values are 1 to 20, inclusive. If <i>index</i> is expressed as a variable, the variable must be initialized to a value between 1 and 20.
<i>color-index</i>	numeric variable name; returns the color index of the fill color associated with the bundle. The color index that is returned corresponds to a color specification in the following order: <ol style="list-style-type: none"> 1 a color index assigned with the GSET('COLREP', . . .)function 2 the <i>n</i>th color in the colors list of the COLORS= graphics option 3 the <i>n</i>th color in the device's default colors list.
<i>font</i>	character variable name; returns the text font associated with the bundle.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

COLORS= graphics option (see "COLORS" on page 272)
 "COLREP" on page 1410
 "COLREP" on page 1467
 "TEXREP" on page 1492

TEXUP

Finds the orientation (angle) of the text string

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('TEXUP', *up-x*, *up-y*, *return-code-variable*);

Description

The GASK('TEXUP', . . .)routine returns the character up vector values. If TEXUP has not been previously selected with the GSET('TEXUP', . . .)function, GASK('TEXUP', . . .)returns the default values for *x* and *y*, 0 and 1. See the "TEXUP" on page 1493 for an explanation of the vector values.

Argument Definitions

<i>up-x</i>	numeric variable name; returns the <i>x</i> component of the vector.
<i>up-y</i>	numeric variable name; returns the <i>y</i> component of the vector.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

“TEXALIGN” on page 1430

“TEXPATH” on page 1436

“TEXUP” on page 1493

TRANS

Finds the viewport and window coordinates associated with a transformation number

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 50

Syntax

CALL GASK ('TRANS', *n*, *vllx*, *vlly*, *vurx*, *vury*, *wllx*, *wlly*, *wurx*, *wury*, *return-code-variable*);

Description

The GASK('TRANS', . . .)routine returns the viewport and window coordinates associated with a particular transformation number. GASK('TRANS', . . .)returns the default coordinates for viewports and windows if other coordinates have not been defined for the transformation specified.

Argument Definitions

<i>n</i>	numeric constant or numeric variable name; indicates the number of the transformation to check. Valid values are 0 to 20, inclusive. If <i>n</i> is expressed as a variable, the variable must be initialized to a value between 0 and 20.
<i>vllx</i>	numeric variable name; returns the <i>x</i> coordinate of the lower-left viewport corner.
<i>vlly</i>	numeric variable name; returns the <i>y</i> coordinate of the lower-left viewport corner.
<i>vurx</i>	numeric variable name; returns the <i>x</i> coordinate of the upper-right viewport corner.
<i>vury</i>	numeric variable name; returns the <i>y</i> coordinate of the upper-right viewport corner.
<i>wllx</i>	numeric variable name; returns the <i>x</i> coordinate of the lower-left window corner.
<i>wlly</i>	numeric variable name; returns the <i>y</i> coordinate of the lower-left window corner.

<i>wurx</i>	numeric variable name; returns the <i>x</i> coordinate of the upper-right window corner.
<i>wury</i>	numeric variable name; returns the <i>y</i> coordinate of the upper-right window corner.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“TRANSNO” on page 1440

“VIEWPORT” on page 1441

“WINDOW” on page 1444

“TRANSNO” on page 1496

“VIEWPORT” on page 1497

“WINDOW” on page 1500

TRANSNO

Finds the number of the transformation to be used

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Syntax

CALL GASK('TRANSNO', *n*, *return-code-variable*);

Description

The GASK('TRANSNO', . . .)routine returns the current transformation. If a transformation has not been previously selected with the GSET('TRANSNO', . . .)function, GASK('TRANSNO', . . .)returns the number of the default transformation, 0.

Argument Definitions

<i>n</i>	numeric variable name; returns the number of the current transformation.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“TRANS” on page 1439

“VIEWPORT” on page 1441

“WINDOW” on page 1444

“VIEWPORT” on page 1497

“WINDOW” on page 1500

“TRANSNO” on page 1496

VIEWPORT

Finds coordinates of the viewport associated with a transformation number

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 50

Syntax

CALL GASK('VIEWPORT', *n*, *llx*, *lly*, *urx*, *ury*, *return-code-variable*);

Description

The GASK('VIEWPORT', . . .)routine returns the coordinates of the viewport associated with the specified transformation. If a viewport has not been defined with the GSET('VIEWPORT', . . .)function for the specified transformation, *n*, GASK('VIEWPORT', . . .)returns the default coordinates for the viewport, (0,0) and (1,1).

Argument Definitions

<i>n</i>	numeric constant or numeric variable name; indicates the transformation number assigned to the viewport to check. Valid values are 0 to 20, inclusive. If <i>n</i> is expressed as a variable, the variable must be initialized to a value between 0 and 20.
<i>llx</i>	numeric variable name; returns the <i>x</i> coordinate of the lower-left corner.
<i>lly</i>	numeric variable name; returns the <i>y</i> coordinate of the lower-left corner.
<i>urx</i>	numeric variable name; returns the <i>x</i> coordinate of the upper-right corner.
<i>ury</i>	numeric variable name; returns the <i>y</i> coordinate of the upper-right corner.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“TRANS” on page 1439

“TRANSNO” on page 1440

“WINDOW” on page 1444

“TRANSNO” on page 1496

“VIEWPORT” on page 1497

“WINDOW” on page 1500

VPOS

Finds the number of rows

Operating States: All

Return Codes: 0

Syntax

CALL GASK('VPOS', *vpos*, *return-code-variable*);

Description

The GASK('VPOS', . . .)routine returns the current number of rows in the graphics output area. GASK('VPOS', . . .)searches for the current number of rows in the following order:

- 1 the value selected in the GSET('VPOS', . . .)function
- 2 the value of the VPOS= graphics option
- 3 the device's default VPOS value found in the device entry.

Argument Definitions

vpos numeric variable name; returns the number of rows in the graphics output area.

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

“HPOS” on page 1417

“VSIZE” on page 1443

“VPOS” on page 1498

VPOS= graphics option (see “VPOS” on page 360)

VSIZE

Finds the vertical dimension of the graphics output area

Operating States: All

Return Codes: 0

Syntax

CALL GASK('VSIZE', *vsiz*, *return-code-variable*);

Description

The GASK('VSIZE', . . .)routine returns the current vertical dimension, in inches, of the graphics output area. GASK('VSIZE', . . .)searches for the current vertical dimension in the following order:

- 1 the value selected in the GSET('VSIZE', . . .)function
- 2 the value of the VSIZE= graphics option
- 3 the device's default VSIZE found in the device entry.

Argument Definitions

vsiz numeric variable name; returns the size of the graphics output area in the *y* dimension (in inches).

return-code-variable numeric variable name; returns the return code of the routine call.

See Also

“HSIZE” on page 1418

“VPOS” on page 1442

“VSIZE” on page 1499

VSIZE= graphics option (see “VSIZE” on page 361)

WINDOW

Finds the coordinates of the window associated with a transformation number

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 50

Syntax

CALL GASK('WINDOW', *n*, *llx*, *lly*, *urx*, *ury*, *return-code-variable*);

Description

The GASK('WINDOW', . . .)routine returns the coordinates of the window associated with the specified transformation number. If no window has been defined with the GSET('WINDOW', . . .)function for transformation *n*, GASK('WINDOW', . . .)returns the default window coordinates, which are device dependent.

Argument Definitions

<i>n</i>	numeric constant or numeric variable name; indicates the transformation number of the window to check. Valid values are 0 to 20, inclusive. If <i>n</i> is expressed as a variable, the variable must be initialized to a value between 0 and 20.
<i>llx</i>	numeric variable name; returns the <i>x</i> coordinate of the lower-left corner.
<i>lly</i>	numeric variable name; returns the <i>y</i> coordinate of the lower-left corner.
<i>urx</i>	numeric variable name; returns the <i>x</i> coordinate of the upper-right corner.
<i>ury</i>	numeric variable name; returns the <i>y</i> coordinate of the upper-right corner.
<i>return-code-variable</i>	numeric variable name; returns the return code of the routine call.

See Also

“TRANS” on page 1439

“TRANSNO” on page 1440

“VIEWPORT” on page 1441

“TRANSNO” on page 1496

“VIEWPORT” on page 1497

“WINDOW” on page 1500

WSACTIVE

Finds whether the interface is active

Operating States: All

Return Codes: 29, 30

Syntax

```
CALL GASK('WSACTIVE', status);
```

Description

The GASK('WSACTIVE', . . .)routine asks if the workstation is active. When the workstation is active, you can execute certain DSGI routines and functions.

Argument Definitions

status numeric variable name; returns either 29 (active) or 30 (inactive).

See Also

“STATE” on page 1430

“WSOPEN” on page 1445

WSOPEN

Finds whether the interface is open

Operating States: All

Return Codes: 24, 25

Syntax

```
CALL GASK('WSOPEN', status);
```

Description

The GASK('WSOPEN', . . .)routine asks if the workstation is open. If a workstation is open, the graphics catalog can be accessed.

Argument Definitions

status numeric variable name; returns either 24 (open) or 25 (closed).

See Also

“WSACTIVE” on page 1445

GDRAW Functions

GDRAW functions create graphics elements. Each GDRAW operator is associated with a set of GSET operators that control its attributes. For example, the color, height, and font for the GDRAW('TEXT', . . .)function are controlled by GSET('TEXCOLOR', . . .), GSET('TEXHEIGHT', . . .), and GSET('TEXFONT', . . .), respectively. For a complete list of the attributes associated with each GDRAW function, see Table 49.2 on page 1364. The complete graph is displayed after the GRAPH('UPDATE', . . .)function is submitted.

When using GDRAW functions, remember the following:

- All arguments must be specified.
- All arguments are specified as variables or constants. If you express an argument as a variable, the variable must be initialized.
- All character arguments that are expressed as character strings must be enclosed in quotes.
- All character variable names used as arguments *must* be declared in a LENGTH statement.
- All character constants must be enclosed in single or double quotes.

GDRAW functions:

- ARC
- BAR
- ELLARC
- ELLIPSE
- FILL
- IMAGE
- LINE
- MARK
- MESSAGE
- PIE
- TEXT

ARC

Draws a circular arc

Operating States: SGOP

Return Codes: 0, 4, 61, 86

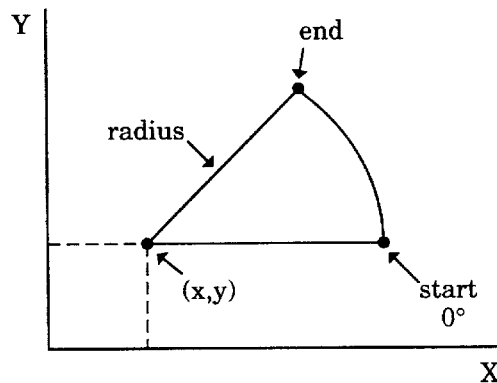
Syntax

return-code-variable=GDRAW('ARC', *x*, *y*, *radius*, *start*, *end*);

Description

The `GDRAW('ARC', . . .)` function draws a circular arc. The line attributes and bundles affect the appearance of this primitive. See Table 49.2 on page 1364 for a list of these attributes. Figure 50.2 on page 1447 illustrates the arguments used with `GDRAW('ARC', . . .)`.

Figure 50.2 Arguments Used with the `GDRAW('ARC', ...)` Function



Argument Definitions

<i>x</i>	numeric constant or numeric variable name; specifies the <i>x</i> coordinate of the position of the arc on the display; <i>x</i> coordinates are in units based on the current window system.
<i>y</i>	numeric constant or numeric variable name; specifies the <i>y</i> coordinate of the position of the arc on the display; <i>y</i> coordinates are in units based on the current window system;
<i>radius</i>	numeric constant or numeric variable name; the arc radius size is in units based on the current window system.
<i>start</i>	numeric constant or numeric variable name; the starting angle of the arc is in degrees, with 0 degrees at 3 o'clock.
<i>end</i>	numeric constant or numeric variable name; the ending angle of the arc is in degrees, with 0 degrees at 3 o'clock.

See Also

- “ELLARC” on page 1449
- “PIE” on page 1455
- “LINCOLOR” on page 1476
- “LININDEX” on page 1477
- “LINREP” on page 1478
- “LINTYPE” on page 1479
- “LINWIDTH” on page 1479

BAR

Draws a rectangle

Operating States: SGOP

Return Codes: 0, 4, 76, 79, 80, 86

Syntax

return-code-variable=GDRAW('BAR', *x1*, *y1*, *x2*, *y2*);

Description

The GDRAW('BAR', . . .)function draws a rectangular bar whose sides are parallel to the sides of the display area. The fill attributes and bundles affect the appearance of this graphics element. See Table 49.2 on page 1364 for a list of these attributes. Figure 50.3 on page 1448 illustrates the arguments used with GDRAW('BAR', . . .).

Figure 50.3 Points that Draw a Bar



Argument Definitions

- | | |
|-----------|--|
| <i>x1</i> | numeric constant or numeric variable name; refers to the <i>x</i> coordinate of one corner of the bar. |
| <i>y1</i> | numeric constant or numeric variable name; refers to the <i>y</i> coordinate of one corner of the bar. |
| <i>x2</i> | numeric constant or numeric variable name; refers to the <i>x</i> coordinate of the corner of the bar that is diagonally opposite to the corner of $(x1,y1)$. |
| <i>y2</i> | numeric constant or numeric variable name; refers to the <i>y</i> coordinate of the corner of the bar that is diagonally opposite to the corner of $(x1,y1)$. |

See Also

- “FILL” on page 1451
- “FILCOLOR” on page 1469
- “FILINDEX” on page 1470

“FILREP” on page 1470
 “FILTYPE” on page 1473
 “FILSTYLE” on page 1471
 “HTML” on page 1475

ELLARC

Draws an elliptical arc

Operating States: SGOP

Return Codes: 0, 4, 61, 86

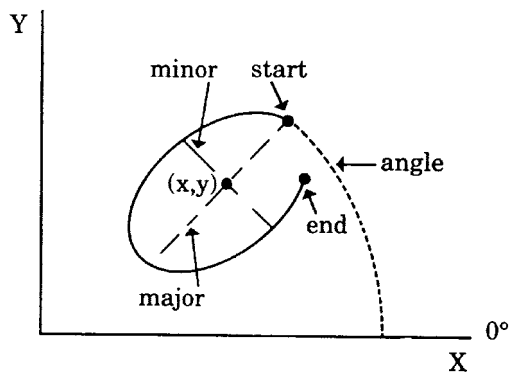
Syntax

return-code-variable =GDRAW('ELLARC', *x*, *y*, *major*, *minor*, *start*, *end*, *angle*);

Description

The GDRAW('ELLARC', . . .)function draws a hollow section of an ellipse. The line attributes and bundles affect the appearance of this primitive. See Table 49.2 on page 1364 for a list of these attributes. Figure 50.4 on page 1449 illustrates the arguments used with GDRAW('ELLARC', . . .)and GDRAW('ELLIPSE', . . .).

Figure 50.4 Arguments Used with GDRAW('ELLARC',...) function and GDRAW('ELLIPSE',...) function



Argument Definitions

- | | |
|--------------|--|
| <i>x</i> | numeric constant or numeric variable name; <i>x</i> coordinates are in units based on the current window system. |
| <i>y</i> | numeric constant or numeric variable name; <i>y</i> coordinates are in units based on the current window system. |
| <i>major</i> | numeric constant or numeric variable name; the major axis lengths for the elliptical arc. |

<i>minor</i>	numeric constant or numeric variable name; the minor axis lengths for the elliptical arc.
<i>start</i>	numeric constant or numeric variable name; the starting angle from the major axis, in degrees, for the elliptical arc with 0 degrees beginning at the major axis.
<i>end</i>	numeric constant or numeric variable name; the ending angle from the major axis, in degrees, for the elliptical arc with 0 degrees at 3 o'clock.
<i>angle</i>	numeric constant or numeric variable name; the angle that the major axis of the elliptical arc has to 0 degrees (with 0 degrees at 3 o'clock).

See Also

“ELLIPSE” on page 1450

“LINCOLOR” on page 1476

“LINTYPE” on page 1479

“LINWIDTH” on page 1479

“LINREP” on page 1478

“LININDEX” on page 1477

ELLIPSE

Draws an ellipse

Operating States: SGOP

Return Codes: 0, 4, 76, 79, 80, 86

Syntax

return-code-variable =GDRAW('ELLIPSE', *x*, *y*, *major*, *minor*, *start*, *end*, *angle*);

Description

The GDRAW('ELLIPSE', . . .)function draws a filled section of an ellipse. The fill attributes and bundles affect the appearance of this primitive. See Table 49.2 on page 1364 for a list of these attributes. Figure 50.4 on page 1449 illustrates the arguments used with GDRAW('ELLARC', . . .)and GDRAW('ELLIPSE', . . .).

Argument Definitions

<i>x</i>	numeric constant or numeric variable name; the <i>x</i> coordinate of the position of the ellipse on the display.
<i>y</i>	numeric constant or numeric variable name; the <i>y</i> coordinate of the position of the ellipse on the display.

<i>major</i>	numeric constant or numeric variable name; the major axis length for the ellipse.
<i>minor</i>	numeric constant or numeric variable name; the minor axis length for the ellipse.
<i>start</i>	numeric constant or numeric variable name; the starting angle for the ellipse from the major axis, with 0 degrees beginning at the major axis.
<i>end</i>	numeric constant or numeric variable name; the ending angle for the ellipse from the major axis, with 0 degrees at 3 o'clock.
<i>angle</i>	numeric constant or numeric variable name; the angle that the major axis of the ellipse has to 0 degrees, with 0 degrees at 3 o'clock.

See Also

“ELLARC” on page 1449
 “FILCOLOR” on page 1469
 “FILINDEX” on page 1470
 “FILREP” on page 1470
 “FILTYPE” on page 1473
 “HTML” on page 1475

FILL

Draws a filled area

Operating States: SGOP

Return Codes: 0, 4, 76, 79, 80, 86, 100, 301

Syntax

return-code-variable=GDRAW('FILL', *n*, *x-values*, *y-values*);

Description

The GDRAW('ILL' . . .)function draws a filled polygon. The fill attributes and bundles affect the appearance of this primitive. See Table 49.2 on page 1364 for a list of these attributes.

Note: All of the *x* coordinates are listed in the function first, followed by the *y* coordinates. This primitive takes the first *n* values and stores them as *x* coordinates. The next *n* values are stored as *y* coordinates. △

Argument Definitions

n numeric constant or numeric variable name; the number of vertices (*x* and *y* pairs) in the polygon. You can specify a missing value (.) for

	<i>n</i> . If <i>n</i> is missing, the number of vertices is computed from the number of <i>x</i> and <i>y</i> arguments.
<i>x-values</i>	list of numeric constants, variables, or OF arguments that describe the <i>x</i> coordinates for the vertices in units based on the current window system.
<i>y-values</i>	list of numeric constants, variables, or OF arguments that describe the <i>y</i> coordinates for the vertices in units based on the current window system.

See Also

“BAR” on page 1448
 “FILCOLOR” on page 1469
 “FILINDEX” on page 1470
 “FILREP” on page 1470
 “FILTYPE” on page 1473
 “FILSTYLE” on page 1471
 “HTML” on page 1475

IMAGE

Displays an image

Operating State: SGOP

Return Codes: 0, 150

Syntax

return-code-variable=GDRAW('IMAGE', 'external-file', *x1*, *y1*, *x2*, *y2*, 'style');

Description

The GDRAW('IMAGE', . . .) function displays the specified image within opposing pairs of coordinates. The format of the external image file varies between operating environments. The (*x1*, *y1*) coordinate pair specifies one corner of the image, and the (*x2*, *y2*) coordinate pair specifies the opposite corner of the image. The *style* parameter must be either 'TILE' to copy the image as many times as necessary to fill the area, or 'FIT' to stretch one instance of the image to fill the area.

LINE

Draws a polyline

Operating States: SGOP**Return Codes:** 0, 4, 61, 86, 100, 301

Syntax

return-code-variable=GDRAW('LINE', *n*, *x-values*, *y-values*);

Description

The GDRAW('LINE' . . .)function draws one line, a series of connected lines, or a dot. The line attributes and bundles affect the appearance of this primitive. See Table 49.2 on page 1364 for a list of these attributes.

Note: All of the *x* coordinates are listed in the function first, followed by the *y* coordinates. This primitive takes the first *n* values and stores them as *x* coordinates and the next *n* values and stores them as *y* coordinates. △

Argument Definitions

<i>n</i>	numeric constant or numeric variable name; the number of vertices (<i>x</i> and <i>y</i> pairs) in the polygon. You can specify a missing value (.) for <i>n</i> . If <i>n</i> is missing, the number of vertices is computed from the number of <i>x</i> and <i>y</i> pairs.
<i>x-values</i>	list of numeric constants, variables, or OF arguments that describe the <i>x</i> coordinates for the vertices in units based on the current window system.
<i>y-values</i>	list of numeric constants, variables, or OF argument lists that describe the <i>y</i> coordinates for the vertices in units based on the current window system.

See Also

"FILCOLOR" on page 1469

"LININDEX" on page 1477

"LINREP" on page 1478

"LINTYPE" on page 1479

"LINWIDTH" on page 1479

MARK

Draws a polymarker

Operating States: SGOP**Return Codes:** 0, 4, 65, 86, 100, 301

Syntax

return-code-variable=GDRAW ('MARK', *n*, *x-values*, *y-values*);

Description

The GDRAW('MARK', . . .)function draws a series of symbols. The marker attributes and bundles affect the appearance of this primitive. See Table 49.2 on page 1364 for a list of these attributes. Refer to the “MARTYPE” on page 1483 for a list of symbols that you can draw with GDRAW('MARK', . . .).

Note: All of the *x* coordinates are listed in the function first, followed by the *y* coordinates. This primitive takes the first *n* values and stores them as *x* coordinates and the next *n* values and stores them as *y* coordinates. △

Argument Definitions

<i>n</i>	numeric constant or numeric variable name; the number of times the symbol is drawn. You can specify a missing value (.) for <i>n</i> . If <i>n</i> is missing, the number of vertices is calculated from the number of <i>x</i> and <i>y</i> pairs.
<i>x-values</i>	list of numeric constants, variables, or OF arguments that describe the <i>x</i> coordinates of the symbols in units based on the current window system.
<i>y-values</i>	list of numeric constants, variables, or OF arguments that describe the <i>y</i> coordinates of the symbols in units based on the current window system.

See Also

“TEXT” on page 1456
 “HTML” on page 1475
 “MARCOLOR” on page 1480
 “MARINDEX” on page 1481
 “MARREP” on page 1482
 “MARTYPE” on page 1483

MESSAGE

Prints a message in the SAS log

Operating States: All

Return Codes: 0

Syntax

return-code-variable=GDRAW('MESSAGE', *message*);

Description

The GDRAW('MESSAGE', . . .)function prints a message in the SAS log. This function may be used for debugging applications or for printing custom messages for your application.

Argument Definitions

message character string enclosed in quotes or character variable name; the text to be printed in the log.

See Also

“MESSAGE” on page 1485

“GPRINT” on page 1402

PIE

Draws a filled circle or section of a filled circle

Operating States: SGOP

Return Codes: 0, 4, 76, 79, 80, 86

Syntax

return-code-variable=GDRAW('PIE', *x*, *y*, *radius*, *start*, *end*);

Description

The GDRAW('PIE', . . .)function draws a filled section of a circular arc. The fill attributes and bundles affect the appearance of this primitive. See Table 49.2 on page 1364 for a list of these attributes.

Argument Definitions

x numeric constant or numeric variable name; *x* coordinates are in units based on the current window system.

y numeric constant or numeric variable name; *y* coordinates are in units based on the current window system.

radius numeric constant or numeric variable name; the pie radius size in units based on the current window system.

start numeric constant or numeric variable name; the starting angle of the pie, with 0 degrees at 3 o'clock on the unit circle.

end numeric constant or numeric variable name; the ending angle of the pie, with 0 degrees at 3 o'clock on the unit circle.

See Also

“ARC” on page 1446
 “FILCOLOR” on page 1469
 “FILINDEX” on page 1470
 “FILREP” on page 1470
 “FILTYPE” on page 1473
 “FILSTYLE” on page 1471
 “HTML” on page 1475

TEXT

Draws a text string

Operating States: SGOP

Return Codes: 0, 4, 69, 86

Syntax

return-code-variable=GDRAW("TEXT", *x*, *y*, *string*);

Description

The GDRAW("TEXT", . . .)function draws a text string. The text attributes and bundles affect the appearance of this primitive. See Table 49.2 on page 1364 for a list of these attributes.

Argument Definitions

<i>x</i>	numeric constant or numeric variable name; <i>x</i> coordinates are in units based on the current window system.
<i>y</i>	numeric constant or numeric variable name; <i>y</i> coordinates are in units based on the current window system.
<i>string</i>	character string enclosed in quotes or character variable name; a set of characters to be drawn on the output beginning at position (<i>x</i> , <i>y</i>).

See Also

“MARK” on page 1453
 “HTML” on page 1475
 “TEXCOLOR” on page 1488
 “TEXINDEX” on page 1490
 “TEXREP” on page 1492
 “TEXHEIGHT” on page 1490

GRAPH Functions

GRAPH functions perform library management tasks from within the DATA Step Graphics Interface. These functions can only be performed on one catalog at a time. They cannot be performed across catalogs. For example, you cannot copy a graph from one catalog to another.

When using GRAPH functions, remember the following:

- All arguments are specified as variables or constants. If you express an argument as a variable, the variable must be initialized.
- All character arguments expressed as character strings must be enclosed in quotes.
- All character variable names used as arguments *must* be declared in a LENGTH statement.
- All character constants must be enclosed in single or double quotes.

GRAPH functions:

- CLEAR
- COPY
- DELETE
- INSERT
- PLAY
- RENAME
- UPDATE

CLEAR

Opens a graphics segment for output

Operating States: WSAC

Return Codes: 0, 3, 301, 302

Resulting Operating State: SGOP

Syntax

return-code-variable=GRAPH ('CLEAR'<, *name*> <, *des*><, *byline*>);

Description

The GRAPH('CLEAR', . . .)function opens a graphics segment for output in the current catalog. The first parameter, 'CLEAR', is the only required one. The values of *name*, *des*, and *byline* are displayed in catalog listings and in catalog information in the GREPLAY procedure.

If the name specified is an existing graph, DSGI will suffix the name with a number. For example, if PIE is chosen for the name and it already exists, DSGI will name the

output PIE1; the next time the code is submitted, DSGI names the output PIE2, and so forth.

This function moves the operating state from WSAC to SGOP.

Argument Definitions

<i>name</i>	character string enclosed in quotes or character variable name; gives a name to the graph to be opened. If <i>name</i> is not specified, DSGI assigns the graph a name that is some form of DSGI: for example, DSGI, DSGI1, and DSGI2.
<i>des</i>	character string enclosed in quotes or character variable name; gives a description to the graph to be opened. If <i>des</i> is not specified, DSGI assigns the following description to the catalog entry: Graph from DATA Step Graphics Interface.
<i>by-line</i>	character string enclosed in quotes or character variable name; gives another line of description for the graph. The byline appears under the titles on the graph. DSGI does not provide a default byline.

See Also

“OPENGRAPH” on page 1428

“UPDATE” on page 1461

COPY

Copies a graph

Operating States: GKOP, WSOP, WSAC, SGOP

Return Codes: 0, 8, 307

Syntax

return-code-variable=GRAPH('COPY', *name*, *new-name*);

Description

The GRAPH('COPY', . . .)function copies a graph to another catalog entry. The graph to be copied must be closed and be in the current catalog. You cannot copy from one catalog to another. The new graph will also be in the current catalog.

Argument Definitions

<i>name</i>	character string enclosed in quotes or character variable name; name of the graph to be copied.
<i>new-name</i>	character string enclosed in quotes or character variable name; name of the graph to be created.

See Also

“CATALOG” on page 1407

“DELETE” on page 1459

“INSERT” on page 1459

“CATALOG” on page 1465

DELETE

Deletes a graph

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 4, 8, 307

Syntax

return-code-variable=GRAPH('DELETE', *name*);

Description

The GRAPH('DELETE', . . .)function deletes a graph in the current catalog. The graph does not have to be closed to be deleted.

Argument Definitions

name character string enclosed in quotes or character variable name; the name of the graph to delete.

See Also

“CATALOG” on page 1407

“COPY” on page 1458

“CATALOG” on page 1465

INSERT

Inserts a previously created segment into the currently open graph

Operating States: SGOP

Return Codes: 0, 4, 302, 307

Syntax

return-code-variable=GRAPH('INSERT', *name*);

Description

The GRAPH('INSERT', . . .)function inserts a graph into the currently open graph. The graph to be inserted must be closed and be in the current catalog.

Argument Definitions

name character string enclosed in quotes or character variable name; the name of a graph to be inserted.

See Also

“CATALOG” on page 1407

“COPY” on page 1458

“CATALOG” on page 1465

PLAY

Displays the specified graph on your output

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 307

Syntax

return-code-variable=GRAPH('PLAY', *graph-name*);

Description

The GRAPH('PLAY', . . .)function displays the specified graph on your output.

Argument Definitions

graph-name character variable name; the name of the graph you would like to play.

See Also

“UPDATE” on page 1461

RENAME

Renames a graph

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 307

Syntax

return-code-variable=GRAPH('RENAME', *name*, *new-name*);

Description

The GRAPH('RENAME', . . .)function renames a graph. The graph to be renamed must be in the current catalog and be closed.

Argument Definitions

name character string enclosed in quotes or character variable name; the name of the closed graph that is to be changed.

new-name character string enclosed in quotes or character variable name; the new name for the graph.

See Also

“CATALOG” on page 1407

“INSERT” on page 1459

“CATALOG” on page 1465

UPDATE

Completes the currently open graph and (optionally) displays it

Operating States: SGOP

Return Codes: 0, 4

Resulting Operating State: WSAC

Syntax

return-code-variable=GRAPH('UPDATE' <, 'show'>);

Description

The GRAPH('UPDATE', . . .)function closes the graph currently open and displays it. DSGI operates in buffered mode, so the picture is never displayed until this function is called.

This function can be called only once for the currently open graph. Therefore, you cannot incrementally build a graph; however, you can close the currently open graph and later insert it into another graph within the same DATA step.

This function moves the operating state from SGOP to WSAC.

Argument Definitions

show character string, optional; valid values are SHOW and NOSHOW. If SHOW is specified, the graph is displayed. If NOSHOW is specified, the graph is closed and not displayed.

See Also

“CLEAR” on page 1457

GSET Functions

GSET functions allow you to set attributes for the graphics elements. Some GSET functions set the attributes for a subset of graphics primitives. For example, attributes prefixed by FIL control the appearance of the graphics primitives GDRAW('BAR', . . .), GDRAW('ELLIPSE', . . .), GDRAW('FILL', . . .), and GDRAW('PIE', . . .). See Table 49.2 on page 1364 for a complete list of the attributes that control the appearance of the graphics primitives.

Some GSET functions affect the appearance of the entire graphics output. For example, GSET('HPOS', . . .) and GSET('VPOS', . . .) set the number of columns and rows for the output. See each GSET function for the aspect of the graphics output that it controls.

When using GSET functions, remember the following:

- All arguments must be specified.
- All arguments are specified as variables or constants. If you express an argument as a variable, the variable must be initialized.
- All character arguments that are expressed as character strings must be enclosed in quotation marks.
- All character variable names used as arguments *must* be declared in a LENGTH statement.
- All character constants must be enclosed in single or double quotation marks.

GSET functions:

ASF

ASPECT

CATALOG

CBACK

CLIP

COLREP

DEVICE

FILCOLOR

FILINDEX

FILREP

FILSTYLE

FILTYPE

HPOS
HSIZE
HTML
LINCOLOR
LININDEX
LINREP
LINTYPE
LINWIDTH
MARCOLOR
MARINDEX
MARREP
MARSIZE
MARTYPE
MESSAGE
PATREP
TEXALIGN
TEXCOLOR
TEXFONT
TEXHEIGHT
TEXINDEX
TEXPATH
TEXREP
TEXUP
TRANSNO
VIEWPORT
VPOS
VSIZE
WINDOW

ASF

Specifies an aspect source flag to bundle or separate attributes

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Default Value: INDIVIDUAL

Syntax

return-code-variable=GSET('ASF', *attribute*, *status*);

Description

The GSET('ASF', . . .)function sets an attribute's aspect source flag (ASF) so that it can be used in a bundle (BUNDLED) or individually (INDIVIDUAL).

If an attribute's ASF is set to 'BUNDLED', it cannot be used outside of a bundle. It must be defined in a GSET('xxxREP', . . .)function and activated with a GSET('xxxINDEX', . . .)function, where *xxx* can have one of the following values: FIL, LIN, MAR, TEX.

If an attribute's ASF is set to 'INDIVIDUAL', it cannot be used with a bundle. In this case, the attribute is set with a GSET('attribute', . . .). The values of *attribute* are listed in "Argument Definitions."

Argument Definitions

<i>attribute</i>	<p>character string enclosed in quotes or character variable name with one of the following values:</p> <ul style="list-style-type: none"> □ FILCOLOR □ FILSTYLE □ FILTYPE □ LINCOLOR □ LINTYPE □ LINWIDTH □ MARCOLOR □ MARSIZE □ MARTYPE □ TEXCOLOR □ TEXTFONT.
<i>status</i>	<p>character string enclosed in quotation marks or character variable name; accepts either the value BUNDLED or INDIVIDUAL.</p>

See Also

- “ASF” on page 1405
- “FILCOLOR” on page 1469
- “FILSTYLE” on page 1471
- “FILTYPE” on page 1473
- “LINCOLOR” on page 1476
- “LINTYPE” on page 1479
- “LINWIDTH” on page 1479
- “MARCOLOR” on page 1480
- “MARSIZE” on page 1483
- “MARTYPE” on page 1483
- “TEXCOLOR” on page 1488

“TEXFONT” on page 1489

ASPECT

Specifies the aspect ratio

Operating States: GKCL

Return Codes: 0, 1, 90, 307

Default Value: 0.0

Syntax

return-code-variable=GSET('ASPECT', *aspect*);

Description

The GSET('ASPECT', . . .)function sets the aspect ratio used to draw graphics output. GSET('ASPECT', . . .)affects only pies, arcs, and software text.

Argument Definitions

aspect numeric constant or numeric variable name; specifies the aspect ratio and cannot be less than 0.

See Also

ASPECT= graphics option (see “ASPECT” on page 262)

“ASPECT” on page 1406

CATALOG

Specifies the catalog for the graphs

Operating States: GKCL

Return Codes: 0, 1

Default Values: *libref* = WORK, *catalog-name*=GSEG

Syntax

return-code-variable=GSET('CATALOG', *libref*, *catalog-name*);

Description

The GSET('CATALOG', . . .)function makes the specified catalog the current catalog in which to store graphs generated with DSGI. GSET('CATALOG', . . .)creates the catalog if it does not exist.

The values of *libref* and *catalog-name* cannot exceed eight characters. The number of characters allowed for a catalog name varies across operating environments; see the SAS companion for your operating system. *Libref* should have been defined through the LIBNAME statement.

Argument Definitions

<i>libref</i>	character string enclosed in quotation marks or character variable name; points to the library that contains the catalog.
<i>catalog-name</i>	character string enclosed in quotation marks or character variable name; specifies the catalog name to be used.

See Also

“CATALOG” on page 1407

“GRAPHLIST” on page 1416

“NUMGRAPH” on page 1427

CBACK

Specifies the background color

Operating States: GKCL

Return Codes: 0, 1

Default Value: 1. CBACK= graphics option, if specified; 2. device’s default background color.

Syntax

```
return-code-variable=GSET('CBACK', cback);
```

Description

The GSET('CBACK', . . .)function sets the background color. GSET('CBACK', . . .)has the same effect as the CBACK= graphics option.

Argument Definitions

<i>cback</i>	character string enclosed in quotation marks or character variable name; can contain any predefined SAS color name. See “SAS Color Names and RGB Values” on page 99 for a list of predefined SAS color names.
--------------	---

See Also

CBACK= graphics option (see “CBACK” on page 266)

“CBACK” on page 1408

CLIP

Specifies whether clipping is on or off

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0

Default Value: OFF

Syntax

return-code-variable=GSET('CLIP', *status*);

Description

The GSET('CLIP', . . .)function activates or suppresses clipping around the current viewport.

Argument Definitions

<i>status</i>	character string enclosed in quotation marks or character variable name; valid values are ON and OFF. When ON is used, the graphics elements outside of the specified viewport are not displayed. If you turn clipping OFF, the graphics elements outside of the defined viewport are displayed.
---------------	--

See Also

“CLIP” on page 1409

“VIEWPORT” on page 1441

“VIEWPORT” on page 1497

COLREP

Associates a color name with a certain color index

Operating States: SGOP

Return Codes: 0, 4, 86

Default Values: 1. colors list of COLORS= graphics option; 2. device's default colors list

Syntax

```
return-code-variable=GSET('COLREP', color-index, color);
```

Description

The GSET('COLREP', . . .)function associates a predefined SAS color name with a color index. Many of the GASK routines and GSET functions use *color-index* as an argument.

If this function is not used, DSGI searches for a color specification in the following order:

- 1 the *n*th color in the colors list of the COLORS= graphics option
- 2 the *n*th color in the device's default colors list.

Argument Definitions

color-index numeric constant or numeric variable name; a number from 1 to 256 that identifies a color.

color character string enclosed in quotation marks or character variable name; a predefined SAS color name. See "SAS Color Names and RGB Values" on page 99 for a list of predefined SAS color names.

See Also

COLORS= graphics option (see "COLORS" on page 272)

"COLINDEX" on page 1409

"COLREP" on page 1410

DEVICE

Specifies the output graphics device

Operating States: GKCL

Return Codes: 0, 1

Default Value: 1. DEVICE= graphics option, if specified; 2. value entered in DEVICE prompt window; 3. value entered in OPTIONS window

Syntax

```
return-code-variable=GSET('DEVICE', device);
```

Description

The GSET('DEVICE', . . .)function selects the device driver.

Argument Definitions

device character string enclosed in quotation marks or character variable name; the name of the driver you will be using. *Device* must match

one of the device entries in the SASHELP.DEVICES catalog or one of your personal device catalogs, GDEVICE0.DEVICES through GDEVICE9.DEVICES. Refer to “About Device Catalogs” on page 916 for more information about catalogs that store device entries.

See Also

DEVICE= graphics option (see “DEVICE” on page 279)
 “DEVICE” on page 1411

FILCOLOR

Specifies the color index of the color used to draw fill areas

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 85

Default Value: 1

Syntax

return-code-variable=GSET('FILCOLOR', *color-index*);

Description

The GSET('FILCOLOR', . . .)function selects the color index of the color used to draw fill areas. The aspect source flag (ASF) of FILCOLOR must be set to 'INDIVIDUAL' for this attribute to be used outside of a fill bundle.

DSGI searches for a color to assign to the index in the following order:

- 1 the color specified for the index in a GSET('COLREP', . . .)function
- 2 the *n*th color in the colors list of the COLORS= graphics option
- 3 the *n*th color in the device's default colors list found in the device entry.

Argument Definitions

color-index numeric constant or numeric variable name; indicates the index of the color to be used. Valid values are 1 to 256, inclusive.

See Also

COLORS= graphics option (see “COLORS” on page 272)
 “ASF” on page 1463
 “COLREP” on page 1467
 “FILCOLOR” on page 1412
 “FILREP” on page 1470

FILINDEX

Specifies the index of the bundle of fill area attributes

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 75

Default Value: 1

Syntax

return-code-variable=GSET('FILINDEX', *index*);

Description

The GSET('FILINDEX', . . .)function activates a particular fill bundle. The aspect source flag (ASF) for FILCOLOR, FILSTYLE, and FILTYPE must be set to 'BUNDLED' before the associated GDRAW function is executed if you want the bundled values to be used when the affected graphics element is drawn.

Argument Definitions

index numeric constant or numeric variable name; specifies the index number of the fill bundle. Valid values are 1 to 20, inclusive.

See Also

“FILINDEX” on page 1413

“ASF” on page 1463

“FILREP” on page 1470

FILREP

Associates a bundle of fill attributes with an index

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 75, 78, 85

Default Value: none

Syntax

return-code-variable =GSET('FILREP', *index*, *color-index*, *interior*, *style-index*);

Description

The GSET('FILREP', . . .)function assigns a color, type of interior, and style of the interior to a specific fill bundle. The aspect source flags for FILCOLOR, FILTYPE, and FILSTYLE must be set to 'BUNDLED' before the associated GDRAW function is executed if you want the bundled values to be used when the affected graphics element is drawn.

Argument Definitions

<i>index</i>	numeric constant or numeric variable name; indicates the index to be used with the bundle. Valid values are 1 to 20, inclusive. If <i>index</i> is expressed as a variable, the variable name must be initialized to a value between 1 and 20.
<i>color-index</i>	numeric constant or numeric variable name; indicates the index of the color to be used. Valid values are 1 to 256, inclusive. The color index should represent one of the following: <ul style="list-style-type: none"> □ a color index assigned with the GSET('COLREP', . . .)function □ the <i>n</i>th color in the colors list of the COLORS= graphics option □ the <i>n</i>th color in the device's default colors list.
<i>interior</i>	character string enclosed in quotation marks or character variable name; indicates the type of interior. Valid values are <ul style="list-style-type: none"> □ HATCH □ HOLLOW □ PATTERN □ SOLID.
<i>style-index</i>	numeric constant or numeric variable name; indicates the index of the style to be used. Valid values are 1 to 15, inclusive, when FILTYPE is PATTERN, or 1 to 60, inclusive, when FILTYPE is HATCH. See the GSET('FILSTYLE', . . .)function "FILSTYLE" on page 1471 for a table of the patterns used for each style index. If <i>interior</i> is HOLLOW or SOLID, <i>style-index</i> is ignored.

See Also

"FILREP" on page 1413
 "ASF" on page 1463
 "COLREP" on page 1467
 "FILCOLOR" on page 1469
 "FILINDEX" on page 1470
 "FILSTYLE" on page 1471
 "FILTYPE" on page 1473

FILSTYLE

Specifies the style of the interior of the fill area when the FILTYPE is PATTERN or HATCH

Operating State: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 78

Default Value: 1

Syntax

return-code-variable=GSET('FILSTYLE', *style-index*);

Description

The GSET('FILSTYLE', . . .)function activates a particular fill pattern when FILTYPE is specified as either PATTERN or HATCH. The aspect source flag (ASF) must be set to 'INDIVIDUAL' for this attribute to be used outside of a fill bundle.

Table 50.1 Style Index Table

Value	PATTERN	HATCH	Value	PATTERN	HATCH
1	X1	M1X	31		M3N045
2	X2	M1X030	32		M3N060
3	X3	M1X045	33		M3N090
4	X4	M1X060	34		M3N120
5	X5	M1N	35		M3N135
6	L1	M1N030	36		M3N150
7	L2	M1N045	37		M4X
8	L3	M1N060	38		M4X030
9	L4	M1N090	39		M4X045
10	L5	M1N120	40		M4X060
11	R1	M1N135	41		M4N
12	R2	M1N150	42		M4N030
13	R3	M2X	43		M4N045
14	R4	M2X030	44		M4N060
15	R5	M2X045	45		M4N090
16		M2X060	46		M4N120
17		M2N	47		M4N135
18		M2N030	48		M4N150
19		M2N045	49		M5X
20		M2N060	50		M5X030
21		M2N090	51		M5X045
22		M2N120	52		M5X060
23		M2N135	53		M5N
24		M2N150	54		M5N030
25		M3X	55		M5N045

Value	PATTERN	HATCH	Value	PATTERN	HATCH
26		M3X030	56		M5N060
27		M3X045	57		M5N090
28		M3X060	58		M5N120
29		M3N	59		M5N135
30		M3N030	60		M5N150

Argument Definitions

style-index numeric constant or numeric variable name. Valid values are 1 to 15, inclusive, when FILTYPE is PATTERN, or 1 to 60, inclusive, when FILTYPE is HATCH. See Table 49.1 on page 1360 for value specifications.

See Also

“FILSTYLE” on page 1414

“ASF” on page 1463

“FILREP” on page 1470

“FILTYPE” on page 1473

FILTYPE

Specifies the type of the interior of the fill area

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 78

Default Value: HOLLOW

Syntax

return-code-variable=GSET('FILTYPE', *interior*);

Description

The GSET('FILTYPE', . . .)function selects a particular type of interior fill. If FILTYPE is set to HATCH or PATTERN, the GSET('FILSTYLE', . . .)function determines the type of hatch or pattern fill used. The aspect source flag (ASF) for FILTYPE must be set to 'INDIVIDUAL' for this attribute to be used outside of a fill bundle.

Argument Definitions

interior character string or character variable name; indicates the type of interior fill. Valid values are

- HATCH

- HOLLOW
- PATTERN
- SOLID.

See Also

“ASF” on page 1463

“FILREP” on page 1470

“FILSTYLE” on page 1471

HPOS

Specifies the number of columns

Operating States: GKCL

Return Codes: 0, 1, 90, 307

Default Value: 1. HPOS= graphics option, if specified; 2. device’s default HPOS setting

Syntax

return-code-variable=GSET('HPOS', *hpos*);

Description

The GSET('HPOS', . . .)function sets the number of columns in the graphics output area. GSET('HPOS', . . .)has the same effect as the HPOS= graphics option. See “HPOS” on page 315 for more information. You can reset the HPOS value by submitting one of the following statements:

```
goptions reset=goptions;
goptions reset=all;
```

```
goptions hpos=0;
```

Argument Definitions

hpos numeric constant or numeric variable name; specifies the number of horizontal columns; must be greater than 0.

See Also

“HPOS” on page 1417

“HSIZE” on page 1418

“VPOS” on page 1442

HPOS= graphics option (see “HPOS” on page 315)

HSIZE

Specifies the horizontal dimension of the graphics output area

Operating States: GKCL

Return Codes: 0, 1, 90, 307

Default Value: 1. HSIZE= graphics option, if specified; 2. HSIZE device parameter

Syntax

return-code-variable=GSET('HSIZE', *hsize*);

Description

The GSET('HSIZE', . . .)function sets the horizontal dimension, in inches, of the graphics output area. GSET('HSIZE', . . .)affects the dimensions of the default window. You can reset the HSIZE value by submitting one of the following statements:

```
goptions reset=goptions;
goptions reset=all;

goptions hsize=0;
```

Argument Definitions

hsize numeric constant or numeric variable name; specifies the horizontal dimension, in inches, of the graphics output area; must be greater than 0.

See Also

“HSIZE” on page 1418

“HPOS” on page 1474

“VSIZE” on page 1499

HSIZE= graphics option (see “HSIZE” on page 315)

HTML

Specifies the HTML string to invoke when an affected DSGI graphic element in a web page is clicked

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Default Value: null

Syntax

```
return-code-variable=GSET('HTML', 'string');
```

Description

The GSET('HTML', . . .)function sets the HTML string to invoke when an affected DSGI graphic element in a web page is clicked. The HTML string is used with ODS processing to create a drill-down graph. The string value is used as the value for the HREF= attribute in the image map that implements the drill-down capability.

The value for *string* must be HREF= followed by a valid URL that is specified in double quotation marks, as in

```
rc = GSET('HTML', 'HREF="http://www.sas.com/"');
```

The HTML string can be used by any of the following graphic element types drawn in the code: BAR, ELLIPSE, FILL, MARK, PIE, and TEXT. The string applies to all of these element types that are drawn *after* the string is set. To change the HTML string, set a new value. To turn off the HTML string, specify a null string:

```
rc = GSET('HTML', '');
```

For more information on drill-down graphs, see “Adding Drill-Down Links to Web Presentations” on page 571. For an example of how to use DSGI to generate a drill-down graph, see “Generating a Drill-down Graph Using DSGI” on page 1395.

Argument Definitions

string the HTML string. The string must be enclosed in single quotation marks and must begin with HREF= followed by a URL that is enclosed in double quotation marks.

See Also

- “HTML” on page 1418
- “BAR” on page 1448
- “ELLIPSE” on page 1450
- “FILL” on page 1451
- “MARK” on page 1453
- “PIE” on page 1455
- “TEXT” on page 1456

LINCOLOR

Specifies the color index of the color used to draw lines

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 85

Default Value: 1

Syntax

return-code-variable=GSET('LINCOLOR', *color-index*);

Description

The GSET('LINCOLOR', . . .)function selects the index of the color used to draw lines. The aspect source flag (ASF) for LINCOLOR must be set to 'INDIVIDUAL' for this attribute to be used outside of a line bundle.

DSGI searches for a color specification in the following order:

- 1 the color specified for the index in a GSET('COLREP', . . .)function
- 2 the *n*th color in the colors list of the COLORS= graphics option
- 3 the *n*th color in the device's default colors list found in the device entry.

Argument Definitions

<i>color-index</i>	numeric constant or numeric variable name; indicates the index of the color to use. Valid values are 1 to 256, inclusive.
--------------------	---

See Also

COLORS= graphics option (see "COLORS" on page 272)

"LINCOLOR" on page 1419

"ASF" on page 1463

"COLREP" on page 1467

"LINREP" on page 1478

LININDEX

Specifies the index of the bundle of line attributes

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 60

Default Value: 1

Syntax

return-code-variable=GSET('LININDEX', *index*);

Description

The GSET('LININDEX', . . .)function activates a particular line bundle. The aspect source flags (ASF) of LINCOLOR, LINTYPE, and LINWIDTH must be set to 'BUNDLED' before the associated GDRAW function is executed if you want the bundled values to be used when the affected graphics element is drawn.

Argument Definitions

index numeric constant or numeric variable name; indicates the index of the bundle to activate. Valid values are 1 to 20, inclusive.

See Also

“LININDEX” on page 1420

“ASF” on page 1463

“LINREP” on page 1478

LINREP

Associates a bundle of line attributes with an index

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 60, 62, 85, 90

Default Value: none

Syntax

return-code-variable=GSET ('LINREP',*index*, *color-index*, *width*, *type*);

Description

The GSET('LINREP', . . .)function assigns a color, width, and line type to a specific line bundle. The aspect source flags (ASF) for LINCOLOR, LINWIDTH, and LINTYPE must be set to 'BUNDLED' before the associated GDRAW function is executed if you want the bundled values to be used when the affected graphics element is drawn.

Argument Definitions

index numeric constant or numeric variable name; indicates the number for the bundle to use as an index. Valid values are 1 and 20, inclusive. If *index* is expressed as a variable, the variable must be initialized to a value between 1 and 20.

color-index numeric constant or numeric variable name; specifies the index of the color to use. Valid values are 1 to 256, inclusive. The color index should represent one of the following:

- a color index assigned with the GSET('COLREP', . . .)function
- the *n*th color in the colors list of the COLORS= graphics option
- the *n*th color in the device's default colors list.

width numeric constant or numeric variable name; indicates the width of the line; must be greater than 0.

type numeric constant or numeric variable name; indicates the type of line. Valid values are 1 to 46, inclusive. See Figure 7.22 on page 208 for representations of the different line types.

See Also

“ASF” on page 1463
 “COLREP” on page 1467
 “LINCOLOR” on page 1476
 “LININDEX” on page 1477
 “LINREP” on page 1478
 “LINTYPE” on page 1479
 “LINWIDTH” on page 1479

LINTYPE

Specifies the line type

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 62

Default Value: 1

Syntax

return-code-variable=GSET('LINTYPE', *type*);

Description

The GSET('LINTYPE', . . .)function selects a line type. See Figure 7.22 on page 208 for representations of the different line types. The aspect source flag (ASF) for LINTYPE must be set to 'INDIVIDUAL' for this attribute to be used outside of a line bundle.

Argument Definitions

<i>type</i>	numeric constant or numeric variable name; indicates the type of line to use. Valid values are 1 to 46, inclusive.
-------------	--

See Also

“LINTYPE” on page 1421
 “ASF” on page 1463
 “LINREP” on page 1478

LINWIDTH

Specifies the thickness of the line

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 90

Default Value: 1

Syntax

return-code-variable=GSET('LINWIDTH', *width*);

Description

The GSET('LINWIDTH', . . .)function selects a line width in units of pixels. The aspect source flag (ASF) for LINWIDTH must be set to 'INDIVIDUAL' for this attribute to be used outside of a line bundle.

Argument Definitions

width numeric constant or numeric variable name; specifies the width of the line in pixels; must be greater than 0.

See Also

“LINWIDTH” on page 1422

“ASF” on page 1463

“LINREP” on page 1478

MARCOLOR

Specifies the color index of the color used to draw markers

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 85

Default Value: 1

Syntax

return-code-variable=GSET('MARCOLOR', *color-index*);

Description

The GSET('MARCOLOR', . . .)function selects the color index of the color used to draw markers. The aspect source flag (ASF) of MARCOLOR must be set to 'INDIVIDUAL' for this attribute to be used outside of a marker bundle.

DSGI searches for a color specification in the following order:

- 1 the color specified for the index in a GSET('COLREP', . . .)function
- 2 the *n*th color in the colors list of the COLORS= graphics option

3 the *n*th color in the device's default colors list found in the device entry.

Argument Definitions

color-index numeric constant or numeric variable name; indicates the index of the color to use. Valid values are 1 to 256, inclusive.

See Also

COLORS= graphics option (see "COLORS" on page 272)

"MARCOLOR" on page 1423

"ASF" on page 1463

"COLREP" on page 1467

"MARREP" on page 1482

MARINDEX

Specifies the index of the bundle of marker attributes

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 64

Default Value: 1

Syntax

return-code-variable=GSET('MARINDEX', *index*);

Description

The GSET('MARINDEX', . . .)function activates the marker bundle indicated by *index*. The aspect source flag (ASF) for MARCOLOR, MARTYPE, and MARSIZE must be set to 'BUNDLED' before the GDRAW('MARK', . . .)function is executed if you want the bundled values to be used when the marker is drawn.

Argument Definitions

index numeric constant or numeric variable name; the number of the bundle to activate. Valid values are 1 to 20, inclusive.

See Also

"MARINDEX" on page 1424

"ASF" on page 1463

"MARREP" on page 1482

MARREP

Associates a bundle of marker attributes with an index

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 64, 66, 85, 90

Default Value: none

Syntax

return-code-variable=GSET ('MARREP',*index*, *color-index*, *size*, *type*);

Description

The GSET('MARREP', . . .)function assigns a color, size, and type of marker to a specific marker bundle. The aspect source flag (ASF) of MARCOLOR, MARSIZE, and MARTYPE must be set to 'BUNDLED' before the GDRAW('MARK', . . .)function is executed if you want the bundled values to be used when the marker is drawn.

Argument Definitions

<i>index</i>	numeric constant or numeric variable name; defines the bundle index number. Valid values are 1 to 20, inclusive.
<i>color-index</i>	numeric constant or numeric variable name; indicates the color index of the color to use. Valid values are 1 to 256, inclusive. The color index should represent one of the following: <ul style="list-style-type: none"> □ a color index assigned to a color name with the GSET('COLREP', . . .)function □ the <i>n</i>th color in the colors list of the COLORS= graphics option □ the <i>n</i>th color in the device's default colors list.
<i>size</i>	numeric constant or numeric variable name; indicates the size of the marker in units of the current window system; must be greater than 0.
<i>type</i>	numeric constant or numeric variable name; specifies the type of marker to use; valid values are 1 to 67, inclusive. See Table 50.2 on page 1484 for a table of the symbols used for each marker type.

See Also

- “ASF” on page 1463
- “COLREP” on page 1467
- “MARCOLOR” on page 1480
- “MARINDEX” on page 1481
- “MARSIZE” on page 1483
- “MARTYPE” on page 1483

MARSIZE

Selects the size of markers

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 90

Default Value: 1

Syntax

return-code-variable=GSET('MARSIZE', *size*);

Description

The GSET('MARSIZE', . . .)function sets the marker size in units of the current window system. The aspect source flag (ASF) of MARSIZE must be set to 'INDIVIDUAL' for this attribute to be used outside of a marker bundle.

Argument Definitions

size numeric constant or numeric variable name; indicates the size of the marker in units of the current window system; must be greater than 0.

See Also

“MARSIZE” on page 1425

“ASF” on page 1463

“MARREP” on page 1482

MARTYPE

Selects the kind of markers

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 66

Default Value: 1

Syntax

return-code-variable=GSET('MARTYPE', *type*);

Description

The GSET('MARTYPE', . . .)function determines the type of marker drawn. See Figure 7.21 on page 202 for representations of the symbols described in Table 50.2 on page

1484. The aspect source flag (ASF) of MARTYPE must be set to 'INDIVIDUAL' for this attribute to be used outside of a marker bundle.

Table 50.2 Symbol Indexes Used with DSGI

<i>Values and Markers</i>					
1	plus	24	K	46	9
2	x	25	L	47	lozenge
3	star	26	M	48	spade
4	square	27	N	49	heart
5	diamond	28	O	50	diamond
6	triangle	29	P	51	club
7	hash	30	Q	52	shamrock
8	Y	31	R	53	fleur-de-lis
9	Z	32	S	54	star
10	paw	33	T	55	sun
11	point	34	U	56	Mercury
12	dot	35	V	57	Venus
13	circle	36	W	58	Earth
14	A	37	0	59	Mars
15	B	38	1	60	Jupiter
16	C	39	2	61	Saturn
17	D	40	3	62	Uranus
18	E	41	4	63	Neptune
19	F	42	5	64	Pluto
20	G	43	6	65	moon
21	H	44	7	66	comet
22	I	45	8	67	asterisk
23	J				

Argument Definitions

type numeric constant or numeric variable name; indicates the index of the marker to draw. Valid values are 1 to 67, inclusive. See Table 50.2 on page 1484 for value specifications.

See Also

“MARTYPE” on page 1426

“ASF” on page 1463

“MARREP” on page 1482

MESSAGE

Specifies whether the interface error message system is enabled or disabled

Operating States: All

Return Codes: 0

Default Value: ON

Syntax

return-code-variable=GSET('MESSAGE', *status*);

Description

The GSET('MESSAGE', . . .)function activates or suppresses automatic error logging.

Argument Definitions

status character string enclosed in quotation marks or character variable name; indicates whether messages should be displayed. Valid values are ON and OFF. When ON is used, messages are automatically generated by the DSGI based on the return code from the function. If you set MESSAGE to OFF, no messages are automatically printed. You may choose to do this if you want to print custom messages for your application or decide which error message you want printed.

See Also

“MESSAGE” on page 1454

“GPRINT” on page 1402

PATREP

Specifies the pattern name of a style index for a particular fill type.

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 79

Default value: 1

Syntax

```
return-code-variable=CALL GSET('PATREP', index, pattern-name, hatch-name);
```

Description

The GSET('PATREP', . . .)function sets a pattern of a style index for a particular fill type.

Argument Definitions

<i>index</i>	numeric variable name; indicates the index of the pattern to be used.
<i>pattern-name</i>	character variable name; sets the name of the pattern at the specified index.
<i>hatch-name</i>	character variable name; sets the name of the hatch at the specified index.

See Also

“PATREP” on page 1429

TEXALIGN

Specifies the horizontal and vertical alignment of the text string

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Default values: *halign*=NORMAL, *valign*=NORMAL

Syntax

```
return-code-variable=GSET('TEXALIGN', halign, valign);
```

Description

The GSET('TEXALIGN', . . .)function sets a particular type of horizontal and vertical alignment for text strings. Figure 50.5 on page 1487 illustrates *halign*.

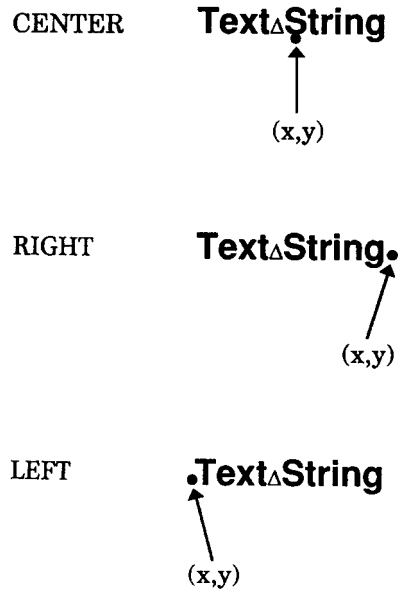
Figure 50.5 Halign Values

Figure 50.6 on page 1487 illustrates *valign*.

Figure 50.6 Valign Values

Argument Definitions

halign character string enclosed in quotation marks or character variable name. Valid values are

CENTER

LEFT

NORMAL (the natural alignment based on the text path); alignment is chosen according to the following logic:

- 1 If `TEXPATH` is 'RIGHT', then NORMAL is 'LEFT'.
- 2 Otherwise, if `TEXPATH` is 'LEFT', then NORMAL is 'RIGHT'.
- 3 Otherwise, the text string is centered.

RIGHT.

valign character string enclosed in quotation marks or character variable name. Valid values are

BASE (alignment based on the baseline of the text string)

BOTTOM (alignment based on the bottom of the text string)

HALF (alignment based on the vertical midpoint of the string)

NORMAL (natural alignment based on the text path); alignment is chosen according to the following logic:

- 1 If `TEXPATH` is 'RIGHT' or `TEXPATH` is 'LEFT', then `NORMAL` is 'BASE'.
- 2 Otherwise, if `TEXPATH` is 'UP', then `NORMAL` is 'BOTTOM'.
- 3 Otherwise, if `TEXPATH` is 'DOWN', then `NORMAL` is 'TOP'.

TOP (alignment based on the top of the string).

See Also

“`TEXALIGN`” on page 1430

“`TEXT`” on page 1456

“`TEXPATH`” on page 1491

“`TEXUP`” on page 1493

TEXCOLOR

Specifies the color index of the color used to draw text strings

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 85

Default Value: 1

Syntax

return-code-variable=GSET("TEXCOLOR", *color-index*);

Description

The GSET("TEXCOLOR", . . .)function selects the color for text. The aspect source flag (ASF) of `TEXCOLOR` must be set to 'INDIVIDUAL' for this attribute to be used outside of a text bundle.

The value of GSET("TEXCOLOR", . . .)can be used in a text bundle. See the “`TEXREP`” on page 1492 for information on how to define a text bundle.

DSGI searches for a color specification in the following order:

- 1 the color specified for the index in a GSET('COLREP', . . .)function
- 2 the *n*th color from the colors list of the `COLORS=` graphics option
- 3 the *n*th color in the device's default colors list found in the device entry.

Argument Definitions

color-index numeric constant or numeric variable name; indicates the color index of the color to be used. Valid values are 1 to 256, inclusive.

See Also

COLORS= graphics option (see “COLORS” on page 272)

“TEXCOLOR” on page 1431

“ASF” on page 1463

“COLREP” on page 1467

“TEXREP” on page 1492

TEXTFONT

Specifies the font used to draw text strings

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Default values: 1. FTEXT= graphics option, if specified; 2. hardware font, if possible; 3. SIMULATE font

Syntax

return-code-variable=GSET(‘TEXTFONT’, *font*);

Description

The GSET(‘TEXTFONT’, . . .)function selects a SAS/GRAPH font for the text. The aspect source flag (ASF) of TEXTFONT must be set to ‘INDIVIDUAL’ for this attribute to be used outside of a text bundle. See “Font Lists” on page 82 for a list of valid SAS/GRAPH fonts. You may also use fonts you have created using the GFONT procedure.

Argument Definitions

font character string enclosed in quotation marks or character variable name; the name of a font that can be accessed by SAS/GRAPH software. If you want to use the hardware font, submit

```
rc=gset('textfont', ' ');
```

When DSGI is used with long font names, the font name must be in double quotation marks that are embedded in single quotation marks, as in “HW font name”.

See Also

FTEXT= graphics options (see “FTEXT” on page 294)

“TEXFONT” on page 1434

“ASF” on page 1463

“TEXREP” on page 1492

TEXHEIGHT

Specifies the character height of the text string

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 73

Default Value: 1. HTEXT= graphics option, if specified; 2. 1 unit

Syntax

return-code-variable=GSET("TEXHEIGHT", *height*);

Description

The GSET("TEXHEIGHT", . . .)function sets the height for text. GSET("TEXHEIGHT", . . .)affects text the same way as the HTEXT= graphics option.

Argument Definitions

height numeric constant or numeric variable name; indicates height in units based on the current window system; must be greater than 0.

See Also

“TEXHEIGHT” on page 1435

HTEXT= graphics options (see “HTEXT” on page 316)

TEXINDEX

Specifies the index of the bundle of text attributes

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 68

Default Value: 1

Syntax

return-code-variable=GSET('TEXINDEX', *index*);

Description

The GSET('TEXINDEX', . . .)function activates the text bundle indicated by *index*. The aspect source flag (ASF) for TEXCOLOR and TEXTFONT must be set to 'BUNDLED' before the GDRAW('TEXT', . . .)function is executed if you want the bundled values to be used when the text is drawn.

Argument Definitions

index numeric constant or numeric variable name; indicates the number of the bundle to activate. Valid values are 1 to 20, inclusive.

See Also

“TEXINDEX” on page 1436

“ASF” on page 1463

“TEXREP” on page 1492

TEXPATH

Specifies the direction of the text string

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8

Default Value: RIGHT

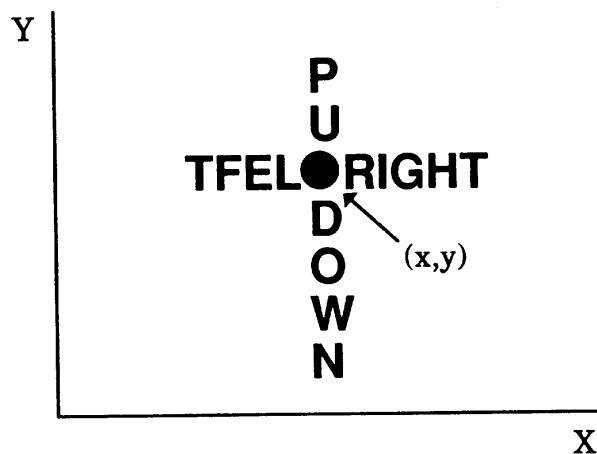
Syntax

return-code-variable=GSET('TEXPATH', *path*);

Description

The GSET('TEXPATH', . . .)function selects a particular type of text path. Text path determines the direction in which the text string reads. Figure 50.7 on page 1492 illustrates the text paths that can be used with DSGI.

Figure 50.7 TEXPATH Values



Argument Definitions

<i>path</i>	character string enclosed in quotation marks or character variable name; specifies the direction in which the text will read. Valid values are
	<input type="checkbox"/> DOWN
	<input type="checkbox"/> LEFT
	<input type="checkbox"/> RIGHT
	<input type="checkbox"/> UP.

See Also

“TEXPATH” on page 1436

“TEXT” on page 1456

“TEXALIGN” on page 1486

“TEXUP” on page 1493

TEXREP

Associates a bundle of text attributes with an index

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 68, 85

Default Value: none

Syntax

return-code-variable=GSET (“TEXREP”,*index*, *color-index*, *font*);

Description

The GSET('TEXREP', . . .)function assigns a color and font to a particular text bundle. The aspect source flags (ASF) of TEXCOLOR and TEXTFONT must be set to 'BUNDLED' before the GDRAW('TEXT', . . .)function is executed if you want the bundled values to be used when the text is drawn.

Argument Definitions

<i>index</i>	numeric constant or numeric variable name; specifies the number to use as an index for the bundle; valid values are 1 to 20, inclusive. If <i>index</i> is expressed as a variable, the variable must be initialized to a value between 1 and 20.
<i>color-index</i>	numeric constant or numeric variable name; indicates the color to use; valid values are 1 to 256, inclusive. The color index should represent one of the following: <ul style="list-style-type: none"> □ a color index assigned with the GSET('COLREP', . . .)function □ the <i>n</i>th color in the colors list of the COLORS= graphics option □ the <i>n</i>th color in the device's default colors list.
<i>font</i>	character string enclosed in quotation marks or character variable name; names the font to use with the bundle. See "Font Lists" on page 82 for a list of valid SAS/GRAPH fonts. You may also use fonts you have created using the GFONT procedure.

See Also

COLORS= graphics option (see "COLORS" on page 272)
 "TEXREP" on page 1437
 "ASF" on page 1463
 "COLREP" on page 1467
 "TEXINDEX" on page 1490

TEXUP

Specifies the orientation (angle) of the text string

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 74

Default Values: *upx*=0, *upy*=1

Syntax

return-code-variable=GSET('TEXUP',*upx*, *upy*);

Description

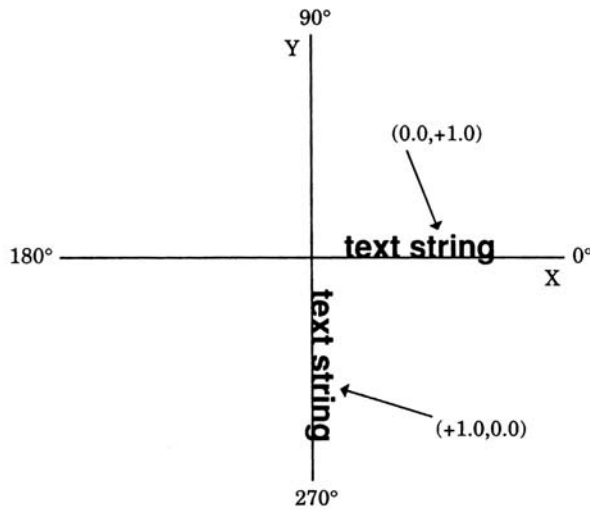
The GSET('TEXUP', . . .)function sets the angle of the text string. DSGI uses the values of character up vectors to determine the angle of a text string. The character up

vector has two components, upx and upy , that describe the angle at which the text string is placed. The angle is calculated with the following formula:

$$\text{angle} = \text{atan}(upx/upy)$$

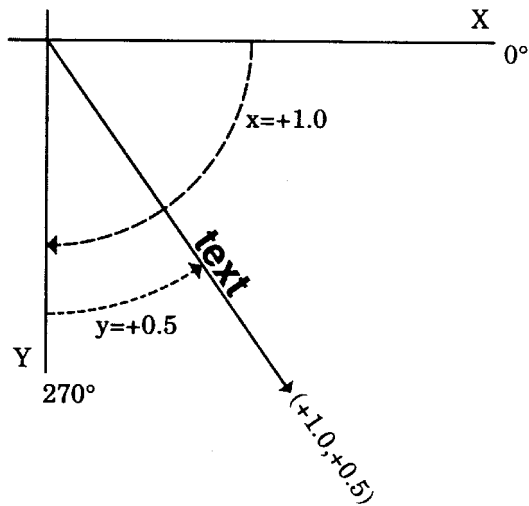
Effectively, when DSGI is calculating the angle for the text, it uses upx and upy as forces that are pushing the string toward an angle. The natural angle of text in the upx direction is toward the 6 o'clock position. In the upy direction, text naturally angles at the 3 o'clock position. If upx is greater than upy , the text is angled toward 6 o'clock. If upy is greater than upx , the text is angled toward 3 o'clock. Figure 50.8 on page 1494 shows the angle of text when the values for upx and upy are (0.0, 1.0) and (1.0, 0.0).

Figure 50.8 Natural Angle of Text



As you change the values of upx and upy , the coordinate that has the highest value is taken as the angle, and the lowest value as the offset. Figure 50.9 on page 1495 shows the angle of text when the character vector values (+1.0, +0.5) are used.

Figure 50.9 Varying the Angle of Text



You can use the following macro to convert angles measured in degrees to character up vectors:

```
%macro angle(x);
  if mod(&x, 180)=90 then do;
    if mod(&x,270) = 0 then
      xup = 1.0;
    else
      xup = -1.0;
    rc = gset('texup', xup, 0.0);
    end;
  else do;
    b = mod(&x, 360);
    /* adjust y vector for 2nd and 3rd quadrants */
    if b > 90 and b lt 270 then
      yup = -1.0;
    else
      yup = 1.0;
    a=&x*1.7453292519943300e-002;
    xup = tan(-a);
    /* adjust x vector for 3rd quadrant */
    if b > 180 and b le 270 then
      xup = -xup;
    rc = gset('texup', xup, yup);
    end;
%mend angle;

data _null_;
  rc = ginit();
  rc = graph('clear', 'angle');
  rc = gset('texalign', 'left', 'base');
  rc = gset('texheight', 5);
  rc = gset('texfont', 'swissl');
  %angle(180);
  rc = gdraw('text', 50, 50, '180');
  %angle(80);
```

```
rc = gdraw('text', 50, 50, '80');
%angle(600);
rc = gdraw('text', 50, 50, '600');
rc = graph('update');
rc = gterm();
run;
```

Argument Definitions

upx numeric constant or numeric variable name; if *upy* is 0, *upx* cannot be 0.

upy numeric constant or numeric variable name; if *upx* is 0, *upy* cannot be 0.

See Also

“TEXUP” on page 1438
 “TEXT” on page 1456
 “TEXALIGN” on page 1486
 “TEXPATH” on page 1491

TRANSNO

Specifies the number of the transformation to be used

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 50

Default Value: 0

Syntax

return-code-variable=GSET("TRANSNO", *n*);

Description

The GSET("TRANSNO", . . .)function activates the viewport and/or window you have defined for the specified transformation number. If you have not defined both a viewport and window for a transformation, the default is used for the one missing.

You can select 0 as the active transformation, but you cannot define a viewport or window for that transformation number. A transformation of 0 activates the default viewport, (0,0) to (1,1), and window, which is device dependent.

Argument Definitions

n numeric constant or numeric variable name; indicates the viewport and/or window to activate; should correspond to the *n* used in the

GSET('VIEWPORT', . . .)and/or GSET('WINDOW', . . .)functions.
Valid values are 0 to 20, inclusive.

See Also

“TRANS” on page 1439

“TRANSNO” on page 1440

“VIEWPORT” on page 1441

“WINDOW” on page 1444

“VIEWPORT” on page 1497

“WINDOW” on page 1500

VIEWPORT

Associates a viewport with a transformation number

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 50, 51, 52

Default Values: *llx=0, lly=0, urx=1, ury=1*

Syntax

return-code-variable=GSET('VIEWPORT', *n*, *llx*, *lly*, *urx*, *ury*);

Description

The GSET('VIEWPORT', . . .)function defines a viewport and associates it with the transformation number, *n*. See the “TRANSNO” on page 1496 for information on how to activate the viewport. See the “WINDOW” on page 1500 for information on how to define a window to be used within the viewport.

Argument Definitions

<i>n</i>	numeric constant or numeric variable name; specifies the transformation number of the viewport. Valid values are 1 to 20, inclusive.
<i>llx</i>	numeric constant or numeric variable name; defines the <i>x</i> component of the lower-left corner of the viewport; must not exceed the value of <i>urx</i> ; cannot be less than 0. Units are based on percent of the graphics output area.
<i>lly</i>	numeric constant or numeric variable name; defines the <i>y</i> component of the lower-left corner of the viewport; must not exceed the value of <i>ury</i> ; cannot be less than 0. Units are based on percent of the graphics output area.

<i>urx</i>	numeric constant or numeric variable name; defines the <i>x</i> component of the upper-right corner of the viewport; cannot be greater than 1. Units are based on percent of the graphics output area.
<i>ury</i>	numeric constant or numeric variable name; defines the <i>y</i> component of the upper-right corner of the viewport; cannot be greater than 1. Units are based on percent of the graphics output area.

See Also

“VIEWPORT” on page 1441

“WINDOW” on page 1500

“TRANSNO” on page 1496

“TRANSNO” on page 1440

“TRANS” on page 1439

“WINDOW” on page 1444

VPOS

Specifies the number of rows

Operating States: GKCL

Return Codes: 0, 1, 90, 307

Default Values: 1. VPOS=graphics option, if specified; 2. device’s default VPOS value

Syntax

```
return-code-variable=GSET('VPOS', vpos);
```

Description

The GSET('VPOS', . . .)function sets the number of rows in the graphics output area. GSET('VPOS', . . .)has the same effect on graphics output as the VPOS= graphics option.

You can reset the VPOS value by submitting one of the following statements:

```
goptions reset=goptions;
goptions reset=all;
```

```
goptions vpos=0;
```

Argument Definitions

vpos numeric constant or numeric variable name; specifies the number of rows in the graphics output area; must be greater than 0.

See Also

“VPOS” on page 1442

“HPOS” on page 1474

“VSIZE” on page 1499

VPOS= graphics option (see “VPOS” on page 360)

VSIZE

Specifies the vertical dimension of the graphics output area

Operating States: GKCL

Return Codes: 0, 1, 90, 307

Default Values: 1. VSIZE= graphics option, if specified; 2. device’s default VSIZE value

Syntax

return-code-variable=GSET('VSIZE', *vsiz*);

Description

The GSET('VSIZE', . . .)function sets the vertical dimension, in inches, of the graphics output area. GSET('VSIZE', . . .)affects the dimensions of the default window.

You can reset the VSIZE value by submitting one of the following statements:

```
goptions reset=goptions;
goptions reset=all;
goptions vsiz=0;
```

Argument Definitions

vsiz numeric constant or numeric variable name; indicates the vertical dimension for the graph in inches; must be greater than 0.

See Also

“VSIZE” on page 1443

“HSIZE” on page 1475

“VPOS” on page 1498

VSIZE= graphics option (see “VSIZE” on page 361)

WINDOW

Associates a window with a transformation number

Operating States: GKOP, SGOP, WSAC, WSOP

Return Codes: 0, 8, 50, 51

Default Values: *llx=0, lly=0*; *urx* and *ury* are device dependent

Syntax

return-code-variable=GSET ('WINDOW', *n*, *llx*, *lly*, *urx*, *ury*);

Description

The GSET('WINDOW', . . .)function defines a window and associates it with a transformation number. See the "TRANSNO" on page 1496 for information on how to activate a window. See the "VIEWPORT" on page 1497 for information on how to define a viewport for a window.

Argument Definitions

<i>n</i>	numeric constant or numeric variable name; specifies the transformation number of the window. Valid values are 1 to 20, inclusive.
<i>llx</i>	numeric constant or numeric variable name; defines the <i>x</i> component of the lower-left corner of the window; must not exceed the value of <i>urx</i> . Units are based on percent of the active viewport.
<i>lly</i>	numeric constant or numeric variable name; defines the <i>y</i> component of the lower-left corner of the window; must not exceed the value of <i>ury</i> . Units are based on percent of the active viewport.
<i>urx</i>	numeric constant or numeric variable name; defines the <i>x</i> component of the upper-right corner of the window. Units are based on percent of the active viewport.
<i>ury</i>	numeric constant or numeric variable name; defines the <i>y</i> component of the upper-right corner of the window. Units are based on percent of the active viewport.

See Also

"TRANS" on page 1439

"TRANSNO" on page 1440

"VIEWPORT" on page 1441

"WINDOW" on page 1444

"TRANSNO" on page 1496

"VIEWPORT" on page 1497

Return Codes for DSGI Routines and Functions

0	Function completed successfully.
1	DATA Step Graphics Interface should be in GKCL state; the statement is out of place within the DATA step.
3	DATA Step Graphics Interface should be in WSAC state; the statement is out of place within the DATA step.
4	DATA Step Graphics Interface should be in SGOP state; the statement is out of place within the DATA step.
7	DATA Step Graphics Interface should be in WSOP, WSAC, or SGOP state; the statement is out of place within the DATA step.
8	DATA Step Graphics Interface should be in GKOP, WSOP, WSAC, or SGOP state; the statement is out of place within the DATA step.
24	Workstation is open.
25	Workstation is not open.
26	Workstation cannot be opened.
29	Workstation is active.
30	Workstation is not active.
50	Invalid transformation number; transformation numbers must be in the range 0 to 20; viewports and windows cannot be defined for transformation 0.
51	Transformation is not a well-defined rectangle; transformations must have coordinates for four vertices.
52	Viewport coordinates are out of range; coordinates must be within dimensions of graphics output area for the device.
55	Clipping is on.
56	Clipping is off.
60	Bad line index; index numbers must be in the range 1 to 20.
61	No bundle defined for the line index; a GSET('LINREP', . . .)function has not been submitted for the referenced line index.
62	Line type is less than or equal to 0 or greater than 46; type must be in the range 1 to 46.
64	Invalid marker index; index numbers must be in the range 1 to 20.
65	No bundle defined for the polymarker index; a GSET('MARREP', . . .)function has not been submitted for the referenced marker index.
66	Marker type is less than or equal to 0 or greater than 67; type must be in the range 1 to 67.
68	Invalid text index; index numbers must be in the range 1 to 20.
69	No bundle defined for the text index; a GSET('TEXREP', . . .)function has not been submitted for the referenced text index.

73	Character height is less than or equal to 0; height must be greater than 0.
74	Both components of the character up vector are 0; both X and Y of a character up vector cannot be 0.
75	Invalid fill index; index numbers must be in the range 1 to 20.
76	No bundle defined for the fill index; a GSET('FILREP', . . .)function has not been submitted for the referenced fill index.
78	Style index is less than or equal to 0 or greater than 60; style indexes must be in the range of 1 to 60.
79	Invalid pattern index.
86	Invalid color index; color index is out of the range 1 to 256 or is not numeric.
87	No color name defined for the color index
90	Value is less than 0; value must be greater than or equal to 0.
150	External image file cannot be accessed. The image file either cannot be accessed, or the image file is in an unsupported format, or the image data is incomplete or otherwise corrupt.
301	Out of memory; your workstation does not have enough memory to generate the graph.
302	Out of room for graph; your device cannot display the size of the graph.
307	Error occurred in program library management; a GRAPH function did not execute properly.

See Also

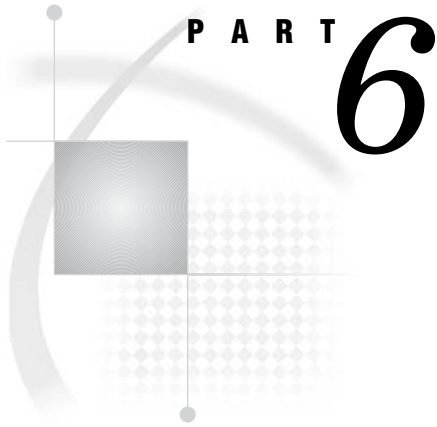
- Chapter 3, "Device Drivers," on page 41
for information about specifying device drivers.
- Chapter 8, "Graphics Options and Device Parameters Dictionary," on page 261
for descriptions of graphics options and device parameters
- Chapter 5, "SAS/GRAPH Fonts," on page 75
for information about the fonts available in SAS/GRAPH software
- Chapter 6, "SAS/GRAPH Colors and Images," on page 91
for information about specifying colors in SAS/GRAPH programs
- "GOPTIONS Statement" on page 146
for an explanation of setting graphics options with the GOPTIONS statement
- "PATTERN Statement" on page 169
for information about specifying patterns with DSGI
- "SYMBOL Statement" on page 183
for representations of the markers that can be used with DSGI
- Chapter 49, "The DATA Step Graphics Interface," on page 1353
for a complete explanation of using DSGI statements to produce graphs

Chapter 31, “The GDEVICE Procedure,” on page 915
for information about device entries

The discussion for ARRAY in *SAS Language Reference: Dictionary*
for an explanation of OF argument lists

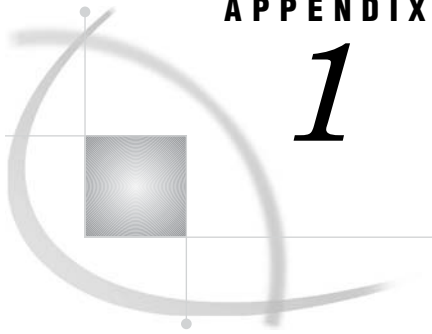
References

Enderle, G.; Kansy, K.; and Pfaff, G. (1985), *Computer Graphics Programming: GKS—The Graphics Standard* Springer-Verlag New York, Inc.



Appendixes

- Appendix 1* **Summary of ActiveX and Java Support** 1507
- Appendix 2* **Recommended Reading** 1547



APPENDIX

1

Summary of ActiveX and Java Support

<i>Introduction</i>	1508
<i>Global Statements</i>	1508
<i>AXIS Statement</i>	1508
<i>Text Description Suboptions</i>	1509
<i>Tick Mark Description Suboptions</i>	1509
<i>GOPTIONS Statement</i>	1510
<i>LEGEND Statement</i>	1514
<i>LEGEND Statement Text Description Suboptions</i>	1515
<i>PATTERN Statement</i>	1515
<i>SYMBOL Statement</i>	1516
<i>POINTLABEL= Label Description Options</i>	1517
<i>TITLE and FOOTNOTE Statements</i>	1517
<i>PROC GAREABAR</i>	1518
<i>PROC GBARLINE</i>	1519
<i>PROC GCHART</i>	1521
<i>Text Description Suboptions</i>	1525
<i>PROC GCONTOUR</i>	1526
<i>PROC GMAP</i>	1527
<i>PROC GPLOT</i>	1530
<i>PROC GRADAR</i>	1535
<i>PROC G3D</i>	1537
<i>Annotate Functions</i>	1539
<i>BAR</i>	1539
<i>DRAW</i>	1539
<i>DRAW2TXT</i>	1540
<i>FRAME</i>	1540
<i>IMAGE</i>	1540
<i>LABEL</i>	1541
<i>MOVE</i>	1541
<i>PIE</i>	1542
<i>PIECNTR</i>	1542
<i>PIEXY</i>	1543
<i>POINT</i>	1543
<i>POLY</i>	1543
<i>POLYCONT</i>	1544
<i>SYMBOL</i>	1544

Introduction

The following tables summarize which options and annotate variables are supported or partially supported by Java and ActiveX. Partial support for options that refer to global statements, such as the GAXIS= option, indicates that some but not all AXIS statement options are supported. Partial support may also indicate that an option works differently for the server than it does for the Java and ActiveX device drivers, or that an option works for one or more applets but not for all. For a complete description of each option or variable, refer to the documentation for the option or variable.

Global Statements

AXIS Statement

Table A1.1 ActiveX and Java Support for the AXIS Statement

Option	Supported by ActiveX?	Supported by Java?
COLOR=	Yes	Yes
C=		
INTERVAL=	Yes	Yes
LABEL=	Yes (partial)	Yes (partial)
LENGTH=	No	No
LOGBASE=	Yes	No
LOGSTYLE=	Yes	No
MAJOR=	Yes	Yes
MINOR=	Yes	Yes
NOBRACKETS	No	No
NOPLANE	Yes	Yes
OFFSET=	Yes	No
ORDER=	Yes (partial)	Yes (partial)
ORIGIN=	No	No
REFLABEL=	No	No
SPLIT=	No	No
STYLE=	Yes	Yes

Option	Supported by ActiveX?	Supported by Java?
VALUE=	Yes (partial)	Yes (partial)
WIDTH=	Yes (partial)	No

Text Description Suboptions

Text description suboptions are used by the LABEL=, REFLABEL=, and VALUE= options.

Table A1.2 ActiveX and Java Support for AXIS Text Description Suboptions

Option	Supported by ActiveX?	Supported by Java?
ANGLE= A=	Yes	Yes (partial)
AUTOREF	Yes	No
COLOR= C=	Yes	Yes
FONT= F=	Yes	Yes (partial)
HEIGHT= H=	Yes	Yes
JUSTIFY= J=	Yes	No
POSITION=	No	No
ROTATE= R=	Yes	Yes (partial)
TICK= T=	No	No

Tick Mark Description Suboptions

Tick mark description suboptions are used by the MAJOR= and MINOR= options to change the color, height, width, and number of the tick marks to which they apply.

Table A1.3 ActiveX and Java Support for Tick Mark Description Suboptions

Option	Supported by ActiveX?	Supported by Java?
COLOR= C=	Yes	Yes
HEIGHT= H=	No	No

Option	Supported by ActiveX?	Supported by Java?
NUMBER= N=	Yes	Yes
WIDTH= W=	Yes	Yes (partial)

GOPTIONS Statement

You must specify the ODS USEGOPT statement for the CTEXT=, CTITLE=, FTEXT=, FTITLE=, HTEXT=, and HTITLE= options to work for the Java and ActiveX devices. See “Controlling the Text Font, Size, and Color” on page 493 for more information.

Table A1.4 ActiveX and Java Support for the GOPTIONS Statement

Option	Supported by ActiveX?	Supported by Java?
ADMGDF NOADMGDF	No	No
ASPECT=	No	No
AUTOCOPY NOAUTOCOPY	No	No
AUTOFEED NOAUTOFEED	No	No
AUTOSIZE=	No	No
BINDING=	No	No
BORDER	Yes	Yes
CBACK=	Yes	Yes
CBY=	No	No
CELL	No	No
CHARACTERS NOCHARACTERS	No	No
CHARTYPE=	No	No
CIRCLEARC NOCIRCLEARC	No	No
COLLATE NOCOLLATE	No	No
COLORS=	Yes	Yes
CPATTERN=	No	No
CSYMBOL=	No	No
CTEXT=	Yes	Yes (partial)
CTITLE=	Yes	Yes
DASH NODASH	No	No
DASHSCALE=	No	No

Option	Supported by ActiveX?	Supported by Java?
DELAY=	No	No
DEVADDR=	No	No
DEVICE=	Yes	Yes
DEVMAP=	No	No
DISPLAY NODISPLAY	No	No
DISPOSAL=	No	No
DRVINIT=	No	No
DRVTERM=	No	No
DUPLEX NODUPLEX	No	No
ERASE NOERASE	No	No
EXTENSION	No	No
FASTTEXT NOFASTTEXT	No	No
FBY=	No	No
FCACHE=	No	No
FILECLOSE=	No	No
FILEONLY NOFILEONLY	No	No
FILL NOFILL	No	No
FILLINC=	No	No
FONTRES=	No	No
FTEXT=	Yes (partial)	Yes (partial)
FTITLE=	Yes	Yes
FTRACK=	No	No
GACCESS=	No	No
GCLASS=	No	No
GCOPIES=	No	No
GDDMCOPY=	No	No
GDDMNICKNAME=	No	No
GDDMTOKEN=	No	No
GDEST=	No	No
GEND=	No	No
GPILOG=	No	No
GFORMS=	No	No
GOUTMODE=	No	No

Option	Supported by ActiveX?	Supported by Java?
GPROLOG=	No	No
GPROTOCOL=	No	No
GRAPHRC NOGRAPHRC	No	No
GSFLEN=	No	No
GSFMODE=	No	No
GSFNAME=	No	No
GSFPROMPT NOGSFPROMPT	No	No
GSIZE=	No	No
GSTART=	No	No
GUNIT=	Yes (partial)	Yes (partial)
GWAIT=	No	No
GWRITER=	No	No
HANDSHAKE=	No	No
HBY=	No	No
HORIGIN=	No	No
HPOS=	No	No
HSIZE=	Yes (partial)	Yes (partial)
HTEXT=	Yes	Yes (partial)
HTITLE=	Yes	Yes
IBACK=	Yes	Yes (partial)
IMAGEPRINT NOIMAGEPRINT	No	No
IMAGESTYLE=	Yes	No
INTERLACED NOINTERLACED	No	No
INTERPOL=	No	No
ITERATION=	No	No
KEYMAP=	No	No
LFACTOR=	No	No
OFFSHADOW=	No	No
PAPERDEST=	No	No
PAPERFEED=	No	No
PAPERLIMIT=	No	No
PAPERSIZE=	No	No
PAPERSOURCE=	No	No
PAPERTYPE=	No	No

Option	Supported by ActiveX?	Supported by Java?
PCLIP NOPCLIP	No	No
PENMOUNTS=	No	No
PENSORT NOPENSORT	No	No
PIEFILL NOPIEFILL	No	No
POLYGONCLIP NOPOLYGONCLIP	No	No
POLYGONFILL NOPOLYGONFILL	No	No
POSTGEPILOG=	No	No
POSTGPROLOG=	No	No
POSTGRAPH=	No	No
PPDFILE=	No	No
PREGEPILOG=	No	No
PREGPROLOG=	No	No
PREGRAPH=	No	No
PROMPT NOPROMPT	No	No
PROMPTCHARS=	No	No
RENDER=	No	No
RENDERLIB=	No	No
REPAINT=	No	No
RESET	Yes	Yes
REVERSE NOREVERSE	No	No
ROTATE=	No	No
ROTATE NOROTATE	No	No
SIMFONT=	No	No
SPEED=	No	No
SWAP NOSWAP	No	No
SWFONTRENDER NOSWFONTRENDER	No	No
SYMBOL NOSYMBOL	No	No
TARGETDEVICE=	No	No
TRANSPARENCY NOTRANSPARENCY	No	No

Option	Supported by ActiveX?	Supported by Java?
TRANTAB=	No	No
UCC=	No	No
USERINPUT NOUSERINPUT	No	No
VORIGIN=	No	No
VPOS=	No	No
VSIZE=	Yes (partial)	Yes (partial)
V6COMP NOV6COMP	Yes (partial)	Yes (partial)
XMAX=	No	No
XPIXELS=	Yes (partial)	Yes (partial)
YMAX=	No	No
YPIXELS=	Yes (partial)	Yes (partial)

LEGEND Statement

Table A1.5 ActiveX and Java Support for the LEGEND Statement

Option	Supported by ActiveX?	Supported by Java?
ACROSS=	Yes	Yes
CBLOCK=	Yes	No
CBORDER=	Yes	Yes
CFRAME=	Yes	Yes
CSHADOW=	Yes	Yes
DOWN=	Yes	Yes
FRAME	Yes	Yes
FWIDTH=	No	No
LABEL=	Yes (partial)	Yes (partial)
MODE=	No	No
OFFSET=	No	No
ORDER=	No	No
ORIGIN=	No	No
POSITION=	Yes	Yes

Option	Supported by ActiveX?	Supported by Java?
SHAPE=	No	No
VALUE=	Yes (partial)	Yes (partial)

LEGEND Statement Text Description Suboptions

Text description suboptions are used by the LABEL= and VALUE= options to change the color, height, justification, font, and angle of either default text or specified text strings. See LABEL= and VALUE=.

Table A1.6 ActiveX and Java Support for LEGEND Text Description Suboptions

Option	Supported by ActiveX?	Supported by Java?
COLOR= C=	Yes	Yes
FONT= F=	Yes	Yes
HEIGHT= H=	Yes	Yes
JUSTIFY= J=	Yes	Yes
POSITION=	Yes	No
ROTATE=	No	No
TICK= T=	Yes	Yes

PATTERN Statement

Table A1.7 ActiveX and Java Support for the PATTERN Statement

Option	Supported by ActiveX?	Supported by Java?
COLOR= C=	Yes (partial)	Yes (partial)
REPEAT= R=	Yes (partial)	Yes (partial)
IMAGE=	Yes (partial)	Yes (partial)
IMAGESTYLE=	Yes (partial)	Yes (partial)
VALUE= <i>bar / block-pattern</i> V= <i>bar / block-pattern</i>	Yes (partial)	Yes (partial)
VALUE= <i>map / plot-pattern</i> V= <i>map / plot-pattern</i>	Yes (partial)	Yes (partial)

Option	Supported by ActiveX?	Supported by Java?
VALUE= <i>pie / star-pattern</i> V= <i>pie / star-pattern</i>	Yes (partial)	Yes (partial)
VALUE=HWxxxnnn	Yes (partial)	Yes (partial)

SYMBOL Statement

Table A1.8 ActiveX and Java Support for the SYMBOL Statement

Option	Supported by ActiveX?	Supported by Java?
BWIDTH=	Yes	Yes
CI=	Yes	Yes
CO=	Yes	Yes
COLOR=	Yes (GPLOT)	Yes (GPLOT)
C=	No (GCONTOUR)	No (GCONTOUR)
CV=	Yes (GPLOT) No (GCONTOUR)	Yes (GPLOT) No (GCONTOUR)
FONT=	No	No
HEIGHT=	Yes (GPLOT)	Yes (GPLOT)
H=	No (GCONTOUR)	No (GCONTOUR)
INTERPOL=BOX I=BOX	Yes	Yes (partial)
INTERPOL=HILO I=HILO	Yes	Yes (partial)
INTERPOL=JOIN I=JOIN	Yes	Yes
INTERPOL=L I=L	Yes	Yes
INTERPOL= <i>map / plot-pattern</i> I= <i>map / plot-pattern</i>	Yes	Yes (partial)
INTERPOL=NEEDLE I=NEEDLE	Yes	Yes
INTERPOL=NONE I=NONE	Yes	Yes
INTERPOL=R I=R	Yes	Yes (partial)
INTERPOL=SM I=SM	Yes	No
INTERPOL=SPLINE I=SPLINE	Yes	Yes
INTERPOL=STD I=STD	Yes	Yes (partial)

Option	Supported by ActiveX?	Supported by Java?
INTERPOL=STEP I=STEP	Yes	Yes
LINE= L=	Yes (GPLOT) No (GCONTOUR)	Yes (GPLOT) No (GCONTOUR)
MODE=	Yes	Yes (partial)
POINTLABEL=	Yes (partial)	Yes (partial)
REPEAT= R=	Yes (GPLOT) No (GCONTOUR)	Yes (GPLOT) No (GCONTOUR)
STEP= S=	No	No
VALUE= V=	Yes (partial for GPLOT) No (GCONTOUR)	Yes (partial for GPLOT) No (GCONTOUR)
WIDTH= W=	Yes (partial for GPLOT) No (GCONTOUR)	Yes (partial for GPLOT) No (GCONTOUR)

POINTLABEL= Label Description Options

Table A1.9 ActiveX and Java Support for POINTLABEL Description Suboptions

Option	Supported by ActiveX?	Supported by Java?
COLOR= C=	Yes	No
FONT= F=	Yes	No
HEIGHT= H=	Yes	No
JUSTIFY= J=	No	No
POSITION=	No	No
"#var" "#x:#y <\$char>" "#y:#x \$<char>"	Yes (partial)	Yes (partial)

TITLE and FOOTNOTE Statements

Table A1.10 ActiveX and Java Support for TITLE and FOOTNOTE Statements

Option	Supported by ActiveX?	Supported by Java?
ANGLE=	No	No
BCOLOR=	Yes	Yes
BLANK=	No	No

Option	Supported by ActiveX?	Supported by Java?
BOX=	No	No
BSPACE=	No	No
COLOR=	Yes	Yes
DRAW=	No	No
FONT=	Yes	Yes
HEIGHT=	Yes (partial)	Yes (partial)
JUSTIFY=	Yes	Yes
LANGLE=	No	No
LINK=	Yes	Yes
LSPACE=	No	No
MOVE=	No	No
ROTATE=	No	No
UNDERLIN=	Yes (partial)	Yes (partial)

PROC GAREABAR

Table A1.11 ActiveX and Java Support for GAREABAR

Statement	Option	Supported by ActiveX?	Supported by Java?
PROC GAREABAR	DATA=	Yes	No
HBAR and VBAR	NAME=	Yes	No
	RSTAT=	Yes	No
	RESPSTAT=		
	RESPONSESTAT=		
	SUBGROUP=	Yes	No
	SUMVAR=	Yes	No
	WSTAT=	Yes	No
	WIDTHSTAT=		

PROC GBARLINE

Table A1.12 ActiveX and Java Support for PROC GBARLINE

Statement	Option	Supported by ActiveX?	Supported by Java?
PROC GBARLINE	ANNOTATE= ANNO=	Yes	No
	DATA=	Yes	No
	IMAGEMAP=	No	No
BAR	ANNOTATE= ANNO=	Yes	No
	ASCENDING	Yes	No
	AUTOREF	Yes	No
	AXIS=	Yes	No
	CAUTOREF=	Yes	No
	CAXIS=	Yes	No
	CERROR=	Yes	No
	CFRAME= CFR=	Yes	No
	CFREQ	Yes	No
	CLIPREF	Yes	No
	CLM=	Yes	No
	COUTLINE=	Yes	No
	CPERCENT CPCT	Yes	No
	CREF=	Yes	No
	CTEXT=	Yes	No
	DESCENDING	Yes	No
	DESCRIPTION= DES=	Yes	No
	DISCRETE	Yes	No
	ERRORBAR=	Yes	No
	FRAME NOFRAME FR NOFR	Yes	No
	FREQ	Yes	No
	FREQ= <i>numeric- variable</i>	No	No
	FRONTREF	Yes	No

Statement	Option	Supported by ActiveX?	Supported by Java?
	HTML=	Yes	No
	INSIDE=	Yes	No
	LAUTOREF=	Yes	No
	LEVELS=	Yes	No
	LREF=	Yes	No
	LR=		
	MAXIS=	Yes (partial)	No
	MEAN	Yes	No
	MIDPOINTS= <i>value-</i> <i>list</i>	Yes	No
	MIDPOINTS=OLD	Yes	No
	MINOR=	Yes	No
	MISSING	Yes	No
	NAME=	Yes	No
	NOAXIS	Yes	No
	NOBASEREF	Yes	No
	NOZERO	Yes	No
	OUTSIDE=	Yes	No
	PATTERNID=	Yes	No
	PERCENT	Yes	No
	PCT		
	RANGE	Yes	No
	RAXIS=	Yes (partial)	No
	AXIS=		
	REF=	Yes	No
	SPACE=	Yes	No
	SUM	Yes	No
	SUMVAR=	Yes	No
	TYPE=	Yes	No
	WIDTH=	Yes	No
	WOUTLINE=	Yes	No
PLOT	ASCENDING	Yes	No
	AXIS=	Yes	No
	FREQ= <i>numeric-</i> <i>variable</i>	No	No
	HTML=	No	No
	MINOR=	Yes	No
	NOLINE	Yes	No

Statement	Option	Supported by ActiveX?	Supported by Java?
	NOMARKER	Yes	No
	RAXIS= AXIS=	Yes	No
	SUMVAR=	Yes	No
	TYPE=	Yes	No

PROC GCHART

Table A1.13 ActiveX and Java Support for PROC GCHART

Statement	Option	Supported by ActiveX?	Supported by Java?
PROC GCHART	ANNOTATE= ANNO=	Yes	Yes
	DATA=	Yes	Yes
	GOUT=	Yes	Yes
	IMAGEMAP=	No	No
BLOCK	ANNOTATE= ANNO=	Yes	Yes
	BLOCKMAX=	No	No
	CAXIS=	Yes	Yes
	COUTLINE=	Yes (partial)	Yes (partial)
	CTEXT=	Yes	Yes
	DESCRIPTION= DES=	Yes	Yes
	DISCRETE	Yes	Yes
	FREQ=	Yes	Yes
	G100	Yes	Yes
	GROUP=	Yes	Yes
	HTML=	Yes	Yes
	HTML_LEGEND=	No	No
	LEGEND=	Yes (partial)	Yes (partial)
	LEVELS=	Yes	Yes
	MIDPOINTS= <i>value-</i> <i>list</i>	Yes	Yes
	MIDPOINTS=OLD	Yes	Yes
	MISSING	Yes	Yes
	NAME=	Yes	Yes
	NOHEADING	No	No

Statement	Option	Supported by ActiveX?	Supported by Java?
	NOLEGEND	Yes	Yes
	PATTERNID=	Yes	Yes
	SUBGROUP=	Yes	Yes
	SUMVAR=	Yes	Yes
	TYPE=	Yes	Yes
	WOUTLINE=	Yes	No
HBAR, HBAR3D, VBAR, and VBAR3D	ANNOTATE= ANNO=	Yes	Yes
	ASCENDING	Yes	Yes
	AUTOREF	Yes	Yes
	AXIS=	Yes	Yes
	CAUTOREF=	Yes	Yes
	CAXIS=	Yes	Yes
	CFRAME= CFR=	Yes	Yes
	CERROR=	Yes	Yes
	CFREQ	Yes	Yes
	CFREQLABEL=	No	No
	CLIPREF	Yes	Yes
	CLM=	Yes	Yes
	COUTLINE=	Yes	Yes
	CPERCENT CPCT	Yes	Yes
	CPERCENTLABEL=	No	No
	CREF=	Yes	Yes
	CTEXT=	Yes	Yes
	DESCENDING	Yes	Yes
	DESCRIPTION= DES=	Yes	Yes
	DISCRETE	Yes	Yes
	ERRORBAR=	Yes	Yes
	FRAME NOFRAME FR NOFR	Yes	Yes
	FREQ	Yes	Yes
	FREQLABEL=	No	No
	FREQ= <i>numeric- variable</i>	Yes	Yes

Statement	Option	Supported by ActiveX?	Supported by Java?
	FRONTREF	Yes	Yes
	G100	Yes	Yes
	GAXIS=	Yes (partial)	Yes (partial)
	GROUP=	Yes	Yes
	GSPACE=	Yes	Yes
	HTML=	Yes	Yes
	HTML_LEGEND=	No	No
	IFRAME=	Yes	No
	IMAGESTYLE=	Yes	Yes (partial)
	INSIDE=	Yes	Yes
	LAUTOREF=	Yes	Yes
	LEGEND=	Yes	Yes
	LEVELS=	Yes	Yes
	LREF=	Yes	No
	LR=		
	MAXIS=	Yes (partial)	Yes (partial)
	MEAN	Yes	Yes
	MEANLABEL=	No	No
	MIDPOINTS= <i>value-</i> <i>list</i>	Yes	Yes
	MIDPOINTS=OLD	Yes	Yes
	MINOR=	Yes	Yes
	MISSING	Yes	Yes
	NAME=	Yes	Yes
	NOAXIS	Yes	Yes
	NOBASEREF	Yes	Yes
	NOLEGEND	Yes	Yes
	NOSTATS	Yes	No
	NOZERO	Yes	No
	OUTSIDE=	Yes	Yes
	PATTERNID=	Yes	Yes
	PERCENT	Yes	Yes
	PCT		
	PERCENTLABEL=	No	No
	RANGE	Yes	Yes
	RAXIS= AXIS=	Yes (partial)	Yes (partial)
	REF=	Yes	Yes

Statement	Option	Supported by ActiveX?	Supported by Java?
	SHAPE=	Yes	Yes
	SPACE=	Yes	Yes
	SUBGROUP=	Yes	Yes
	SUM	Yes	Yes
	SUMLABEL=	No	No
	SUMVAR=	Yes	Yes
	TYPE=	Yes	Yes
	WIDTH=	Yes	Yes
	WOUTLINE=	Yes	No
PIE, PIE3D, and DONUT	ACROSS=	Yes	Yes
	ANGLE=	Yes	Yes
	ANNOTATE= ANNO=	Yes	Yes
	ASCENDING	Yes	Yes
	CFILL=	Yes	Yes
	CLOCKWISE	Yes	Yes
	COUTLINE=	Yes	Yes
	CTEXT=	Yes	Yes
	DESCENDING	Yes	Yes
	DESCRIPTION= DES=	Yes	Yes
	DETAIL=	Yes	Yes
	DETAIL_PERCENT=	Yes	Yes
	DETAIL_RADIUS=	Yes	Yes
	DETAIL_SLICE=	Yes	Yes
	DETAIL_THRESHOLD=	Yes	Yes
	DETAIL_VALUE=	Yes	Yes
	DISCRETE	Yes	Yes
	DONUTPCT=	Yes	Yes
	DOWN=	Yes	Yes
	EXPLODE=	Yes	Yes
	FILL=	Yes (partial)	Yes (partial)
	FREQ=	Yes	Yes
	GROUP=	Yes	Yes
	HTML=	Yes	Yes
	HTML_LEGEND=	No	No
	INVISIBLE=	Yes	Yes

Statement	Option	Supported by ActiveX?	Supported by Java?
	JSTYLE	Yes	Yes
	LABEL=	Yes (partial)	Yes (partial)
	LEGEND=	Yes (partial)	Yes (partial)
	LEVELS=	Yes	Yes
	MATCHCOLOR	Yes	Yes
	MIDPOINTS= <i>value-list</i>	Yes	Yes
	MIDPOINTS=OLD	Yes	Yes
	MISSING	Yes	Yes
	NAME=	Yes	Yes
	NOGROUPHEADING	Yes	Yes
	NOHEADING	No	No
	NOLEGEND	Yes	Yes
	OTHER=	Yes	Yes
	OTHERCOLOR=	Yes	Yes
	OTHERLABEL=	Yes	Yes
	PERCENT=	Yes	Yes
	SLICE=	Yes	Yes
	SUBGROUP=	Yes	Yes
	SUMVAR=	Yes	Yes
	TYPE=	Yes	Yes
	VALUE=	Yes	Yes
	WOUTLINE=	Yes	No
STAR		No	No

Text Description Suboptions

Text description suboptions are used by the LABEL= option in the DONUT statement.

Table A1.14 ActiveX and Java Support for LABEL Text Description Suboptions

Option	Supported by ActiveX?	Supported by Java?
ANGLE= A=	Yes	No
COLOR= C=	Yes	Yes
FONT= F=	Yes (partial)	Yes (partial)

Option	Supported by ActiveX?	Supported by Java?
HEIGHT= H=	Yes	Yes
JUSTIFY= J=	No	No
ROTATE= R=	Yes	No

PROC GCONTOUR

Table A1.15 ActiveX and Java Support for PROC GCONTOUR

Statement	Option	Supported by ActiveX?	Supported by Java?
PROC GCONTOUR	ANNOTATE= ANNO=	Yes	Yes
	DATA=	Yes	Yes
	GOUT=	No	No
	INCOMPLETE	No	No
PLOT	ANNOTATE= ANNO=	Yes	Yes
	AUTOHREF	Yes	No
	AUTOLABEL=	No	No
	AUTOVREF	Yes	No
	CAUTOHREF=	Yes	No
	CAUTOVREF=	Yes	No
	CAXIS=	Yes	Yes
	CFRAME= CFR=	Yes	Yes
	CHREF= CH=	Yes	No
	CLEVELS=	Yes	Yes
	COUTLINE=	Yes (partial)	Yes
	CTEXT=	Yes	Yes
	CVREF= CV=	Yes	No
	DESCRIPTION= DES=	Yes	Yes
	GRID	Yes	No
	HAXIS=	Yes (partial)	Yes (partial)

Statement	Option	Supported by ActiveX?	Supported by Java?
	HMINOR= HM=	Yes	Yes
	HREF=	Yes	No
	HREVERSE=	Yes	No
	JOIN	Yes	Yes
	LAUTOHREF=	Yes	No
	LAUTOVREF=	Yes	No
	LEGEND=	Yes (partial)	Yes (partial)
	LEVELS=	Yes	Yes
	LHREF= LH=	Yes (partial)	Yes (partial)
	LLEVELS=	Yes	No
	LVREF= LV=	Yes (partial)	Yes (partial)
	NAME=	Yes	Yes
	NLEVELS=	Yes	Yes
	NOAXIS NOAXES	Yes	Yes
	NOFRAME	Yes	Yes
	NOLEGEND	Yes	Yes
	PATTERN	Yes (partial)	Yes (partial)
	VAXIS=	Yes (partial)	Yes (partial)
	VMINOR= VM=	Yes	Yes
	VREF=	Yes	No
	VREVERSE	Yes	No
	XTICKNUM= YTICKNUM=	Yes	Yes

PROC GMAP

Table A1.16 ActiveX and Java Support for PROC GMAP

Statement	Option	Supported by ActiveX?	Supported by Java?
PROC GMAP	MAP=	Yes	Yes
	ALL	Yes	Yes
	ANNOTATE=	Yes	Yes
	ANNO=		

Statement	Option	Supported by ActiveX?	Supported by Java?
	DATA=	Yes	Yes
	GOUT=	No	No
	IMAGEMAP=	No	No
ID		Yes	Yes
BLOCK	ANNOTATE=	Yes	Yes
	ANNO=		
	AREA=	Yes	Yes
	BLOCKSIZE=	Yes	Yes
	CBLKOUT=	Yes	Yes
	CEMPTY=	Yes	No
	COUTLINE=	Yes	Yes
	CTEXT=	Yes	Yes
	DESCRIPTION=	Yes	Yes
	DES=		
	DISCRETE	Yes	Yes
	HTML=	Yes	Yes
	HTML_LEGEND=	No	No
	LEGEND=	Yes (partial)	Yes (partial)
	LEVELS=	Yes	Yes
	MIDPOINTS=	Yes	Yes (partial)
	MISSING	Yes	Yes
	NAME=	Yes	Yes
	NOLEGEND	Yes	Yes
	SHAPE=	Yes	Yes
	WOUTLINE=	Yes	No
	XSIZE=	No	No
	YSIZE=		
	XVIEW=	Yes	Yes (partial)
	YVIEW=		
	ZVIEW=		
CHORO	ANNOTATE=	Yes	Yes
	ANNO=		
	CEMPTY=	Yes	No
	COUTLINE=	Yes	Yes
	CTEXT=	Yes	Yes
	DESCRIPTION=	Yes	Yes
	DES=		
	DISCRETE	Yes	Yes
	HTML=	Yes	Yes

Statement	Option	Supported by ActiveX?	Supported by Java?
	HTML_LEGEND=	No	No
	LEGEND=	Yes (partial)	Yes (partial)
	LEVELS=	Yes	Yes
	MIDPOINTS=	Yes	Yes (partial)
	MISSING	Yes	Yes
	NAME=	Yes	Yes
	NOLEGEND	Yes	Yes
	WOUTLINE=	Yes	No
	XSIZE=	No	No
	YSIZE=		
	XVIEW=	Yes	Yes (partial)
	YVIEW=		
	ZVIEW=		
PRISM	ANNOTATE=	Yes	Yes
	ANNO=		
	AREA=	Yes	Yes
	CEMPTY=	Yes	No
	COUTLINE=	Yes	Yes
	CTEXT=	Yes	Yes
	DESCRIPTION=	Yes	Yes
	DES=		
	DISCRETE	Yes	Yes
	HTML=	Yes	Yes
	HTML_LEGEND=	No	No
	LEGEND=	Yes (partial)	Yes (partial)
	LEVELS=	Yes	Yes
	MIDPOINTS=	Yes	Yes (partial)
	MISSING	Yes	Yes
	NAME=	Yes	Yes
	NOLEGEND	Yes	Yes
	WOUTLINE=	No	No
	XLIGHT=	No	No
	YLIGHT=		
	XSIZE=	No	No
	YSIZE=		

Statement	Option	Supported by ActiveX?	Supported by Java?
	XVIEW= YVIEW= ZVIEW=	Yes	Yes (partial)
SURFACE		No	No

PROC GPLOT

When used with the JAVA or JAVAMETA device driver, the BUBBLE statement must have at least one axis that is assigned to a numeric variable.

Table A1.17 ActiveX and Java Support for PROC GPLOT

Statement	Option	Supported by ActiveX?	Supported by Java?
PROC GPLOT	ANNOTATE= ANNO=	Yes	Yes
	DATA=	Yes	Yes
	GOUT=	Yes	Yes
	IMAGEMAP=	Yes	Yes
	UNIFORM	No	No
BUBBLE	ANNOTATE= ANNO=	Yes	Yes
	AUTOHREF	Yes	Yes
	AUTOVREF	Yes	Yes
	BCOLOR=	Yes	Yes
	BFONT=	No	No
	BLABEL	Yes	Yes
	BSCALE=	No	No
	BSIZE=	Yes (partial)	Yes (partial)
	CAUTOHREF=	Yes	Yes
	CAUTOVREF=	Yes	Yes
	CAXIS= CA=	Yes	Yes
	CFRAME= CFR=	Yes	Yes
	CHREF= CH=	Yes	Yes
	CTEXT= C=	Yes	Yes
	CVREF= CV=	Yes	Yes

Statement	Option	Supported by ActiveX?	Supported by Java?
	DESCRIPTION= DES=	Yes	Yes
	FRAME NOFRAME FR NOFR	Yes	Yes
	GRID	Yes	Yes
	HAXIS=	Yes (partial)	Yes (partial)
	HMINOR= HM=	Yes	Yes
	HREF=	Yes	Yes
	HREVERSE	Yes	Yes (partial)
	HZERO	Yes	Yes
	IFRAME=	Yes	No
	IMAGESTYLE=	Yes	No
	LAUTOHREF=	Yes	Yes
	LAUTOVREF=	Yes	Yes
	LHREF= LH=	Yes	Yes
	LVREF= LV=	Yes	Yes
	NAME=	Yes	Yes
	NOAXIS NOAXES	Yes	Yes
	VAXIS=	Yes (partial)	Yes (partial)
	VMINOR= VM=	Yes	Yes
	VREF=	Yes	Yes
	VREVERSE	Yes	Yes
	VZERO	Yes	Yes
BUBBLE2	ANNOTATE= ANNO=	Yes	Yes
	AUTOVREF	Yes	Yes
	BCOLOR=	Yes	Yes
	BFONT=	No	No
	BLABEL	Yes	Yes
	BSCALE=	No	No
	BSIZE=	Yes (partial)	Yes (partial)
	CAUTOVREF=	Yes	Yes

Statement	Option	Supported by ActiveX?	Supported by Java?
	CAXIS= CA=	Yes	Yes
	CFRAME= CFR=	Yes	Yes
	CTEXT= C=	Yes	Yes
	CVREF= CV=	Yes	Yes
	FRAME NOFRAME FR NOFR	Yes	Yes
	GRID	Yes	Yes
	LAUTOVREF=	Yes	Yes
	LVREF= LV=	Yes	Yes
	NOAXIS NOAXES	Yes	Yes
	VAXIS=	Yes (partial)	Yes (partial)
	VMINOR= VM=	Yes	Yes
	VREF=	Yes	Yes
	VREVERSE	Yes	Yes
	VZERO	Yes	Yes
PLOT	ANNOTATE= ANNO=	Yes	Yes
	AREAS=	Yes	Yes (partial)
	AUTOHREF	Yes	Yes
	AUTOVREF	Yes	Yes
	CAUTOHREF=	Yes	Yes
	CAUTOVREF=	Yes	Yes
	CAXIS= CA=	Yes	Yes
	CFRAME= CFR=	Yes	Yes
	CHREF= CH=	Yes	Yes
	COUTLINE=	Yes	No
	CTEXT= C=	Yes	Yes
	CVREF= CV=	Yes (partial)	Yes (partial)

Statement	Option	Supported by ActiveX?	Supported by Java?
	DESCRIPTION= DES=	Yes	Yes
	FRAME NOFRAME FR NOFR	Yes	Yes
	GRID	Yes	Yes
	HAXIS=	Yes (partial)	Yes (partial)
	HMINOR= HM=	Yes	Yes
	HREF=	Yes	Yes
	HREVERSE	Yes	Yes (partial)
	HTML=	Yes (partial)	Yes (partial)
	HTML_LEGEND=	No	No
	HZERO	Yes	Yes
	IFRAME=	Yes	No
	IMAGESTYLE=	Yes	No
	LAUTOHREF=	Yes	Yes
	LAUTOVREF=	Yes	Yes
	LEGEND=	Yes	Yes
	LHREF= LH=	Yes	Yes
	LVREF= LV=	Yes (partial)	Yes (partial)
	NAME=	Yes	Yes
	NOAXIS NOAXES	Yes	Yes
	NOLEGEND	Yes	Yes
	OVERLAY	Yes	Yes (partial)
	REGEQN	No	No
	SKIPMISS	Yes	Yes
	VAXIS=	Yes (partial)	Yes (partial)
	VMINOR= VM=	Yes	Yes
	VREF=	Yes	Yes
	VREVERSE	Yes	Yes
	VZERO	Yes	Yes
PLOT2	With INTERPOL= BOX, HILO, or STD	No	No

Statement	Option	Supported by ActiveX?	Supported by Java?
	ANNOTATE= ANNO=	Yes	Yes
	AREAS=	Yes	Yes (partial)
	AUTOVREF	Yes	Yes
	CAUTOVREF=	Yes	Yes
	CAXIS= CA=	Yes	Yes
	CFRAME= CFR=	Yes	Yes
	COUTLINE=	Yes	No
	CTEXT= C=	Yes	Yes
	CVREF= CV=	Yes	Yes
	FRAME NOFRAME FR NOFR	Yes	Yes
	GRID	Yes	Yes
	HTML=	Yes (partial)	Yes (partial)
	HTML_LEGEND=	No	No
	LAUTOVREF=	Yes	Yes
	LEGEND=	Yes	Yes
	LVREF= LV=	Yes	Yes
	NAME=	Yes	Yes
	NOAXIS NOAXES	Yes	Yes
	NOLEGEND	Yes	Yes
	OVERLAY	Yes	Yes (partial)
	REGEQN	No	No
	SKIPMISS	Yes	Yes
	VAXIS=	Yes (partial)	Yes (partial)
	VMINOR= VM=	Yes	Yes
	VREF=	Yes	Yes

Statement	Option	Supported by ActiveX?	Supported by Java?
	VREVERSE	Yes	Yes
	VZERO	Yes	Yes

PROC GRADAR

Table A1.18 ActiveX and Java Support for PROC GRADAR

Statement	Option	Supported by ActiveX?	Supported by Java?
PROC GRADAR	ANNOTATE=	Yes	No
	DATA=	Yes	No
	GOUT=	Yes	No
CHART	ACROSS=	Yes	No
	ACROSSVAR=		
	ANNOTATE=	No	No
	ANNO=		
	CAXIS=	No	No
	CAXES=		
	CA=		
	CFRAME=	Yes	No
	CFR=		
	CFRAMESIDE=	Yes	No
	CFRAMETOP=	Yes	No
	CSPOKES=	Yes	No
	CSPOKE=		
	CSTARCIRCLES=	Yes	No
	CSTARCIRCLE=		
	CSTARFILL=	Yes	No
	CSTARS=	Yes	No
	CSTAR=		
	CTEXT=	Yes	No
	CTILES=	No	No
	CTILE=		
	DESCRIPTION=	Yes	No
	DES=		
DOWN=	Yes	No	
DOWNVAR=			
FONT=	Yes	No	
FREQ=	Yes	No	
FRAME	No	No	
NOFRAME			

Statement	Option	Supported by ActiveX?	Supported by Java?
	HEIGHT= HLABEL=	Yes	No
	HTML=	Yes	No
	HTML_LEGEND=	Yes	No
	IFRAME=	No	No
	IMAGESTYLE=	Yes	No
	INBORDER	No	No
	INHEIGHT=	No	No
	INTERTILE= INTERCHART=	Yes	No
	LAST=	No	No
	LSPOKE=	Yes	No
	LSTARCIRCLES= LSTARCIRCLE=	Yes	No
	LSTARS= LSTAR=	Yes	No
	MAXNVERT= MAXVERT=	Yes	No
	MISSING	No	No
	MODE=	Yes	No
	NAME=	Yes	No
	NCOLS= NCOL=	No	No
	NOZEROREF	Yes	No
	NROWS= NROW=	No	No
	ORDERACROSS=	No	No
	OTHER=	Yes	No
	OVERLAY= OVERLAYVAR=	Yes	No
	SPIDERWEB SPIDER	Yes	No
	SPKLABEL=	Yes	No
	STARAXIS= STARAXES=	No	No
	STARCIRCLES= STARCIRCLE=	Yes	No
	STARFILL=	Yes	No
	STARINRADIUS= STAROUTRADIUS=	No	No
	STARLEGEND=	Yes	No

Statement	Option	Supported by ActiveX?	Supported by Java?
	STARLEGENDLAB=	No	No
	STARSTART=	Yes	No
	STARTYPE=	Yes	No
	SUMVAR=	Yes	No
	TILELEGEND=	No	No
	TILEGLABEL=	No	No
	WAXIS=	No	No
	WEIGHT=	No	No
	WSPOKES=	Yes	No
	WSPOKE=		
	WSTARCIRCLES=	Yes	No
	WSTARCIRCLE=		
	WSTARS=	Yes	No
	WSTAR=		

PROC G3D

Table A1.19 ActiveX and Java Support for PROC G3D

Statement	Option	Supported by ActiveX?	Supported by Java?
PROC G3D	ANNOTATE=	Yes	Yes
	ANNO=		
	DATA=	Yes	Yes
	GOUT=	Yes	Yes
PLOT	ANNOTATE=	Yes	Yes
	ANNO=		
	CAXIS=	Yes	Yes
	CBOTTOM=	No	No
	CTEXT=	No	No
	CTOP=	No	No
	DESCRIPTION=	Yes	Yes
	DES=		
	GRID	Yes	Yes
	NAME=	Yes	Yes
	NOAXIS	Yes	Yes
	NOAXES		
NOLABEL	Yes	Yes	

Statement	Option	Supported by ActiveX?	Supported by Java?
	ROTATE=	Yes (partial)	Yes (partial)
	SIDE	Yes	Yes
	TILT=	Yes (partial)	Yes (partial)
	XTICKNUM=	Yes	Yes
	YTICKNUM=		
	ZTICKNUM=		
	XYTYPE=	Yes	No
	ZMAX=	Yes	No
	ZMIN=		
SCATTER	ANNOTATE=	Yes	Yes
	ANNO=		
	CAXIS=	Yes	Yes
	COLOR=	Yes	Yes
	CTEXT=	Yes	Yes
	DESCRIPTION=	Yes	Yes
	DES=		
	GRID	Yes	Yes
	NAME=	Yes	Yes
	NOAXIS	Yes	Yes
	NOAXES		
	NOLABEL	Yes	Yes
	NONEEDLE	Yes	Yes
	ROTATE=	Yes (partial)	No
	SHAPE=	Yes	Yes
	SIZE=	Yes	Yes
	TILT=	Yes	No
	XTICKNUM=	Yes	Yes
	YTICKNUM=		
	ZTICKNUM=		
	ZMAX=	Yes	Yes (partial)
	ZMIN=		

Annotate Functions

BAR

Table A1.20 ActiveX and Java Support for the BAR Function

Variable	Supported by ActiveX?	Supported by Java?
COLOR	Yes	Yes
GROUP	Yes	Yes
HTML	Yes	No
LINE	Yes (Partial)	Yes (Partial)
MIDPOINT	Yes	Yes
SIZE	Yes	Yes
STYLE	Yes (Partial)	Yes (Partial)
SUBGROUP	Yes	Yes
WHEN	Yes	Yes
X, Y, Z	Yes	Yes
XC, YC	Yes	Yes
XSYS, YSYS, ZSYS	Yes	Yes

DRAW

Table A1.21 ActiveX and Java Support for the DRAW Function

Variable	Supported by ActiveX?	Supported by Java?
COLOR	Yes	Yes
GROUP	Yes	Yes
HSYS	No	Yes
LINE	Yes	Yes
MIDPOINT	Yes	Yes
SIZE	No	Yes
SUBGROUP	Yes	Yes
WHEN	Yes	Yes
X, Y, Z	Yes	Yes

Variable	Supported by ActiveX?	Supported by Java?
XC, YC	Yes	Yes
XSYS, YSYS, ZSYS	Yes	Yes

DRAW2TXT

Table A1.22 ActiveX and Java Support for the DRAW2TXT Function

Variable	Supported by ActiveX?	Supported by Java?
COLOR	Yes	Yes
HSYS	No	Yes
LINE	Yes	Yes
SIZE	No	Yes
WHEN	Yes	Yes

FRAME

Table A1.23 ActiveX and Java Support for the FRAME Function

Variable	Supported by ActiveX?	Supported by Java?
COLOR	Yes	No
HSYS	No	No
HTML	Yes	No
LINE	Yes	No
SIZE	Yes	No
STYLE	Yes	No
WHEN	Yes	No
XSYS, YSYS	Yes	No

IMAGE

Table A1.24 ActiveX and Java Support for the IMAGE Function

Variable	Supported by ActiveX?	Supported by Java?
HTML	Yes	No
IMGPATH	Yes	No
STYLE	Yes	No

Variable	Supported by ActiveX?	Supported by Java?
WHEN	Yes	No
X, Y	Yes	No
XSYS, YSYS	Yes	No

LABEL

Table A1.25 ActiveX and Java Support for the LABEL Function

Variable	Supported by ActiveX?	Supported by Java?
ANGLE	No	No
CBORDER	Yes	Yes
CBOX	Yes	Yes
COLOR	Yes	Yes
GROUP	Yes	Yes
HSYS	No	No
HTML	Yes	No
MIDPOINT	Yes	Yes
POSITION	Yes (Partial)	Yes (Partial)
ROTATE	No	No
SIZE	No	Yes
STYLE	Yes (Partial)	No
SUBGROUP	Yes	Yes
TEXT	Yes	Yes
WHEN	Yes	Yes
X, Y, Z	Yes	Yes
XC, YC	Yes	Yes
XSYS, YSYS, ZSYS	Yes	Yes

MOVE

Table A1.26 ActiveX and Java Support for the MOVE Function

Variable	Supported by ActiveX?	Supported by Java?
GROUP	Yes	Yes
MIDPOINT	Yes	Yes
SUBGROUP	Yes	Yes

Variable	Supported by ActiveX?	Supported by Java?
WHEN	Yes	Yes
X, Y, Z	Yes	Yes
XC, YC	Yes	Yes
XSYS, YSYS, ZSYS	Yes	Yes

PIE

Table A1.27 ActiveX and Java Support for the PIE Function

Variable	Supported by ActiveX?	Supported by Java?
ANGLE	Yes	Yes
COLOR	Yes	Yes
GROUP	Yes	Yes
HSYS	No	Yes
HTML	Yes	No
LINE	Yes (Partial)	Yes (Partial)
MIDPOINT	Yes	Yes
ROTATE	Yes	Yes
SIZE	Yes	Yes
STYLE	Yes (Partial)	Yes (Partial)
SUBGROUP	Yes	Yes
WHEN	Yes	Yes
X, Y, Z	Yes	Yes
XC, YC	Yes	Yes
XSYS, YSYS, ZSYS	Yes	Yes

PIECNTR

Table A1.28 ActiveX and Java Support for the PIECNTR Function

Variable	Supported by ActiveX?	Supported by Java?
GROUP	Yes	Yes
HSYS	No	Yes
MIDPOINT	Yes	Yes
SIZE	Yes	Yes
SUBGROUP	Yes	Yes

Variable	Supported by ActiveX?	Supported by Java?
WHEN	Yes	Yes
X, Y, Z	Yes	Yes
XC, YC	Yes	Yes
XSYS, YSYS, ZSYS	Yes	Yes

PIEXY

Table A1.29 ActiveX and Java Support for the PIEXY Function

Variable	Supported by ActiveX?	Supported by Java?
ANGLE	Yes	Yes
SIZE	Yes	Yes
WHEN	Yes	Yes

POINT

Table A1.30 ActiveX and Java Support for the POINT Function

Variable	Supported by ActiveX?	Supported by Java?
COLOR	Yes	Yes
GROUP	Yes	Yes
MIDPOINT	Yes	Yes
SUBGROUP	Yes	Yes
WHEN	Yes	Yes
X, Y, Z	Yes	Yes (Partial)
XC, YC	Yes	Yes (Partial)
XSYS, YSYS, ZSYS	Yes	Yes (Partial)

POLY

Table A1.31 ActiveX and Java Support for the POLY Function

Variable	Supported by ActiveX?	Supported by Java?
COLOR	Yes	Yes
GROUP	Yes	Yes
HTML	Yes	No

Variable	Supported by ActiveX?	Supported by Java?
LINE	Yes	No
MIDPOINT	Yes	Yes
SUBGROUP	Yes	Yes
SIZE	No	Yes
STYLE	Yes (Partial)	Yes (Partial)
WHEN	Yes	Yes
X, Y, Z	Yes	Yes
XC, YC	Yes	Yes
XSYS, YSYS, ZSYS	Yes	Yes

POLYCONT

Table A1.32 ActiveX and Java Support for the POLYCONT Function

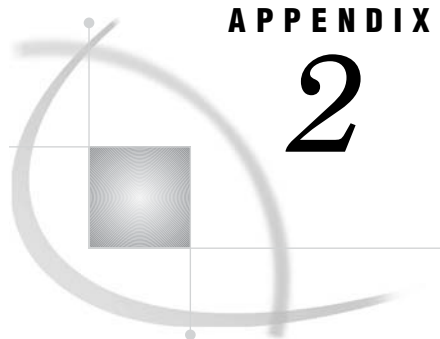
Variable	Supported by ActiveX?	Supported by Java?
COLOR	Yes	Yes
GROUP	Yes	Yes
MIDPOINT	Yes	Yes
SUBGROUP	Yes	Yes
WHEN	Yes	Yes
X, Y, Z	Yes	Yes
XC, YC	Yes	Yes
XSYS, YSYS, ZSYS	Yes	Yes

SYMBOL

Table A1.33 ActiveX and Java Support for the SYMBOL Function

Variable	Supported by ActiveX?	Supported by Java?
CBOX	No	No
CBORDER	No	No
COLOR	Yes	Yes
GROUP	Yes	Yes
SUBGROUP	Yes	Yes
HSYS	No	Yes
HTML	Yes	No

Variable	Supported by ActiveX?	Supported by Java?
MIDPOINT	Yes	Yes
SIZE	Yes	Yes
STYLE	Yes (Partial)	Yes (Partial)
TEXT	Yes (Partial)	Yes (Partial)
WHEN	Yes	Yes
X, Y, Z	Yes	Yes
XC, YC	Yes	Yes
XSYS, YSYS, ZSYS	Yes	Yes



APPENDIX

2

Recommended Reading

Recommended Reading 1547

Recommended Reading

Here is the recommended reading list for this title:

- *Annotate: Simply the Basics*
- *The How-To Book for SAS/GRAPH Software*
- *Multiple-Plot Displays: Simplified with Macros*
- *Output Delivery System: The Basics*
- *SAS Language Reference: Concepts*
- *SAS Language Reference: Dictionary*
- *SAS Output Delivery System: User's Guide*
- *SAS System for Statistical Graphics*
- *Visualizing Categorical Data*

For a complete list of SAS publications, see the current *SAS Publishing Catalog*. To order the most current publications or to receive a free copy of the catalog, contact a SAS representative at

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: (800) 727-3228*
Fax: (919) 677-8166
E-mail: sasbook@sas.com
Web address: support.sas.com/pubs

* For other SAS Institute business, call (919) 677-8000.

Customers outside the United States should contact their local SAS office.

Glossary

absolute coordinates

coordinates measured from the origin of the coordinate system. In two-dimensional graphs, the origin is (0,0). In three-dimensional graphs, the origin is (0,0,0). See also relative coordinates.

aspect ratio

the ratio of width to height (that is, width divided by height) in an output area such as a display, plotter, or film recorder. In SAS/GRAPH software, the ASPECT= graphics option simulates a change in the aspect ratio of the display, causing fonts and circles to be compressed horizontally or vertically or both.

axis

a one-dimensional line representing the zero point on a scale used to plot values of x , y , or z coordinates. In SAS/GRAPH software, in two dimensions, the X axis represents the horizontal plane, and the Y axis represents the vertical plane. In three dimensions, the X axis represents width, the Y axis represents depth, and the Z axis represents height. See also Cartesian coordinate system. The term *axis* may also refer collectively to the axis line, the major and minor tick marks, the major tick mark values, and the axis label.

axis area

an area bounded by axes. In SAS/GRAPH software, this area may be enclosed by an axis frame. See also frame.

baseline

in a font, the imaginary line upon which the characters rest.

block map

a three-dimensional map that uses blocks of varying heights to represent the value of a variable for each map area.

border

in SAS/GRAPH software, the line drawn around the entire graphics output area. This area includes the title and footnote areas as well as the procedure output area. See also frame.

boundary

in the GMAP procedure, a separating line or point that distinguishes between two or more unit areas or segments.

BY group

all observations with the same values for all BY variables.

BY-group processing

the process of using the BY statement to process observations that are ordered, grouped, or indexed according to the values of one or more variables. Many SAS procedures and the DATA step support BY-group processing.

BY variable

a variable named in a BY statement whose values define groups of observations to process.

Cartesian coordinate system

the two- or three-dimensional coordinate system in which perpendicular axes meet at the origin (0,0) or (0,0,0). Typically, Cartesian coordinate axes are called X, Y, and Z. See also axis.

Cartesian coordinates

values that locate a point in two- or three-dimensional space. Each value represents units measured along an X, Y, or Z axis. See also Cartesian coordinate system.

capline

the highest point of a normal uppercase letter. In some fonts, the capline may be above the top of the letter to allow room for an accent.

catalog

See SAS catalog.

catalog entry

See entry type and SAS catalog entry.

cell

a unit of measure defined by the number of rows and the number of columns in the graphics output area. See also aspect ratio.

CGM

an abbreviation for computer graphics metafile. A CGM is a graphics output file written in the internationally recognized format for describing computer graphics images. This standardization allows any image in a CGM to be imported and exported among different systems without error or distortion.

character string

one or more alphanumeric or other keyboard characters or both.

character value

a value that can contain alphabetic characters, numeric characters 0 through 9, and other special characters. See also character variable.

character variable

a variable whose values can consist of alphabetic and special characters as well as numeric characters.

chart

a graph in which graphics elements (bars, pie slices, and so on) show the magnitude of a statistic. The graphics elements can represent one data value or a range of data values.

chart statistic

the statistical value calculated for the chart variable: frequency, cumulative frequency, percentage, cumulative percentage, sum, or mean.

chart variable

a variable in the input data set whose values are categories of data represented by bars, blocks, slices, or spines.

choropleth map

a two-dimensional map that uses color and fill pattern combinations to represent different categories or levels of magnitude.

class variable

in some SAS procedures, a variable used to group, or classify, data. Class variables can be character or numeric. Class variables can have continuous values, but they typically have a few discrete values that define the classifications of the variable.

classification variable

See class variable.

CMYK

a color-coding scheme that specifies a color in terms of levels of cyan, magenta, yellow, and black components. The levels of each component range from 0 to 255. See also HLS, HSV, and RGB.

color map

a table that is used to translate the original colors in graphics output to different colors when replaying graphics output using the GREPLAY procedure. The table is contained in a catalog entry.

color, predefined

one of the set of colors for which SAS/GRAPH software defines and recognizes names, for example, BLACK, BLUE, and CYAN.

color, user-defined

a color expressed in CMYK, HLS, HSV, RGB, or gray-scale format. See also CMYK, HLS, HSV, RGB, and gray scale.

colors list

the list of foreground colors available for the graphics output. The colors list is either the default list established from the device entry or the list established from the colors specified with the COLORS= graphics option.

computer graphics metafile

See CGM.

confidence limits

the upper and lower values of a confidence interval. There is a percentage of confidence (typically 95%) that the true value of the parameter being estimated lies within the interval.

contour plot

a three-variable plot that uses line styles or patterns to represent levels of magnitude of z corresponding to x and y coordinates.

coordinate system

the context in which to interpret coordinates. Coordinate systems vary according to their origin, limits, and units. See also Cartesian coordinate system.

coordinates

the values representing the location of a data point or a graphics element along the X, Y, and Z axes. Coordinate values are measured from the origin of the coordinate system.

data area

the portion of the graphics output area in which data values are displayed. In the Annotate facility, the data area defines a coordinate system. In plots and bar charts, the data area is bounded by axes; in choropleth maps, the data area is bounded by the edge of the unit areas. See also graphics output area, procedure output area, and coordinate system.

data value

a unit of character or numeric information in a SAS data set. A data value represents one variable in an observation. In the rectangular structure of a SAS data set, intersection of a row and a column.

date value

See SAS date value.

default

(1) The setting of a value, parameter, or argument used by the SAS System if the user does not specify a setting.

(2) the value, parameter, or option setting used by the SAS System if the user specifies no particular setting.

density value

a value assigned to each observation in a map data set reflecting the amount of detail (resolution) contributed by the observation.

dependent variable

a variable whose value is determined by the value of another variable or set of variables.

device driver

a routine that generates the specific machine-language commands needed to display graphics output on a particular device. SAS/GRAPH device drivers take device-independent graphics information produced by SAS/GRAPH procedures and create the commands required to produce the graph on the particular device.

device entry

a SAS catalog entry that stores the values of device parameters (or the characteristics) that are used with a particular output device.

device map

a catalog entry used to convert the SAS/GRAPH internal encoding for one or more characters to the device-specific encoding needed to display the character(s) in hardware text on a particular graphics output device. See also hardware character set.

device parameter

a value in a device entry that defines a default behavior or characteristic of a device driver. Some device parameters can be overridden by graphics options. See also graphics option.

display

the area of the monitor that displays what the software presents to you.

display manager

See SAS Display Manager System.

entry type

a characteristic of a SAS catalog entry that identifies its structure and attributes to the SAS System. When you create an entry, the SAS System automatically assigns the entry type as part of the name.

export

to put a SAS catalog entry containing graphics output into a format that can be moved to another software product.

feature table

a SAS data set that uses the \$GEOREF format to store geometric coordinates for each unique map area in a single variable value. See also \$GEOREF format, geo-variable, map area, map data set, and traditional map data set.

fileref

a name temporarily assigned to an external file or to an aggregate storage location that identifies it to the SAS System. You assign a fileref with a FILENAME statement or with an operating system command.

fill pattern

a design of parallel or crosshatched lines, solid colors, or empty space used to fill an area in a graph.

font

a complete set of all the characters of the same design and style. The characters in a font can be figures or symbols as well as alphanumeric characters. See also type style.

font maximum

in the GFONT procedure, the highest vertical coordinate in a font.

font minimum

in the GFONT procedure, the lowest vertical coordinate in a font.

font units

in the GFONT procedure, units defined by the range of coordinates specified in the font data set. For example, a font in which the vertical coordinates range from 10 to 100 has 90 font units.

font, hardware

a font stored in an output device. See also font, software.

font, software

a font in which the characters are drawn by graphics software. See also font, hardware.

format

an instruction the SAS System uses to display or write each value of a variable. Some formats are supplied by SAS software. Other formats can be written by the user with the FORMAT procedure in base SAS software or with SAS/TOOLKIT software.

frame

a box enclosing a group of graphics elements. In GSLIDE procedure output, the frame encloses the procedure output area. In GPLOT, GCHART, and GCONTOUR procedure output, the frame encloses the axis area. In a legend, the frame encloses the legend label and entries. See also border.

geo-variable

the \$GEOREF formatted variable in a feature table that stores the spatial information as a geometry object. When a feature table is used, this variable is specified in the ID statement of the GMAP procedure. See also \$GEOREF variable, identification variable, and feature table.

\$GEOREF format

a geometric coordinate data arrangement that stores all the spatial information as a geometry object contained in a single variable. This format, which is used by feature tables, references to the geometry objects that encapsulate the points, lines, and polygons necessary to render a map. See also geo-variable and feature table.

global statement

a SAS statement that you can specify anywhere in a SAS program.

graph

a visual representation of data showing the variation of a variable in comparison to one or more other variables.

graphics element

a discrete visual part of a picture. For example, a bar in a chart and a plot's axis label are both graphics elements.

graphics device

See graphics output device.

graphics option

a value specified in a GOPTIONS statement that controls some attribute of the graphics output. The values specified remain in effect only for the duration of the SAS session. Some graphics options override device parameters.

graphics output

output from a graphics program that can be stored as a catalog entry of type GRSEG, or as a graphics stream file. Graphics output can be displayed or printed on a graphics output device. See also graphics output device and graphics stream file (GSF).

graphics output area

the area of a graphics output device where the graphics output is displayed or drawn. Typically, the graphics output area occupies the full drawing area of the device, but the dimensions of the graphics output area can be changed with graphics options or device parameters. See also procedure output area and graphics output device.

graphics output device

any terminal, printer, or other output device capable of displaying or producing graphics output. See also graphics output.

graphics stream file (GSF)

a file containing device-dependent graphics commands from a SAS/GRAPH device driver. This file can be sent to a graphics device or to other software packages.

gray scale

a color-coding scheme that specifies a color in terms of gray components. Gray-scale color codes are commonly used with some laser printers and PostScript devices.

grid request

in the G3GRID procedure, the request specified in a GRID statement that identifies the horizontal variables that identify the x - y plane and one or more z variables for the interpolation.

group variable

a variable in the input data set used to categorize chart variable values into groups.

GSF

See graphics stream file (GSF).

HLS

a color-coding scheme that specifies a color in terms of its hue, lightness, and saturation components. Hue is the color, lightness is the percentage of white, and saturation is the attribute of a color that determines its relative strength and its departure from gray. Lightness and saturation added to the hue produce a specific shade. See also CMYK, HSV, and RGB.

HSV (or HSB)

a color-coding scheme that specifies a color in terms of its hue, saturation, and value (or brightness) components. Hue is the color, saturation is the attribute of a color that determines its relative strength and its departure from gray, and value or brightness is its departure from black. See also CMYK, HLS, and RGB.

identification variable

a variable common to both the traditional map data set and the response data set that the ID statement of the GMAP procedure uses to associate each pair of map coordinates and each response value with a unique map area. See also response variable and traditional map data set.

import

(1) to read a computer graphics metafile (CGM) and store the graphics output in a SAS catalog. Use the GIMPORT procedure to import the CGM. (2) to restore a SAS transport file to its original form (a SAS data library, a SAS catalog, or a SAS data set) in the format appropriate to the host operating system. Use the CIMPORT procedure to import a SAS transport file created by the CPORT procedure.

independent variable

a variable that does not depend on the value of another variable; in a two-dimensional plot, the independent variable is usually plotted on the x (horizontal) axis.

interpolate

to estimate values between two or more known values.

justify

to position text in relation to the left or right margin or the center of the line.

key map

a SAS catalog entry used to translate the codes generated by the keys on a keyboard into their corresponding SAS/GRAPH internal character encoding. See also device map.

label

(1) in the AXIS and LEGEND statements and GPLOT and GCHART procedures, the text that names the variable associated with an axis, a legend, or a bubble in a bubble plot. By default, this text is the name of a variable or of a label previously assigned with a LABEL statement. The text of a label also can be specified with the LABEL= option. (2) in special cases of pie charts and star charts in the GCHART procedure, the midpoint value and the value of the chart statistic for a slice or spine. (3) in the Annotate facility, the text displayed by the LABEL function or macro.

latitude

the angular measure between the equator and the circle of parallel on which a point lies.

legend

refers collectively to the legend value, the legend value description, the legend label, and the legend frame.

libref

the name temporarily associated with a SAS data library. For example, in the name SASUSERS.ACCOUNTS, the name SASUSER is the libref. You assign a libref with a LIBNAME statement or with operating system control language. See also first-level name.

longitude

the angular measure between the reference meridian and the plane intersecting both poles and a point. The reference meridian, called the prime meridian, is assigned a

longitude of 0, and other longitude values are measured from there in appropriate angular units (degrees or radians, for example).

major tick marks

the points on an axis that mark the major divisions of the axis scale. See also minor tick marks.

map

a graphic representation of an area, often a geographic area, but also any other area of any size. See also device map and key map.

map area

a polygon or group of polygons on a map, for example, a state, province, or country. In a traditional map data set, a map area consists of all the observations with the same values for the identification variable or variables. In a feature table, each unique geo-variable value is a map area. Map areas are also called unit areas. See also geo-variable, identification variable, map data set, feature table, and traditional map data set.

map data set

a SAS data set that contains the spatial information the GMAP procedure uses to draw a map. See also feature table and traditional map data set.

meridian

an imaginary circle of constant longitude around the surface of the earth perpendicular to the equator. See also parallel.

midpoint

a value that represents one data value or the middle of a range of data values. When a midpoint represents a range of values, the algorithm used to calculate it depends on the procedure.

minor tick marks

the divisions of the axis scale that fall between major tick marks. See also major tick marks.

needle plot

a plot in which a vertical line connects each data point to the horizontal axis (two dimensions) or the horizontal plane (three dimensions).

numeric variable

a variable that can contain only numeric values. By default, the SAS System stores all numeric variables in floating-point representation.

observation

a row in a SAS data set. An observation is a collection of data values associated with a single entity, such as a customer or state. Each observation contains one data value for each variable. See also variable.

offset

(1) in a legend, the distance between the edge of the legend or the edge of the legend frame and the axis frame or the border surrounding the graphics output area. (2) on an axis, the distance from the origin to either the first major tick mark or the midpoint of the first bar, or the distance from the last major tickmark or the midpoint of the last bar to the end of the axis.

origin

(1) in a three-dimensional coordinate system, the point at which the X, Y, and Z axes intersect, defined by the coordinates (0,0,0). In a two-dimensional coordinate system, the point at which the X and Y axes intersect, defined by the coordinates (0,0). (2) in the AXIS statement, the origin is the point at which the axis line begins (the left end

of the horizontal axis or the bottom of the vertical axis). In the LEGEND statement, the origin is the location of the lower-left corner of the legend. (3) in the graphics output area, the lower-left corner.

palette

the range of colors that can be generated on a graphics device. See also colors list.

panel

in the GREPLAY procedure, a part of the template in which one or more pictures can be displayed. A template can contain one or more panels.

parallel

an imaginary circle of constant latitude around the surface of the earth parallel to the equator. See also meridian.

pattern type

the set of fill patterns that are valid for a particular type of graph. The PATTERN statement supports three pattern types: bar and block patterns, map and plot patterns, and pie and star patterns. See also fill pattern.

pen mounts

on a pen plotter, the holders for the drawing pens.

pie chart

a chart made up of a circle divided by radial lines used to display the relative contribution of each part to the whole.

plot

a graph showing the relationship between variables. The coordinates of each point on the graph represent the values you plot. See also coordinates.

plot line

the line joining the data points in a plot.

plotter

a class of graphics devices that typically use pens to draw hardcopy output.

polygon

a closed, geometric figure bounded by lines or arcs. Polygons can be filled in to represent a surface.

polygon font

a font in which the characters are drawn with enclosed areas that can be filled or empty. See also stroked font.

prism map

a three-dimensional map that uses prisms (polyhedrons with two parallel surfaces) of varying height to indicate the ordinal magnitude of a response variable.

procedure output area

the portion of the graphics output area where the output from a graphics procedure is displayed. See also graphics output area and data area.

projection

a two-dimensional map representation of unit areas on the surface of a sphere, for example, geographic regions on the surface of the Earth.

regression analysis

an analysis of the nature of the relationship between two or more variables, expressed as a mathematical function. On a scatter plot, this relationship is diagrammed as a line drawn through data points. A straight line indicates simple regression; a curve indicates a higher-order regression.

relative coordinates

the coordinates measured from a point other than the origin, usually the endpoint of the last graphics element drawn. See also absolute coordinates.

relative move

a move that repositions the graphics element by a specified distance from its current location. See also absolute move.

replay

to display graphics output that is stored in a catalog entry.

response data set

a SAS data set the GMAP procedure uses that contains data values associated with map areas and one or more identification variables. See also identification variable, response values, and response variable.

response levels

the individual values or ranges of values into which the GMAP or GCHART procedure divides the response variable. See also midpoint.

response values

values of a response variable that the GMAP procedure represents on a map as different pattern/color combinations, or as raised map areas (prisms), spikes, or blocks of different heights. The GCHART procedure represents response values as bars, slices, spines, or blocks. See also midpoint.

response variable

the SAS data set variable in a response data set the GMAP procedure uses that contains data values associated with a map area. Response variables used by the GCHART procedure contain data values associated with bars, slices, spines, or blocks. See also chart variable, response data set, response levels, and response values.

RGB

a color-coding scheme that specifies a color in terms of levels of red, green, and blue components. The levels of each component range from 0 to 255. See also CMYK, HLS, and HSV.

rotate

in the graphics editor, to turn a graphics object about its axis.

RUN group

in SAS procedures, a set of statements ending with a RUN statement.

SAS catalog

a SAS file that stores many different kinds of information in smaller units called catalog entries. A single SAS catalog can contain several different types of catalog entries.

SAS catalog entry

a separate storage unit within a SAS catalog. Each entry has an entry type that identifies its purpose to the SAS System. See also entry type.

SAS data library

a collection of one or more SAS files that are recognized by the SAS System and that are referenced and stored as a unit. Each file is a member of the library.

SAS data set

descriptor information and its related data values organized as a table of observations and variables that can be processed by the SAS System. A SAS data set can be either a SAS data file or a SAS data view.

SAS date value

an integer representing a date in the SAS System. The integer represents the number of days between January 1, 1960, and another specified date. (For example, the SAS date value 366 represents the calendar date January 1, 1961.)

SAS Display Manager System

an interactive, windowing interface to SAS System software. Display manager commands can be issued by typing them on the command line, pressing function keys, or selecting items from the PMENU facility. Within one session, many different tasks can be accomplished, including preparing and submitting programs, viewing and printing results, and debugging and resubmitting programs.

scatter plot

a two- or three-dimensional plot showing the joint variation of two (or three) variables from a group of observations. The coordinates of each point in the plot correspond to the data values for a single observation.

segment

in the GMAP procedure, a polygon that is a part of a unit area consisting of more than one polygon. For example, consider a map of Hawaii. The representation of the single unit area (the state) consists of a group of individual segments (the islands), each of which is a separate polygon. In the GFONT procedure, a segment is a single continuous line that forms part of all of a character or symbol.

software font

See font, software.

spine

a line on a star chart used to represent the relative value of the chart statistic for a midpoint. Spines are drawn outward from the center of the chart.

spline

a method of interpolation in which a smooth line or surface connects data points.

standard deviation

a statistical measure of the variability of a group of data values. This measure, which is the most widely used measure of the dispersion of a frequency distribution, is equal to the positive square root of the variance.

string

See character string.

stroked font

a font in which the characters are drawn with discrete line segments or circular arcs. See also polygon font.

subgroup variable

the variable in the input data set for a chart that is used to proportionally fill areas of the bars or blocks on the chart.

summary variable

a variable in an input data set whose values the GCHART procedure totals or averages to produce the sum or mean statistics, respectively.

surface map

a three-dimensional map that uses spikes of varying heights to indicate levels of relative magnitude.

surface plot

a three-dimensional graph that displays a grid-like surface formed by the values of the vertical (*Z*) variable plotted on a plane specified by the *X* and *Y* variables.

template

in the GREPLAY procedure, a framework that enables you to display one or more pictures on a page.

text string

See character string.

tilt angle

the measure in degrees from the horizontal axis to the major axis of an object.

traditional map data set

a SAS data set that contains variables whose values are x, y coordinates that define points that are the boundaries of map areas, such as states or counties. Each observation also contains an identification variable whose value identifies the map area to which the point belongs. See also identification variable, map area, map data set, and feature table.

type style

a typeface design and its variations, for example, Swiss, Swiss Bold, and Swiss Italic. See also font.

unit area

See map area.

user-definable colors

the colors that can be defined using SAS color names, or CMYK (cyan, magenta, yellow, black), RGB (red, green, blue), HLS (hue, lightness, saturation), HSV (hue, saturation, value), or gray-scale color equivalents.

value

the text that labels a major tick mark on an axis. Also, in a legend, a value is a line, bar, or shape that the legend explains.

variable

a column in a SAS data set. A variable is a set of data values that describe a given characteristic across all observations. See also macro variable.

variable type

the classification of a variable as either numeric or character. Type is an attribute of SAS variables.

WORK data library

the SAS data library automatically defined by the SAS System at the beginning of each SAS session or SAS job. It contains SAS files that are temporary by default. When the libref USER is not defined, the SAS System uses WORK as the default library for SAS files created with one-level names.

X axis

in a two-dimensional plot, the horizontal axis. In a three-dimensional plot, the X axis is the axis perpendicular to the Y-Z plane.

Y axis

in a two-dimensional plot, the vertical axis. In a three-dimensional plot, the Y axis is the axis perpendicular to the X-Z plane.

Z axis

in a three-dimensional plot, the axis perpendicular to the X-Y plane.

Index

(pound sign), variables as plot point labels 198
 ? statement, GREPLAY procedure 1246

Numbers

3D charts
 3D bar charts 130
 3D pie charts 6

A

A= option
 AXIS statement 136, 225
 LABEL= option, DONUT statement 829
 TITLE, FOOTNOTE, and NOTE statements 213, 225
 A option, GOPTIONS procedure 1077
 access permissions, browsers 582
 accuracy of color representation 105
 ACROSS= option
 CHART statement, GRADAR procedure 1187, 1201
 LEGEND statement options 152
 PIE and DONUT statements 820
 STAR statement 835
 ACROSSVAR= option, CHART statement 1187
 ACTION= macro argument 546
 action statements 26
 ActiveX Control 370, 385, 387
 authentication 583
 drill-down links 392
 drill-down tags 412
 embedded graphics in Microsoft Word 393, 395
 generating output for 391
 installing and uninstalling 389
 interactive contour plots 394
 internationalization 392
 ActiveX control file (.exe file)
 location of 422
 ACTIVEX device driver 381, 388
 data tips for 568
 drill-down links in presentations 571
 ActiveX parameters and attributes 421
 ActiveX support 1508
 Annotate functions 1539
 AXIS statement 1508
 G3D procedure 1537

GAREABAR procedure 1518
 GBARLINE procedure 1519
 GCHART procedure 1521
 GCONTOUR procedure 1526
 GMAP procedure 1527
 GOPTIONS statement 1510
 GPLOT procedure 1530
 GRADAR procedure 1535
 LEGEND statement 1514
 PATTERN statement 1515
 SYMBOL statement 1516
 TITLE and FOOTNOTE statements 1517
 ACTXIMG device driver 377, 381, 388
 data tips for 568
 drill-down links in images 571
 GIF, JPEG, PNG vs. 440
 ODS with 447
 Web presentations, developing 442
 ADD statement, GDEVICE procedure 921
 ADMGDF option 262
 Africa, creating outline map of 1232
 AFTER option, MOVE statement 1255
 AHUNITS= macro argument 536
 Albers' projections 1165
 clipping map areas 1178
 default projection specifications 1174
 projection criteria 1172
 when to use 1172
 ALIGN= macro argument 536
 alignment
 axis labels 127, 133, 135, 137
 axis values 137
 character cells 268
 legend labels 153
 legend text 158, 159
 legend values 157
 legends 155, 162
 plot print labels 197
 text in graphics output 218
 ALL option, GMAP procedure 1008
 ALL option, GMAP statement 144
 ALT= macro argument 536
 alternative hardware fonts 80
 AMBIENT= parameter, JAVA and ActiveX 427
 anchors 168, 259
 angle, rotation
 angling text in pie charts 1381
 axis labels 127, 133, 135, 136
 donut chart labels 829, 830
 hardware text rotation 330

landscape orientation of graphics output
 area 34, 322, 323, 349
 orientation of graphics output area 34
 portrait orientation of graphics output
 area 34, 332, 344, 349
 print orientation 349
 printing orientation 349
 rotating and tilting surface map 1068
 surface and scatter plots 1299, 1316, 1323
 text in graphics output 213, 219, 222
 text in pie charts 1381
 ANGLE= macro argument 546
 ANGLE= option
 AXIS statement 136, 225
 LABEL= option, DONUT statement 829
 PIE and DONUT statements 820
 STAR statement 835
 TITLE, FOOTNOTE, and NOTE statements 213, 225
 ANGLE variable, Annotate facility 642
 animation 378, 457
 creating sequences of 458
 delay between graphs 278
 graphics options for 459
 repeating as loop 321
 sample programs 459, 463
 ANNO= option
 BAR statement 753
 BLOCK statement, GCHART procedure 788
 BLOCK statement, GMAP procedure 1011
 BUBBLE statement 1092
 CHART statement, GRADAR procedure 1187
 CHORO statement 1018
 G3D procedure 1300
 GANNO procedure 708
 GBARLINE procedure 750
 GCHART procedure 786
 GCONTOUR procedure 888
 GMAP procedure 1008
 GPLOT procedure 1088
 GPRINT procedure 1149
 GRADAR procedure 1185
 GSLIDE procedure 1279, 1283
 HBAR and VBAR statements 799
 PIE and DONUT statements 820
 PLOT statement, G3D procedure 1302
 PLOT statement, GCONTOUR procedure 891
 PLOT statement, GPLOT procedure 1104
 PRISM statement 1024
 SCATTER statement, G3D procedure 1307

- STAR statement 835
 - SURFACE statement 1031
 - %ANNOMAC macro, Annotate facility 679
 - Annotate data sets 587, 599
 - applying to web output 500
 - missing values 601
 - observation and structure of 589
 - producing graphics output 601
 - annotate facility
 - examples 597
 - Annotate facility 14, 588, 614
 - ActiveX and Java support for 1539
 - Annotate graphics in drill-down graphs 719
 - coordinates 596
 - debugging 604
 - drill-down links, generating 500
 - DSGI vs. 1354
 - error messages, list of 699
 - functions 594, 615
 - graphic elements and formatting 595
 - images, displaying 118
 - in text slides 1278, 1283
 - internal coordinates 678
 - macro data sets 23
 - macros, how to use 697
 - macros for 679, 697
 - processing details 602
 - producing multiple graphs 715
 - projecting an Annotate data set 1180
 - scaling data-dependent output 710
 - scaling graphs 710
 - storing Annotate graphics 713
 - variables 591, 599, 602, 642
 - Web output, generating 499
 - Annotate macros 600
 - ANNOTATE= option 500, 601
 - BAR statement 753
 - BLOCK statement, GCHART procedure 788
 - BLOCK statement, GMAP procedure 1011
 - BUBBLE statement 1092
 - CHART statement, GRADAR procedure 1187
 - CHORO statement 1018
 - G3D procedure 1300
 - GANNO procedure 708
 - GBARLINE procedure 750
 - GCHART procedure 786
 - GCONTOUR procedure 888
 - GMAP procedure 1008
 - GPLOT procedure 1088
 - GPRINT procedure 1149
 - GRADAR procedure 1185
 - GSLIDE procedure 1279, 1283
 - HBAR and VBAR statements 799
 - PIE and DONUT statements 820
 - PLOT statement, G3D procedure 1302
 - PLOT statement, GCONTOUR procedure 891
 - PLOT statement, GPLOT procedure 1104
 - PRISM statement 1024
 - PROC statement 145
 - SCATTER statement, G3D procedure 1307
 - STAR statement 835
 - SURFACE statement 1031
 - Any mode drill-down mode, Java 401
 - APPLET element (HTML), macro arguments for 536
 - APPLETLOC= system option 399, 423
 - arc-drawing capability, device 270
 - ARC function (DSGI) 1446
 - ARCHIVE= macro argument 536
 - ARCHIVE= option 422
 - arcs
 - drawing with Annotate facility 631, 632
 - drawing with DSGI 1446
 - writing in, DSGI for 1455
 - area bar charts 725
 - ActiveX and Java support for 1518
 - chart with numeric category variable 731
 - chart with subgrouping 733
 - chart with subgrouping and variable percentages 735
 - simple area bar chart 729
 - syntax and options 727
 - area boundaries, unmatched
 - GREDUCE procedure and 1215
 - REMOVE procedure and 1215
 - AREA element (HTML) 260
 - AREA= option
 - BLOCK statement 1011
 - PRISM statement 1024
 - areas, Annotate graphics 596
 - AREAS= option, PLOT statement 1104, 1134
 - array of arguments, callback method 408
 - ASCENDING option
 - BAR statement 753
 - HBAR and VBAR statements 799
 - PIE and DONUT statements 820
 - PLOT statement 766
 - ASCII-to-EBCDIC translation 358
 - ASF function (DSGI) 1405, 1464
 - ASIS option, GPROJECT procedure 1168
 - ASPECT function (DSGI) 1407, 1465
 - ASPECT= option 263
 - aspect ratio 263
 - maintaining for cells 1152
 - asymmetrical keymaps and device maps 986
 - attributes, JAVA and ActiveX parameters and attributes 421
 - ATTRIBUTES= option, ODS statements 421
 - audience for presentations, considering 380
 - AUTOCOPY option 263
 - AUTOFEED option 264
 - AUTOHREF option
 - BUBBLE statement 1092
 - PLOT statement, GCONTOUR procedure 891
 - PLOT statement, GPLOT procedure 1105
 - AUTOLABEL and AUTOLABEL= options,
 - PLOT statement 891, 903, 906
 - automatic paper feed 264, 327
 - AUTOREF option
 - BAR statement 753
 - HBAR and VBAR statements 799
 - AUTOREF= option, AXIS statement options 136
 - AUTOSIZE= graphics option 265
 - AUTOVREF option
 - BUBBLE statement 1092
 - PLOT statement, GCONTOUR procedure 891
 - PLOT statement, GPLOT procedure 1105
 - AWUNITS= macro argument 541
 - axes 124
 - axis definitions 1193
 - colors, bar line charts 765
 - colors, block charts 795
 - colors, CAXIS= option for 753, 789, 799, 892,,
 - contour plots 886, 901
 - labels 127, 133
 - line type 134
 - logarithmic 127, 229, 815, 1087
 - offset 130
 - origins 133
 - plots with two vertical axes 1084, 1119, 1124
 - scatter plots, reversing 1312
 - splines in star charts 1193, 1210
 - suppressing, NOAXES option for 897, 1097, 1111, 130
 - suppressing, NOAXIS option for 760, 810, 897, 1097,
 - surface and scatter plots 1299
 - tick marks 129
 - tick marks, ordering 226
 - AXIS= option
 - BAR statement 753, 761
 - HBAR and VBAR statements 812
 - PLOT statement 766, 767
 - AXIS option, GOPTIONS procedure 1077
 - AXIS statement 27, 123, 124
 - ActiveX and Java support for 1508
 - GCONTOUR procedure 886
 - logarithmic axes 229
 - ordering datetime tick marks 226
 - AXIS1= and AXIS2= options, GRID statement 1334, 1336
- ## B
- B= option, GFONT procedure 947
 - BACKGROUND= parameter, JAVA 427
 - background color
 - graphics output area 266
 - image transparency 357
 - legends 152
 - text in graphics output 215
 - background images 113, 318, 319
 - BACKGROUNDCOLOR= parameter, Metaview Applet 475
 - backplane images 115
 - %BAR, %BAR2 macro, Annotate facility 680
 - bar charts 5
 - 3D plane 130
 - basics 775
 - drill-down functionality in 856
 - error bars in horizontal bar chart 854
 - group brackets on axis 130
 - images on bars of 116
 - midpoints and statistics in horizontal bar chart 851
 - patterns, outlines, and colors 171, 816
 - subgroup labels 607
 - subgrouping in pie or donut chart 848
 - subgrouping in vertical bar chart 848
 - sum statistic, specifying (example) 846
 - vocabulary of 778
 - with Web drill-down (example) 255
 - BAR function, Annotate facility 616, 1539
 - BAR function, DSGI 1448
 - bar line charts 739, 796
 - ActiveX and Java support for 1519
 - BAR statement, GBARLINE procedure 751

- basic graph with styles (example) 768
 - colors and images 764
 - interpolation methods 740
 - missing values 747
 - parts of 741
 - patterns and outlines 748, 764
 - plot overlays 765
 - PLOT statement, GBARLINE procedure 765
 - statistics in, displaying 763
 - SYMBOL statement, GBARLINE procedure 768
 - symbols 768
 - syntax and options 749
 - BAR statement, GBARLINE procedure 751, 1519
 - bar statistics 741, 763
 - bar variables 741, 742, 745
 - bars, drawing with DSGI 1448
 - baseline, font 940, 947
 - baseline, text in graphics output 213, 219
 - rotating characters from 222
 - underlining 223
 - BASELINE= option, GFONT procedure 947
 - batch environment, GDEVICE procedure in 918
 - switching to 925, 1252
 - batch mode 32
 - BC= option, TITLE, FOOTNOTE, and NOTE statements 215, 225
 - BCOLOR= option
 - BUBBLE statement 1092
 - TITLE, FOOTNOTE, and NOTE statements 215, 225
 - BDCLASS= macro argument 552
 - BEFORE option, MOVE statement 1255
 - BFONT= option, BUBBLE statement 1092
 - BG= macro argument 552
 - BGTYPE= macro argument 552
 - BINDING= option 265
 - Bitstream fonts, rendering 82, 293, 346
 - spacing between letters 295
 - bivariate interpolation 1330
 - BL= option, TITLE, FOOTNOTE, and NOTE statements 225
 - BLABEL option, BUBBLE statement 1092
 - black and white, reversing 353
 - BLANK= option, TITLE, FOOTNOTE, and NOTE statements 215, 225
 - blank spaces, removing from data values 414
 - block charts 4
 - basics 774
 - BLOCK statement syntax 787
 - grouping and subgrouping in (example) 844
 - negative and zero values 795
 - patterns, outlines, and colors 794
 - sum statistic, specifying (example) 842
 - text in 795
 - block effects for legends 152, 163
 - block maps 11, 996
 - assigning formats to response variables (example) 1049
 - drill-down functionality in maps (example) 1054
 - identification variables 1005
 - patterns 1016
 - predefined formats for 1035
 - producing simple block map (example) 1045
 - response levels 1004
 - specifying response levels (example) 1047
 - syntax and options 1010
 - BLOCK statement, GCHART procedure
 - ActiveX and Java support for 1521
 - block chart with sum statistic (example) 842
 - syntax and options 787
 - BLOCK statement, GMAP procedure 1010
 - ActiveX and Java support for 1527
 - assigning formats to response variables (example) 1049
 - drill-down functionality in maps (example) 1054
 - producing simple block map (example) 1045
 - specifying response levels (example) 1047
 - BLOCKMAX= option, BLOCK statement 144, 789
 - BLOCKSIZE= option, BLOCK statement 1012
 - BO= option, TITLE, FOOTNOTE, and NOTE statements 215, 225
 - body, animation 458
 - BODY= argument, ODS HTML statement 165
 - body files for graphics output 491
 - BODY= option, ODS HTML statement 491
 - BORDER= graphics option 266
 - BORDER= macro argument 546
 - BORDER option, GSLIDE procedure 1279, 1281
 - borders
 - Annotate facility to draw 622
 - graphics output area 266, 1281
 - legends 152
 - BOX= option, TITLE, FOOTNOTE, and NOTE statements 215, 225
 - box plots 185, 187
 - creating and modifying (example) 233
 - line width 201
 - boxes around graphics output text 215, 216
 - brackets, bar charts 130
 - BROWSE option, GDEVICE procedure 920
 - browser permissions 582
 - BRITITLE= macro argument 553
 - BS= option, TITLE, FOOTNOTE, and NOTE statements 216, 225
 - BSCALE= option, BUBBLE statement 1092
 - BSIZE= option, BUBBLE statement 1093
 - BSPACE= option, TITLE, FOOTNOTE, and NOTE statements 216, 225
 - bubble plots 8, 1083, 1090
 - adding right vertical axis (example) 1124
 - controlling bubble display 1098
 - coordinating BUBBLE and BUBBLE2 plots requests 1100
 - generating simple bubble plots (example) 1121
 - labeling and sizing plot bubbles (example) 1122
 - BUBBLE statement, GPLOT procedure 1090
 - ActiveX and Java support for 1530
 - controlling bubble display 1098
 - coordinating with BUBBLE2 statements 1100
 - generating simple bubble plots (example) 1121
 - labeling and sizing plot bubbles (example) 1122
 - BUBBLE2 statement, GPLOT procedure 1098
 - ActiveX and Java support for 1530
 - adding right vertical axis (example) 1124
 - coordinating with BUBBLE statements 1100
 - coordinating with BUBBLE statements 1100
 - bundling attributes, DSGI 1373
 - BWIDTH= option, SYMBOL statement 185
 - BY lines 143
 - BY statement 26, 124, 141
 - Annotate facility with 603
 - color of BY lines 267
 - displaying with (suppressing from) catalog entries 1247, 1256
 - fonts FOR BY lines 289
 - generating chart series (example) 240
 - REMOVE procedure 1227
 - height of BY lines 312
 - RUN-group processing 33
 - #BYLINE option, text string specifications 222
 - BYLINE statement, GREPLAY procedure 1243, 1247
 - #BYVAL option, text string specifications 223
 - #BYVAR option, text string specifications 223, 226
- ## C
- C= option
 - AXIS statement options 126, 136, 139
 - BUBBLE statement 1094
 - GFONT procedure 947
 - LABEL= option, DONUT statement 829
 - LEGEND statement options 158
 - PLOT statement, GPLOT procedure 1106
 - POINTLABEL= specification 197
 - SYMBOL statement 185, 206
 - TITLE, FOOTNOTE, and NOTE statements 216
 - CA= option
 - BUBBLE statement 1093
 - CHART statement, GRADAR procedure 1187
 - PLOT statement, GPLOT procedure 1106
 - callback method, JavaScript 407
 - Canada
 - census division data 1003, 1033
 - reducing map of (example) 1220
 - capline, font 940, 947
 - CAPLINE= option, GFONT procedure 947
 - CAPTURE= argument, META2HTM macro 564
 - carriage return at record ends 300
 - Cartesian coordinates, projecting spherical coordinates into 1161
 - basic usage of GPROJECT procedure 1172
 - clipping map areas (example) 1178
 - default projection specifications, using (example) 1174
 - emphasizing map areas (example) 1177
 - ID statement, GPROJECT procedure 1172
 - input map data sets 1163
 - projecting an Annotate data set (example) 1180
 - syntax and options, GPROJECT procedure 1168
 - types of map projections 1165
 - Cartographic font 87
 - CAT command 55
 - catalog entries 1239
 - BY line 143
 - changing information about 1254
 - copying or duplicating 1251, 1263

- deleting from catalogs 1251
- displaying BY statement information with 1247
- duplicate entry names 1240
- grouping 1253
- managing 1267
- names for 63
- names for image output files 445
- printing 1254
- replacing 70
- replaying 1268
- selecting for replay 1257
- CATALOG function (DSGI) 1407, 1465
- CATALOG= option, GDEVICE procedure 921
- category variables 726
- CATEXT= macro argument 546
- CAUTOHREF= option
 - BUBBLE statement 1093
 - PLOT statement, GCONTOUR procedure 891
 - PLOT statement, GPLOT procedure 1106
- CAUTOREF= option
 - BAR statement 753
 - HBAR and VBAR statements 799
- CAUTOVREF= option
 - BUBBLE statement 1093
 - PLOT statement, GCONTOUR procedure 892
 - PLOT statement, GPLOT procedure 1106
- CAXES= option, CHART statement 1187
- CAXIS= option
 - BAR statement 753
 - BLOCK statement 789
 - BUBBLE statement 1093
 - CHART statement 1187
 - HBAR and VBAR statements 799
 - PLOT statement, G3D procedure 1302
 - PLOT statement, GCONTOUR procedure 892
 - PLOT statement, GPLOT procedure 1106
 - SCATTER statement, G3D procedure 1307
- CBACK= argument, META2HTM macro 565
- CBACK function (DSGI) 1408, 1466
- CBACK= macro argument 546, 562
- CBACK= option 266
- CBLKOUT= option, BLOCK statement 1012
- CBLOCK= option, LEGEND statement 152, 163
- CBODY= option, SURFACE statement 1031
- CBORDER= option, LEGEND statement options 152
- CBORDER variable, Annotate facility 643
- CBOTTOM= option, PLOT statement 1302
- CBOX variable, Annotate facility 644
- CBY= graphics option 267
- CC option
 - ? statement, GREPLAY procedure 1246
 - LIST statement, GREPLAY procedure 1254
- CC= option, GREPLAY procedure 1243
- CC statement, GREPLAY procedure 1247
- CCOPY statement, GREPLAY procedure 1248
- CDEF statement, GREPLAY procedure 1249, 1274
- CDELETE statement, GREPLAY procedure 1250
- CELL option 268
- CEMPTY= option
 - BLOCK statement 1012
 - CHORO statement 1019
 - PRISM statement 1025
- census division data, Canada 1003, 1033
- CENTER= macro argument 553
- CENTIMETERS option, GOPTIONS procedure 1077
- %CENTROID macro, Annotate facility 680
- CERROR= option
 - BAR statement 753
 - HBAR and VBAR statements 800
- CFILL= option
 - PIE and DONUT statements 821
 - STAR statement 835
- CFR= option
 - BAR statement 754
 - CHART statement, GRADAR procedure 1187
 - HBAR and VBAR statements 728, 800
 - PLOT statement, GCONTOUR procedure 892
 - PLOT statement, GPLOT procedure 1106
- CFRAME= option
 - BAR statement 754
 - CHART statement, GRADAR procedure 1187, 1208
 - GSLIDE procedure 1279, 1281
 - HBAR and VBAR statements 800
 - LEGEND statement options 152
 - PLOT statement, GCONTOUR procedure 892
 - PLOT statement, GPLOT procedure 1106
- CFRAMESIZE= option, CHART statement 1187
- CFRAMETOP= option, CHART statement 1188
- CFREQ option
 - BAR statement 754
 - HBAR and VBAR statements 800
- CFREQLABEL= option, HBAR and VBAR statements 800
- CGMs (computer graphics metafiles)
 - adjusting graphics output (example) 979
 - creating and importing (example) 977
 - elements not supported 971
 - font substitutions 974
- CH= option
 - BUBBLE statement 1093
 - PLOT statement, GCONTOUR procedure 892
 - PLOT statement, GPLOT procedure 1106
- CHANDLE= macro argument 546
- CHAR variable (font data sets) 952
- CHAR1, CHAR2 variables (kern data sets) 959
- character bar variables 742
- character cells 36
 - alignment 268
 - as units of measurement 38
 - aspect ratio of 1152
 - size of 265
- character chart variables 780
- character codes 81
- character codes, displaying 946, 950, 962
- character transcoding 561
- characters
 - adjusting character size in output (example) 1156
 - as axis values 131
 - as legend values 155
 - HTML entities 582
 - prefixing output records 309
 - prompts 343
 - special plot symbols 200
- characters, mapping to keyboard 983
- asymmetrical 986
- basics of 983
- creating and using 990
- GKEYMAP data sets 987
- GKEYMAP data sets, generating 990
- ignoring 945, 949
- modifying (example) 990
- CHARACTERS option 268
- CHARREC= option, GDEVICE procedure 269
- CHARSET= macro argument 561
- CHARSPACETYPE= option, GFONT procedure 948
- CHART statement, GRADAR procedure
 - ActiveX and Java support for 1535
 - assigning axis definitions to axis spokes (example) 1210
 - changing star type in radar charts (example) 1207
 - color and line styles in radar charts (example) 1208
 - filling stars in radar charts (example) 1204
 - images in radar charts (example) 1205
 - multiple classification variables in radar charts (example) 1202
 - overlapping radar charts (example) 1199
 - producing basic radar chart (example) 1198
 - specifying mode for radar charts (example) 1209
 - syntax and options 1185
 - tiling radar charts (example) 1201
- chart statistics
 - GBARLINE procedure 745, 778
 - GCHART procedure 782, 815
 - horizontal bar chart (example) 851
- chart variables 778, 779, 780
- charts 4
 - drill-down functionality, ActiveX 392
- CHARTYPE= graphics option 79, 270
- CHARTYPE variable (GKEYMAP data set) 987
- Chartype window (GDEVICE) 931
- CHECK= suboption, AUTOLABEL= option 898
- CHORO statement, GMAP procedure 1017
 - ActiveX and Java support for 1527
 - creating maps using feature tables (example) 1069
 - drill-down functionality in maps (example) 1054
 - labeling U.S. states in choropleth map (example) 1061
 - producing simple choropleth map (example) 1053
 - projecting an Annotate data set (example) 1180
- choropleth maps 11, 997
 - creating maps using feature tables (example) 1069
 - drill-down functionality in maps (example) 1054
 - identification variables 1005
 - labeling U.S. states in choropleth map (example) 1061
 - predefined formats for 1035
 - producing simple choropleth map (example) 1053
 - response levels 1004
 - syntax and options 1017

- CHREF= option
 - BUBBLE statement 1093
 - PLOT statement, GCONTOUR procedure 892
 - PLOT statement, GPLOT procedure 1106
- CHUB= macro argument, DS2CSF macro 562
- CI= option, SYMBOL statement 185, 206
- CIMPORT procedure 56
- CINDIC= macro argument, DS2CSF macro 562
- circle-drawing capability, device 270
- %CIRCLE macro, Annotate facility 681
- circle of stars, drawing (example) 609
- CIRCLEARC option 270
- circles, writing in (DSGI) 1455
- city map data (U.S.) 1003
- CLABTXT= macro argument, DS2CSF macro 562
- CLABVAL= macro argument, DS2CSF macro 562
- classification variables, plotting 1083
- CLASSPATH environmental variables 583
- CLEAR function (DSGI) 1457
- CLEVELS= option, PLOT statement 892, 903
- client graphs vs. server graphs 584
- CLINK= macro argument 546
- CLIP function (DSGI) 1409, 1467
- CLIP options, TDEF statement 1260
- clipped polygons 331, 335
- clipping around viewports (DSGI) 1377
- clipping map data sets 1039, 1173, 1213, 12
 - example of 1178
 - reducing map of Canada (example) 1220
- CLIPREF option
 - BAR statement 754
 - HBAR and VBAR statements 800
- CLIPTIPS= parameter, JAVA 427
- CLM= option
 - BAR statement 754
 - HBAR and VBAR statements 800
- CLOCKWISE option, PIE and DONUT statements 821, 878
- closed destinations, ODS 489
- closing
 - GRAPH window 49
 - GSF (graphics stream file) 290
- CM option, GOPTIONS procedure 1077
- CMAP option
 - ? statement, GREPLAY procedure 1246
 - LIST statement, GREPLAY procedure 1254
- CMAP= option, GREPLAY procedure 271, 1243
- CMAP statement, GREPLAY procedure 1250
- %CMY macro 100
- CMYK color scheme 95
- %CMYK macro 101
- CNODE= macro argument 547
- %CNS macro 101
- CNS (SAS Color Naming Scheme) 97, 99
- CNTL2TXT function, Annotate facility 618
- CO= option, SYMBOL statement 185, 206
- CODEBASE attribute, OBJECT element (HTML) 399, 424
- CODEBASE= macro argument 541
- CODEBASE= option 422
- CODEBASE parameter, APPLETT element (HTML) 399
- CODELEN= option, GFONT procedure 948
- COLINDEX function (DSGI) 1409
- COLLATE option 271
- collating printed output 271
- color codes
 - CMYK scheme 96
 - gray-scale scheme 99
 - HLS scheme 97
 - HSV and HBS schemes 98
 - RGB scheme 95
- color lists 93
 - default 94
 - managing colors list for device driver (example) 1291
- COLOR MAPPING window (GREPLAY) 1266
- color maps 1239
 - copying or duplicating 1248
 - creating 1268, 1274
 - defining or modifying in catalogs 1249
 - deleting from catalogs 1250
 - printing contents of 1254
 - specifying/assigning 271, 1247, 1250
 - transporting 58
- COLOR= option
 - AXIS statement options 126, 136, 139
 - LABEL= option, DONUT statement 829
 - LEGEND statement options 158
 - PATTERN statement 170
 - POINTLABEL= specification 197
 - SCATTER statement, G3D procedure 1307, 1310
 - SYMBOL statement 185, 206
 - TDEF statement, GREPLAY procedure 1260
 - TITLE, FOOTNOTE, and NOTE statements 216
- color schemes 95
- COLOR variable, Annotate facility 645
- %COLORMAC macro 100
- COLORMAP= macro argument 547
- Colormap window (GDEVICE) 931
- COLORNAMELIST= parameter, JAVA 427
- COLORNAMES= parameter, JAVA 427
- colors 92
 - active, number of (plotters) 333
 - axes 126, 136
 - axes, CAXIS= option for 753, 789, 799, 892,,
 - axis labels 126, 127, 133, 135,
 - axis tick marks 126, 129, 139
 - axis values 136
 - bar charts 816
 - bar line charts 764
 - block charts 794
 - borders 276
 - bubble plots 1092
 - BY lines 143, 267
 - contour plot lines and labels 903
 - default, specifying 92, 272
 - donut chart labels 829
 - graphics output area 266
 - image transparency 357
 - legend label 153
 - legend text 158
 - legend values 157
 - legends 152
 - managing colors list for device driver (example) 1291
 - mapping 971
 - maximum display at once 324
 - Netscape troubleshooting 583
- ODS styles 94
 - output text (example) 1153
 - patterns 274
 - pie and donut chart slices 831
 - plot print labels 197
 - plot symbols 93, 185, 206, 231, 2
 - plotting in order of 333
 - reference lines 892, 893, 1093, 1094
 - reversing black and white 353
 - star charts 1188, 1204, 1208
 - text in graphics output 93, 215, 216
 - titles, footnotes, and notes 276, 493
 - utility macros for 100
- COLORS= graphics option 93, 178, 272
- COLORSCHEME= parameter, JAVA and ActiveX 428
- COLORTYPE= option, GDEVICE procedure 273
- COLREP function (DSGI) 1410, 1468
- columns, legends 152
- columns in graphics output area 3, 36, 274, 315, 322
- COMMENT function, Annotate facility 619
- %COMMENT macro, Annotate facility 682
- comments 279
- communications ports, how output is written to 306
- confidence intervals 756, 803, 854
 - error bars in horizontal bar chart (example) 854
 - HBAR and VBAR statements 803
- confidence limits 192
- conformal projections 1166, 1172
- CONSTANT= option, SURFACE statement 1031
- constant-width (uniform) fonts 940, 951
- Constellation applet 373, 513
 - chart with simple arcs (example) 518
 - chart with weighted arcs (example) 520
 - data tips with 569
 - drill-down functionality 572, 573
 - DS2CONST macro with 515
 - hotspots 524
 - when to use 514
 - XML written to external file (example) 522
- CONTENTS= argument, ODS HTML statement 166
- contents files 495
- CONTENTS option, MAPIMPORT procedure 1349
- CONTENTS= option, ODS statements 495
- continent formats for maps 1035
- continuous output stream 296
- continuous paper feed 264, 327
- continuous variables
 - bar variables 742, 744
 - chart variables 779, 781
 - map variables 1004
- Contour applet 372
 - parameters for, list of 424
- contour labels, size of 187
- contour lines
 - colors for 185, 206
 - distance between labels 199
 - fonts 186
 - size of 201
 - type of 196, 207

- contour plots 10, 885
 - ActiveX and Java support for 1526
 - axes 886, 901
 - AXIS statement, GCONTOUR procedure 886
 - contour levels, specifying 895, 899, 908
 - contour levels, specifying (example) 908
 - interactive, with ActiveX (example) 394
 - interpolation methods 887
 - labels for contour lines (example) 906
 - missing values 887, 889
 - modifying lines and labels with SYMBOL statement 903
 - PATTERN statement, GCONTOUR procedure 169
 - patterns 173
 - patterns and joins in contour plots (example) 910
 - patterns and joins in (example) 910
 - PLOT statement, GCONTOUR procedure 889
 - simple, generating (example) 904
 - simple contour plot, generating (example) 904
 - spline interpolation (example) 1343
 - contour variables 885
 - control characters, device 359
 - converting
 - between RGB and HLS colors 103
 - graphics output 56, 59
 - coordinates, projecting from spherical to Cartesian 1161
 - basic usage of GPROJECT procedure 1172
 - clipping map areas (example) 1178
 - default projection specifications, using (example) 1174
 - emphasizing map areas (example) 1177
 - ID statement, GPROJECT procedure 1172
 - input map data sets 1163
 - projecting an Annotate data set (example) 1180
 - syntax and options, GPROJECT procedure 1168
 - types of map projections 1165
 - coordinates and coordinate systems
 - Annotate facility 596
 - data-dependent, GSLIDE with 1281
 - longitude and latitude 1164
 - templates, changing 1259
 - COPY function (DSGI) 1458
 - COPY= option, TDEF statement 1260
 - COPY statement
 - custom device entry, creating (example) 936
 - GDEVICE procedure 924
 - GREPLAY procedure 1251
 - copying
 - catalog entries 1251, 1263
 - color maps 1248
 - numbers of print copies 298
 - templates 1258
 - country formats for maps 1035
 - COUTLINE= macro argument, DS2CSF macro 562
 - COUTLINE= option
 - BAR statement 754
 - BLOCK statement, GCHART procedure 789
 - BLOCK statement, GMAP procedure 1012
 - CHORO statement 1019
 - HBAR and VBAR statements 800
 - patterns 180
 - PIE and DONUT statements 821
 - PLOT statement, GCONTOUR procedure 892
 - PLOT statement, GPLOT procedure 1106
 - PRISM statement 1025
 - STAR statement 835
 - CPATTERN= graphics option 171, 274
 - patterns 179
 - patterns and 180
 - CPCT option
 - BAR statement 754
 - HBAR and VBAR statements 801
 - CPERCENT option
 - BAR statement 754
 - HBAR and VBAR statements 801
 - CPERCENTLABEL= option, HBAR and VBAR statements 801
 - CPORT procedure 56
 - CR= option
 - BAR statement 754
 - HBAR and VBAR statements 801
 - CREATE_ID_, MAPIMPORT procedure 1349
 - creating graphs interactively 395
 - CREF= option
 - BAR statement 754
 - HBAR and VBAR statements 801
 - critical success factor (CSF) diagrams
 - drill-down functionality 573
 - DS2CSF macro 527, 528
 - hotspots 532
 - sample diagrams 530
 - CSELECT= macro argument 547
 - CSF diagrams
 - drill-down functionality 573
 - DS2CSF macro 527, 528
 - hotspots 532
 - sample diagrams 530
 - CSFTYPE= macro argument, DS2CSF macro 562
 - CSHADOW= option, LEGEND statement options 153, 163
 - CSP= option, GFONT procedure 948
 - CSPOKE= and CSPOKES= options, CHART statement 1188
 - CSTAR= option, CHART statement 1188
 - CSTARCIRCLE= and CSTARCIRCLES= options, CHART statement 1188
 - CSTARFILL= option, CHART statement 1188, 1204
 - CSTARS= option, CHART statement 1188, 1208
 - CSYMBOL= graphics option 207, 275
 - CT=, GFONT procedure 944
 - CTEXT= macro argument 553
 - CTEXT= option
 - BAR statement 755
 - BLOCK statement, GCHART procedure 789
 - BLOCK statement, GMAP procedure 1013
 - BUBBLE statement 1094
 - CHART statement, GRADAR procedure 1189
 - CHORO statement 1019
 - GFONT procedure 944
 - GPRINT procedure 1149, 1153
 - HBAR and VBAR statements 728, 801
 - PIE and DONUT statements 821
 - PLOT statement, G3D procedure 1302
 - PLOT statement, GCONTOUR procedure 893
 - PLOT statement, GPLOT procedure 1106
 - PRISM statement 1025
 - SCATTER statement, G3D procedure 1307
 - STAR statement 836
 - CTEXT= options, GOPTIONS statement 225, 276
 - CTILE= and CTILES= options, CHART statement 1189
 - CTIPHILT= argument, META2HTM macro 565
 - CTITLE= graphics option 225, 276
 - CTOP= option, PLOT statement 1302
 - cumulative frequency statistic 745, 782
 - cumulative percentage statistic 746, 783
 - current window system, DSGI 1359
 - curve-drawing capability, device 270
 - custom graphs, creating with DSGI 1356
 - CUTOFF= macro argument 547
 - CV= option
 - BUBBLE statement 1094
 - PLOT statement, GCONTOUR procedure 893
 - PLOT statement, GPLOT procedure 1107
 - SYMBOL statement 186, 206
 - CVALUE= macro argument, DS2CSF macro 563
 - CVREF= option
 - BUBBLE statement 1094
 - PLOT statement, GCONTOUR procedure 893
 - PLOT statement, GPLOT procedure 1107
- ## D
- D= option, TITLE, FOOTNOTE, and NOTE statements 217
 - DASH option 277
 - dashed lines
 - hardware-generated 277
 - lengths of dashes, scaling 278
 - DASHLINE= option, GDEVICE procedure 277
 - DASHSCALE= graphics option 278
 - data-dependent coordinates with GSLIDE procedure 1281
 - data library for rendered fonts 347
 - DATA= macro argument, DS2CSF macro 561
 - DATA= option
 - G3D procedure 1300
 - G3GRID procedure 1332
 - GBARLINE procedure 750
 - GCHART procedure 786
 - GCONTOUR procedure 888
 - GFONT procedure 946
 - GKEYMAP procedure 989
 - GMAP procedure 1001, 1007
 - GPLOT procedure 1089
 - GPROJECT procedure 1168
 - GRADAR procedure 1185
 - GREDUCE procedure 1216
 - REMOVE procedure 1226
 - data sets 29
 - contour plots, input for 887
 - DSGI data sets 1358
 - font data sets 951, 952, 958
 - generating for radar/star charts (example) 1196
 - GKEYMAP data sets 987
 - kern data sets 949, 958, 959
 - locking, automatic 31
 - space data sets 960

- three-dimensional graphs 1298
- DATA step 27
 - Annotate data sets 600
- data tips in Web presentations 568
- data values, formatting 413
- DATAFILE= option, MAPIMPORT procedure 1349
- DATASYS option, GANNO procedure 708, 710
- DATATIPHIGHLIGHTCOLOR= parameter, Metaview Applet 475
- DATATIPSTYLE= parameter, Metaview Applet 475
- DATATYPE= macro argument 538
- date-time information
 - as axis values, ordering 131
 - ordering axis tick marks (example) 226
- %DCLANNO macro, Annotate facility 683
- DDLEVEL# applet parameter 405
- DDLEVEL= parameter, JAVA and ActiveX 428
- DEBUG function, Annotate facility 620
- debugging
 - Annotate facility 604
 - DSGI programs 1360
- DEF option, TDEF statement 1260
- DEFAULTTARGET= graphics option 476
- DEFINE STYLE statement, TEMPLATE procedure 489
- DEG option, GPROJECT procedure 1169
- DEGREE option, GPROJECT procedure 1169
- DEL option, TDEF statement 1260
- delay between displayed graphs 278, 310
- DELAY= graphics option 459
- DELETE function (DSGI) 1459
- DELETE option, TDEF statement 1260
- DELETE statement, GDEVICE procedure 925
- DELETE statement, GREPLAY procedure 1251
- deleting
 - blanks from data values 414
 - catalog entries 1251
 - color maps 1250
 - graphics output, after display 285, 287
 - polygon overlap 331, 335
 - replacing/overwriting files 70
 - templates 1262
- density values, map data sets 1218
- DENSITY variable (map data sets) 1039, 1213
- DEPTH= macro argument 547, 563
- DES= option
 - BLOCK statement, GCHART procedure 789
 - BLOCK statement, GMAP procedure 1013
 - BUBBLE statement 1094
 - CDEF statement, GREPLAY procedure 1249
 - CHART statement, GRADAR procedure 1189
 - CHORO statement 1019
 - GANNO procedure 709
 - GPRINT procedure 1149
 - GSLIDE procedure 1280
 - HBAR and VBAR statements 802
 - PIE and DONUT statements 822
 - PLOT statement, G3D procedure 1303
 - PLOT statement, GCONTOUR procedure 893
 - PRISM statement 1025
 - SCATTER statement, G3D procedure 1307
 - STAR statement 836
 - SURFACE statement 1032
 - TDEF statement, GREPLAY procedure 1260
 - TREPLAY statement, GREPLAY procedure 1263
- DES option, BAR statement 755
- DESCENDING option
 - BAR statement 755
 - BY statement 142, 1227
 - HBAR and VBAR statements 802
 - PIE and DONUT statements 821
 - PLOT statement 766
- DESCRIPTION= option
 - BAR statement 755
 - BLOCK statement, GCHART procedure 789
 - BLOCK statement, GMAP procedure 1013
 - BUBBLE statement 1094
 - CHART statement, GRADAR procedure 1189
 - CHORO statement 1019
 - GANNO procedure 709
 - GDEVICE procedure 279
 - GPRINT procedure 1149
 - GSLIDE procedure 1280
 - HBAR and VBAR statements 802
 - ODS statements 496
 - PIE and DONUT statements 822
 - PLOT statement, G3D procedure 1303
 - PLOT statement, GCONTOUR procedure 893
 - PLOT statement, GPLOT procedure 1107
 - PRISM statement 1025
 - SCATTER statement, G3D procedure 1307
 - STAR statement 836
 - SURFACE statement 1032
- descriptions of catalog entries 55
- destinations, ODS 489
- DETAIL= option, PIE and DONUT statements 822, 883
- DETAIL_= options, PIE and DONUT statements 822, 883
- DEV option, ? statement 1246
- DEVADDR= GOPTIONS statement 279
- device catalogs 916, 917
- device drivers 41
 - assigning 43
 - comparisons between 440
 - controlling output with 45
 - entries and catalogs 916
 - Listing available 44
 - managing colors list of (example) 1292
 - selecting 43
 - Web output 369
- device drivers, specifying 1251
- device entries 42, 916
 - browsing contents of 44
 - creating or modifying 72, 934
 - custom, creating (example) 936
 - transporting 58
- DEVICE function (DSGI) 1411, 1468
- device-generated graphics
 - circles and arcs 271
 - dashed lines 277
 - line thickness 323
 - pie filling 334
 - plot symbols 354
 - polygon-fill 336
 - rectangle-fill 345
 - vertices, maximum drawn 324
- DEVICE= graphics option 52, 72, 280
- static graphics 439
- device maps 983
 - asymmetrical 986
 - basics of 986
 - creating and using 990
 - GKEYMAP data sets 987
 - GKEYMAP data sets, generating 990
 - specifying 280
- DEVICE option, ? statement 1246
- DEVICE= option, GKEYMAP procedure 989
- device parameters 42, 261
 - complete list of, alphabetical 262
 - modifying 45
- DEVICE statement, GREPLAY procedure 1251
- devices
 - capabilities of, listing 281
 - colors list, default 94
 - external dimensions, display 34
 - hardware patterns 176, 181
 - how output is written to 306
 - identifying type of 283
 - location of, for output 279
 - model numbers 324
 - nicknames for 299
 - portability between 40
 - sending strings to 301, 303
 - specifying type of 358
 - user input, enabling 360
- DEVMAP= graphics option 280, 986
- DEVMAP= option, GDEVICE procedure 280
- DEVMAP option, GKEYMAP procedure 989
- DEVOPTS= option, GDEVICE procedure 281
- DEVTYPE= option, GDEVICE procedure 283
- diagnostic messages, Annotate facility 699
- diagrams, CSF
 - drill-down functionality 573
 - DS2CSF macro 527, 528
 - hotspots 532
 - sample diagrams 530
- direct display 50
- DIRECT= parameter, JAVA and ActiveX 428
- DIRECTORY window (GDEVICE) 929
- DIRECTORY window (GREPLAY) 1265
- DISABLE DRILLDOWN applet parameter 414
- disabling drill-down functionality 414
- DISCRETE option
 - BAR statement 743, 756
 - BLOCK statement, GCHART procedure 790
 - BLOCK statement, GMAP procedure 1013
 - CHORO statement 1019
 - GCHART procedure statements 780
 - HBAR and VBAR statements 728, 802
 - PIE and DONUT statements 823
 - PRISM statement 1026
 - STAR statement 836
- discrete variables
 - bar variables 742, 743
 - chart variables 779, 780
 - charting in star chart (example) 881
 - map variables 1004
- DISPLAY environment variable, batch mode 32
- display size (lines) 308
- DISPLAY statement 284
- displaying fonts 940, 943
- DISPOSAL= graphics option 459
- DISPOSAL statement 285
- DOCTYPE= macro argument 553

- documentation, SAS/GRAPH
 - online help, locations for 385
 - donut charts 6
 - basics 776
 - labels 829
 - slice labels and formatting 830
 - statistic and group headings 827, 832
 - subgrouping in (example) 848
 - terms used with 778
 - DONUT statement, GCHART procedure
 - ActiveX and Java support for 1521
 - subgrouping in pie or donut chart (example) 848
 - syntax and options 818
 - DONUTPCT= option, DONUT statement 823
 - DOWN= option
 - CHART statement, GRADAR procedure 1190
 - LEGEND statement options 153
 - PIE and DONUT statements 823
 - STAR statement 837
 - DOWNVAR= option, CHART statement 1190
 - DRAW function, Annotate facility 620, 1539
 - %DRAW macro, Annotate facility 683
 - DRAW= option, TITLE, FOOTNOTE, and NOTE statements 217
 - DRAW2TXT function, Annotate facility 622, 1540
 - %DRAW2TXT macro, Annotate facility 684
 - DRAWIMAGE= parameter, JAVA 428
 - drawing areas, Annotate graphics 596
 - DRAWMISSING= parameter, JAVA 428
 - DRAWSIDES= parameter, JAVA 428
 - drill-down functionality 571
 - Annotate facility for 500
 - Annotate graphics in drill-down graphs (example) 719
 - bar charts with (example) 255, 856
 - choropleth map (example) 1054
 - constellation charts 524
 - creating plots with (example) 1141
 - critical success factor diagrams 532
 - generating drill-down graphs using DSGI (example) 1395
 - GIF output hotspots (example) 452
 - treeview diagrams 510
 - drill-down links
 - ActiveX 392
 - disabling 414
 - Java applets 400
 - levels for, customizing 405
 - drill-down tags 411, 412
 - DRILLDOWN= parameter, JAVA and ActiveX 428
 - DRILLDOWNMODE applet parameter 401, 408
 - DRILLDOWNMODE parameter 571
 - DRILLDOWNMODE= parameter, JAVA and ActiveX 429
 - DRILLFUNC= parameter, JAVA and ActiveX 428
 - DRILLPATTERN= parameter, JAVA and ActiveX 429
 - DRILLTARGET applet parameter 410, 413
 - DRILLTARGET= parameter, JAVA and ActiveX 429
 - DRILTARG= macro argument 547, 563
 - DRILTARG= option 573
 - DRILURL= macro argument, DS2CSF macro 563
 - DRILURL= option 573
 - driver modules 325
 - driver termination 286
 - drivers, initializing 285
 - drop shadows, legends 153, 163
 - DRVINIT1= and DRVINIT2= options, GDEVICE procedure 285
 - DRVINIT1= and DRVINIT2= options, GPTIONS statement 285
 - DRVQRY= option, GDEVICE procedure, executing before driver initialization 286
 - DRVTERM1= and DRVTERM2= options, GDEVICE procedure 286
 - DRVTERM1= and DRVTERM2= options, GPTIONS statement 286
 - DS2CONST macro 383, 515
 - arguments of 518, 535
 - arguments of, character transcoding 561
 - arguments of, data definition 537
 - arguments of, diagram appearance 545
 - arguments of, file generation 544
 - arguments of, page formatting 552
 - arguments of, titles and footnotes formatting 556
 - chart with simple arcs (example) 518
 - chart with weighted arcs (example) 520
 - enhancing presentations for 517
 - hotspots 524
 - stylesheets, macro arguments for 554
 - XML written to external file (example) 522
 - DS2CSF macro 383
 - arguments of 530, 535
 - arguments of, character transcoding 561
 - arguments of, data definition 561
 - arguments of, diagram appearance 562
 - arguments of, page formatting 552
 - arguments of, titles and footnotes formatting 556
 - drill-down functionality 573
 - enhancing presentations for 529
 - hotspots 532
 - sample diagrams 530
 - stylesheets, macro arguments for 554
 - DS2TREE macro 383
 - arguments of 507, 535
 - arguments of, character transcoding 561
 - arguments of, data definition 537
 - arguments of, diagram appearance 545
 - arguments of, file generation 544
 - arguments of, page formatting 552
 - arguments of, titles and footnotes formatting 556
 - enhancing presentations for 506
 - stylesheets, macro arguments for 554
 - DSGI (DATA Step Graphics Interface) 15, 1354, 1401
 - Annotate facility vs. 1354
 - attributes for graphics elements 1368, 1373
 - creating simple graphics 1367
 - examples of using 1381
 - functions and routines 1360
 - GASK routines 1394, 1404
 - GDRAW functions, list of 1446
 - global statements with 1358
 - GRAPH functions, list of 1457
 - GSET functions, list of 1462
 - how to use 1357
 - images, displaying 119
 - inserting graphs into DSGI output 1379
 - operating states 1359, 1370, 1402
 - processing statements in loops 1380
 - return codes, list of 1501
 - syntax 1355
 - utility functions, list of 1402
 - viewports and windows 1376, 1385
 - DUPCHECK= macro argument 547
 - duplex printing 265, 287
 - DUPLICATEVALUES= parameter, JAVA 429
 - DUPOK option, GPROJECT procedure 1168
- ## E
- E1=, ..., E5= options, GREduce procedure 1216
 - EAST option, GPROJECT procedure 1169
 - EASTLONG option, GPROJECT procedure 1169
 - EBCDIC-to-ASCII translation 358
 - editable output 296
 - editing
 - device entries 72
 - graphics output 55
 - Electronic font 87
 - ELLARC function (DSGI) 1449
 - ELLIPSE function (DSGI) 1450
 - ellipses, drawing with DSGI 1449, 1450
 - EMPTY variable, Annotate facility 662
 - ENCODE= macro argument 553
 - engines 31
 - enhancement variables in Web presentations 574
 - Enterprise Guide 2.0 395
 - equal-area map (Albers_##APOSTROPHE##_) projections 1165
 - clipping map areas (example) 1178
 - default projection specifications, using (example) 1174
 - projection criteria 1172
 - when to use 1172
 - ERASE= graphics option 287
 - ERASE= option, GDEVICE procedure 287
 - erasing
 - blanks from data values 414
 - graphics output, after display 285, 287
 - replacing/overwriting files 70
 - ERRORBAR= option
 - BAR statement 756
 - HBAR and VBAR statements 803
 - horizontal bar charts (example) 854
 - errors, sizing 40
 - errors and error messages, Annotate facility 699
 - ESRI files, importing as map data sets 1347, 1348
 - examples 1349
 - example programs 21
 - examples 19
 - EXCLUDE statement, MAPIMPORT procedure 1349
 - executable driver modules 325
 - executable module 42
 - EXPLODE= option, PIE and DONUT statements 823

- exporting graphics output 59
 - interactively 62
 - modified device entries for 72
 - program statements for 62
 - exporting output interactively 62
 - EXTENSION= graphics option 288
 - external dimensions, device 34
 - external files 1148
 - file extensions for 288
 - names for 63
 - replacing 70
 - external files, converting text to graphics output 1147
 - adjusting character size in output (example) 1156
 - adjusting output size 1150
 - external text files, about 1148
 - fonts 1152
 - specifying color text in output (example) 1153
 - syntax and options, GPRINT procedure 1149
 - files, converting text to graphics output 1147
 - adjusting character size in output (example) 1156
 - adjusting output size 1150
 - external text files, about 1148
 - fonts 1152
 - specifying color text in output (example) 1153
 - syntax and options, GPRINT procedure 1149
 - FILETYPE= option, GIMPORT procedure 973
 - FILINDEX function (DSGI) 1413, 1470
 - filig images 114
 - FILL function (DSGI) 1451
 - FILL= graphics option 292
 - FILL= option
 - PIE and DONUT statements 823
 - STAR statement 837
 - FILL= option, GDEVICE procedure 292
 - filled fonts 941
 - FILLED option, GFONT procedure 949
 - FILLINC= graphics option 292
 - FILLINC= option, GDEVICE procedure 292
 - FILLPOLYGONEDGES= parameters, JAVA and ActiveX 430
 - FILREP function (DSGI) 1470
 - FILSTYLE function (DSGI) 1415, 1472
 - FILTYPE function (DSGI) 1415, 1471
 - FIPS codes 1003, 1033
 - labeling U.S. states in choropleth map (example) 1061
 - FISHEYE= macro argument 548
 - fixed-length output records 296
 - flow control, device 311
 - FNTNAME= macro argument 548
 - FNTSIZE= macro argument 548
 - FNTSTYL= macro argument 548
 - font data sets 951
 - creating 958
 - variables for, list of 952
 - FONT= option 158
 - AXIS statement options 136
 - CHART statement, GRADAR procedure 1190
 - LABEL= option, DONUT statement 829
 - POINTLABEL= specification 197
 - SYMBOL statement 186, 904
 - TITLE, FOOTNOTE, and NOTE statements 217
 - FONT= specification 76
 - FONTLIST command 76
 - FONTRES= graphics option 293
 - fonts 75
 - ActiveX and 392
 - axis labels 127, 133, 136
 - axis values 136
 - basics about 940
 - bubble plots 1092
 - BY lines 143, 289
 - complete list of 82
 - contour plot labels 903
 - creating and storing 940, 946, 951
 - creating figures for symbols font (example) 964
 - default 77, 78, 268, 270
 - displaying 940, 943
 - displaying with character codes (example) 962
 - donut chart labels 829
 - full names for 80
 - GFONT0 library 941
 - GPRINT procedure and 1152
 - graphics output text 294
 - Java and 400
 - kern data sets 949, 958, 959
 - legend label 153
 - legend text 158
 - legend values 157
 - listing available 76
 - listing available characters 987
 - mapping 972
 - maximum and minimum 940
 - open at one time 290
 - plot point label 197
 - plot symbols 186
 - rendering 82, 289, 295, 346, 3
 - rendering, data library for 347
 - resolution 293
 - scaling in graphics output 274, 350, 351
 - space data sets 960
 - specifications for 76
 - substituting for imported CGMs 974
 - text in graphics output 217
 - titles and footnotes 295, 493
 - transporting 58
 - troubleshooting 583
 - where stored 77, 82
 - FOOTNOTE element (HTML), macro arguments for 556
 - FOOTNOTE option, GOPTIONS procedure 1077, 1079
 - FOOTNOTE statement 27, 123, 210, 224
 - ActiveX and Java support for 1517
 - BY statement with 145
 - displaying with GOPTIONS procedure (example) 1079
 - footnotes 211, 224
 - angle of rotation 213, 219, 222
 - boxes around 215, 216
 - colors for 215, 216, 276
 - default characteristics, setting 225
 - defining text of 222, 226
 - fonts for 217
 - hyperlinks for 220
 - justification 218
 - ODS output 492, 493
 - positioning 39, 221
 - size of 218, 316
 - spacing around 221
 - text breaks 225
 - underlining 223
 - footnotes macro, arguments for 556
 - foreground colors
 - default, defining 272
 - reversing black and white 353
 - FORMAT= attribute 108, 113
 - FORMAT= option, GDEVICE procedure 294
 - FORMAT= option, GIMPORT procedure 973
 - FORMAT statement 27
 - formatting
 - axis labels 127, 133, 135
 - axis tick marks 129, 139
 - axis values 135
 - BY lines 143
 - contour plot lines and labels 903
 - donut chart labels 829
- ## F
- F option
 - GFONT procedure 949
 - GOPTIONS procedure 1077
 - AXIS statement options 136
 - LABEL= option, DONUT statement 829
 - LEGEND statement options 158
 - POINTLABEL= specification 197
 - SYMBOL statement 186
 - TITLE, FOOTNOTE, and NOTE statements 217
 - FACHE= graphics option 290
 - FACTOR= macro argument 548
 - FASTTEXT= graphics option 288
 - FBY= graphics option 289
 - FCLASS= macro argument 556
 - FCOLOR= macro argument 556
 - feature tables 1001
 - creating maps using (example) 1069
 - merging map data sets with 1001, 1069
 - FFACE= macro argument 556
 - FILCOLOR function (DSGI) 1412, 1469
 - FILE= argument, ODS HTML statement 165
 - file extensions 288
 - file specifications for data sets 30
 - FILECLOSE= graphics option 290
 - FILECLOSE= option, GDEVICE procedure 290
 - filename extensions 63
 - FILENAME statement 444
 - storing in device entry 314
 - FILENAME statements 27, 28
 - FILEONLY= graphics option 291
 - FILEREF= option
 - GIMPORT procedure 973
 - GPRINT procedure 1149
 - filerefs 970
 - FILEREP function (DSGI) 1413
 - files
 - image file types 106
 - replacing/overwriting 70
 - sending strings to 301, 303
 - storing graphics output as 291

- legend label 153
 - legend values 157
 - legends 150
 - maps 1035
 - pie and donut chart slice labels 830, 873
 - Web output 488
 - FR option
 - BAR statement 757
 - BUBBLE statement 1095
 - HBAR and VBAR statements 804
 - PLOT statement, GPLOT procedure 1108
 - FRAME= argument, ODS HTML statement 166
 - FRAME function, Annotate facility 1540
 - frame. legend 153
 - %FRAME macro, Annotate facility 684
 - FRAME option
 - BAR statement 757
 - BUBBLE statement 1095
 - CHART statement, GRADAR procedure 1190
 - GSLIDE procedure 1280
 - HBAR and VBAR statements 728, 804
 - PLOT statement, GPLOT procedure 1108
 - FRAME= option, LEGEND statement options 153
 - FRAME= option, ODS statements 498
 - frames
 - backplane images 115
 - images on 115, 806, 1096, 1110
 - ODS output 497
 - frames, drawing 1281
 - FREQ and FREQ= options
 - BAR statement 757
 - BLOCK statement 790
 - CHART statement, GRADAR procedure 1190
 - GCHART procedure statements 783
 - HBAR and VBAR statements 804, 805
 - PIE and DONUT statements 823
 - PLOT statement 746, 766
 - STAR statement 837
 - weighted statistics, bar line chart (example) 770
 - FREQLABEL= option, HBAR and VBAR statements 804
 - FREQNAME= parameters, JAVA and ActiveX 430
 - frequency statistic 745, 782
 - frequency variable, specifying
 - BAR statement 757
 - BLOCK statement 790
 - CHART statement, GRADAR procedure 1190
 - GCHART procedure statements 783
 - HBAR and VBAR statements 804, 805
 - PIE and DONUT statements 823
 - PLOT statement 746, 766
 - STAR statement 837
 - weighted statistics, bar line chart (example) 770
 - FROM variable (GKEYMAP data set) 987
 - FRONTREF option
 - BAR statement 758
 - HBAR and VBAR statements 805
 - FS option, GREPLAY procedure 1244
 - FS statement, GDEVICE procedure 925
 - FS statement, GREPLAY procedure 1252
 - FSIZE= macro argument 556
 - FTEXT= option
 - GIMPORT procedure 972
 - GOPTIONS statement 225, 294
 - FTITLE= graphics option 225, 295
 - FTRACK= graphics option 295
 - FUNCTION variable, Annotate facility 595, 646
 - functions, Annotate 594, 615
 - FWIDTH= option, LEGEND statement options 153
- ## G
- G_ COLOR= parameters, JAVA and ActiveX 430
 - G_ COLORV= parameters, JAVA and ActiveX 430
 - G_ DEP= parameters, JAVA and ActiveX 430
 - G_ DEPTH= parameters, JAVA and ActiveX 430
 - G_ DEPTHV= parameters, JAVA and ActiveX 430
 - G_ DEPV= parameters, JAVA and ActiveX 430
 - G_ GROUP= parameters, JAVA and ActiveX 431
 - G_ GROUPV= parameters, JAVA and ActiveX 431
 - G_ INDEP= parameters, JAVA and ActiveX 431
 - G_ INDEPV= parameters, JAVA and ActiveX 431
 - G_ LABEL= parameters, JAVA and ActiveX 431
 - G_ LABELV= parameters, JAVA and ActiveX 431
 - G_ SUBGR= parameters, JAVA and ActiveX 431
 - G100 option
 - BLOCK statement 790
 - HBAR and VBAR statements 805
 - G3D procedure 1295
 - ActiveX and Java support for 1537
 - generating default surface plot (example) 1314
 - generating simple scatter plot (example) 1318
 - input data sets 1298
 - PLOT statement 1301
 - rotating scatter plot (example) 1323
 - rotating surface plot (example) 1316
 - SCATTER statement 1305
 - shapes in (example) 1320
 - syntax and options 1300
 - tilting surface plots (example) 1317
 - G3GRID procedure 887, 1327
 - controlling observations in output data set 1336
 - GRID statement, G3GRID procedure 1333
 - interpolation methods 1329
 - simple contour plot, generating (example) 905
 - spline interpolation, partial (example) 1342
 - spline interpolation, with smoothed spline (example) 1339
 - spline interpolation (example) 1343
 - syntax and options 1332
 - using default interpolation method (example) 1337
 - GACCESS= graphics option 72, 296
 - GACCESS= option, GDEVICE procedure 296
 - GANNO procedure 601, 707
 - Annotate graphics in drill-down graphs (example) 719
 - producing multiple graphs (example) 715
 - scaling data-dependent output (example) 710
 - scaling graphs with DATASYS option 710
 - storing Annotate graphics (example) 713
 - syntax and options 708
 - Web output, generating 500
 - GAREABAR procedure 725
 - ActiveX and Java support for 1518
 - chart with numeric category variable (example) 731
 - chart with subgrouping and variable percentages (example) 735
 - chart with subgrouping (example) 733
 - simple area bar chart (example) 729
 - syntax and options 727
 - GASK routines
 - DSGI 1394
 - list of, reference 1404
 - GAXIS= option, HBAR and VBAR statements 805
 - GBARLINE procedure 739, 796
 - ActiveX and Java support for 1519
 - BAR statement 751
 - basic graph with styles (example) 768
 - interpolation methods 740
 - missing values 747
 - patterns and outlines 748
 - PLOT statement 765
 - SYMBOL statement 768
 - syntax and options 749
 - V6COMP graphics option 749
 - weighted statistics, calculating (example) 770
 - GCHART procedure 785
 - ActiveX and Java support for 1521
 - bar chart with sum statistic (example) 846
 - block chart with sum statistic (example) 842
 - BLOCK statement 787
 - BY-group processing with (example) 240
 - BY statement 144
 - discrete numeric variables, in star chart (example) 881
 - DONUT statement 818
 - drill-down functionality in bar chart (example) 856
 - DSGI viewport with (example) 1385
 - error bars in horizontal bar chart (example) 854
 - grouping and subgrouping in block chart (example) 844
 - HBAR and VBAR statements 796
 - legends for pie chart patterns and midpoints (example) 875
 - midpoints and statistics in horizontal bar chart (example) 851
 - missing values 779
 - PATTERN statement 169
 - patterns 171
 - patterns and outlines 174, 784
 - pie and donut chart slice labels (example) 873
 - PIE and PIE3D statements 818
 - pie chart with sum statistic (example) 842
 - star chart with sum statistic (example) 879
 - STAR statement 833
 - subgroup labels (example) 607

- subgrouping in pie or donut chart (example) 848
- subgrouping in vertical bar chart (example) 848
- syntax and options 785, 888
- V6COMP graphics option 785
- GCLASS= graphics option 298
- Gcolors window (GDEVICE) 931
- GCONTOUR procedure 885
 - ActiveX and Java support for 1526
 - AXIS statement 886
 - contour levels, specifying (example) 908
 - missing values 887, 889
 - PATTERN statement 169
 - patterns 173
 - patterns and joins in contour plots (example) 910
 - PLOT statement 889
 - simple contour plot, generating (example) 904
 - spline interpolation (example) 1343
- GCOPIES= graphics option 298
- GCOPIES= option, GDEVICE procedure 298
- GDDM device driver
 - device nicknames 299
 - patterns 181
 - writing ADMGDF or GDF files 262
- GDDMCOPY= graphics option 299
- GDDMTOKEN= graphics option 299
- GDEST= graphics option 300
- GDEVICE procedure 42, 916
 - ADD statement 921
 - COPY statement 924
 - creating or modifying device entries 934
 - custom device entry, creating (example) 936
 - default hardware fonts 79
 - DELETE statement 925
 - FS statement 925
 - LIST statement 926
 - Listing available drivers 44
 - MODIFY statement 926
 - QUIT statement 927
 - RENAME statement 928
 - syntax and options 920
 - windowing and program modes 917
 - windowing mode, using 928
- GDEVICE windows 261
- GDF files, writing with GDM driver 262
- GDRAW function, DSGI 119, 1446
- G_drill-down tags 412
- GEND= graphics option 300
- GEND= option, GDEVICE procedure 300
- Gend window (GDEVICE) 933
- \$GEOREF format 1001
- Gepilog field, device entries 336, 338, 339
- GEPILOG= graphics option 301
- GEPILOG= option, GDEVICE procedure 301
- Gepilog window (GDEVICE) 933
- Geprolog field, device entries 337
- GFONT procedure 939
 - creating figures for symbols font (example) 964
 - creating fonts 940, 946, 951
 - displaying fonts 940, 943
 - displaying fonts and character codes (example) 962
 - syntax and options 942
- GFONT0 library 941
- GFOOTNOTE= option, ODS HTML statement 492
- GFORMS= graphics option 302
- GIF device driver 382
 - ACTXIMG, JAVAIMG vs. 440
 - data tips for 568
 - developing web presentations 443
 - drill-down links in images 571
 - HTML files, generating 445
 - names for image output files 445
- GIF output
 - generating with ODS (example) 450
 - hyperlinks in (example) 452
- GIF presentations 377
- GIFANIM device driver 378, 457
 - graphics options for presentations 459
 - sample programs 459, 463
- GIMPORT procedure 969
 - adjusting graphics output (example) 979
 - basics of importing graphics 970
 - creating and importing CGM (example) 977
 - MAP statement 974
 - mapping colors 971
 - SCALE statement 975, 979
 - syntax and options 973
 - TRANSLATE statement 976, 979
- GINIT function (DSGI) 1402
- GKEYMAP data sets 987, 990
- GKEYMAP procedure 983
 - asymmetrical maps 986
 - creating maps 990
 - modifying keymaps (example) 990
 - syntax and options 989
- global statements 14, 27, 33, 123, 135
- GMAP procedure 169, 996, 1006
 - ActiveX and Java support for 1527
 - assigning formats to response variables (example) 1049
 - BLOCK statement 1010
 - BY statement 144
 - CHORO statement 1017
 - creating maps using feature tables (example) 1069
 - drill-down functionality in maps (example) 1054
 - ID statement 1009
 - labeling U.S. states in choropleth map (example) 1061
 - midpoints in prism map, specifying (example) 1065
 - PRISM statement 1023
 - projecting an Annotate data set (example) 1180
 - response levels 1004
 - response levels in block map, specifying (example) 1047
 - rotating and tilting surface map (example) 1068
 - simple block map (example) 1045
 - simple choropleth map (example) 1053
 - simple prism map (example) 1063
 - simple surface map (example) 1066
 - specialty map data sets 1003
 - SURFACE statement 1030
 - syntax and options 1007
- gnomonic projections 1167
 - emphasizing map areas (example) 1177
 - projection criteria 1172
 - when to use 1172
- GOPTIONS procedure 1075
 - displaying graphics options without descriptions (example) 1079
 - displaying TITLE and FOOTNOTE statements (example) 1079
 - syntax and options 1077
 - using with GOPTIONS statement 1075
- GOPTIONS statement 27, 123, 146, 261, 2
 - ActiveX and Java support for 1510
 - resetting options 348
 - testing with GTESTIT procedure (example) 1291
 - using with GOPTIONS procedure 1075
- GOUT= option
 - GANNO procedure 709
 - GCHART procedure 786
 - GCONTOUR procedure 888
 - GFONT procedure 944
 - GIMPORT procedure 974
 - GMAP procedure 1008
 - GPLOT procedure 1089
 - GPRINT procedure 1149
 - GRADAR procedure 1185
 - GREPLAY procedure 1244
 - GSLIDE procedure 1280
 - GTESTIT procedure 1290
 - PROC statement 54
- GOUT option, ? statement 1246
- GOUT statement, GREPLAY procedure 1252
- GOUTMODE= graphics option 302
- GPLOT procedure 169, 1081
 - ActiveX and Java support for 1530
 - adding right vertical axis (example) 1124
 - BUBBLE statement 1090
 - BUBBLE2 statement 1098
 - BY statement 144
 - connecting plot data points (example) 1129
 - different scales of values (example) 1138
 - filling areas in overlay plot (example) 1134
 - generating overlay plot (example) 1131
 - generating simple bubble plots (example) 1121
 - input data set 1086
 - labeling and sizing plot bubbles (example) 1122
 - PATTERN statement 1120
 - plot basics 1085
 - PLOT statement 1101, 1117
 - PLOT2 statement 1115
 - plots with drill-down for Web (example) 1141
 - plotting three variables (example) 1135
 - plotting two variables (example) 1126
 - scaling graphs with DSGI windows (example) 1388
 - SYMBOL statement 204, 1114, 1120
 - syntax and options 1088
- GPRINT function (DSGI) 1403
- GPRINT procedure 1147
 - adjusting character size in output (example) 1156
 - adjusting output size 1150
 - external text files, about 1148
 - fonts 1152
 - specifying color text in output (example) 1153

- syntax and options 1149
- GPROJECT procedure 1161
 - basic usage 1172
 - clipping map areas (example) 1178
 - default projection specifications, using (example) 1174
 - emphasizing map areas (example) 1177
 - ID statement 1172
 - input map data sets 1163
 - projecting an Annotate data set (example) 1180
 - syntax and options 1168
 - types of map projections 1165
- GPROLOG= graphics option 303
- GPROLOG= option, GDEVICE procedure 303
- Gprolog window (GDEVICE) 932
- GPROTOCOL= graphics option 303
- GPROTOCOL= option, GDEVICE procedure 303
- GRADAR procedure 1183
 - ActiveX and Java support for 1535
 - assigning axis definitions to axis spokes (example) 1210
 - changing star type in radar charts (example) 1207
 - CHART statement 1185
 - color and line styles in radar charts (example) 1208
 - filling stars in radar charts (example) 1204
 - generating data set for (example) 1196
 - images in radar charts (example) 1205
 - multiple classification variables in radar charts (example) 1202
 - overlying radar charts (example) 1199
 - producing basic radar chart (example) 1198
 - specifying mode for radar charts (example) 1209
 - syntax and options 1184
 - tiling radar charts (example) 1201
- GRADIENTBACKGROUND= parameters, JAVA and ActiveX 431
- GRADIENTENDCOLOR= parameters, JAVA and ActiveX 431
- GRADIENTSTARTCOLOR= parameters, JAVA and ActiveX 431
- Graph applet 372
 - disabling drill-down functionality 414
 - drill-down tags 412
 - Local drill-down example 415
 - Local drill-down mode 400, 401, 402, 405,, parameters for, list of 424
 - Script drill-down example 416
- GRAPH functions, DSGI 1456
- Graph-N-Go 15
- GRAPH window 49, 62
 - printing from 52
 - size of 50
- graphics, importing 969
 - adjusting graphics output (example) 979
 - basics of importing graphics 970
 - creating and importing CGM (example) 977
 - MAP statement, GIMPORT procedure 974
 - mapping colors 971
 - SCALE statement 979
 - SCALE statement, GIMPORT procedure 975
 - syntax and options 973
 - TRANSLATE statement, GIMPORT procedure 976, 979
- graphics output area 34
 - Annotate facility 598
 - border around 266
 - columns in 36, 274, 322, 332
 - maximum colors allowed 93, 324
 - multiple device output 40
 - offset between graphs and 314, 360
 - rows in 3, 36, 323, 344, 350,
 - size of 316, 361, 363, 364,
- graphics output names 63
- graphics output text
 - colors for 276
 - fonts 294
 - size of 316
- GRAPHLIST function (DSGI) 1416
- GRAPHRC= graphics option 305
- graphs 4
 - background images 318
 - box plots 185, 187, 201, 233
 - BY lines 143
 - client vs. server rendering 584
 - creating interactively 395
 - displaying in timed series 310
 - enhancing with DSGI 1356
 - positioning 39
 - redrawing (overdrawing) 348
 - saving to files 66, 68
 - suppressing display of 284
- gray scale color scheme 96, 99
- GREDUCE procedure 1039, 1213
 - ID statement 1218
 - reducing map of Canada (example) 1220
 - specifying density levels 1218
 - syntax and options 1216
 - unmatched area boundaries 1215
- REMOVE procedure 1223
 - BY statement 1227
 - ID statement 1228
 - outline map of Africa, creating (example) 1232
 - removing U.S. state boundaries (example) 1228
 - syntax and options 1226
- GREPLAY procedure 55, 1238
 - catalog entries, about 1239
 - creating templates and color maps 1268, 1270, 1274
 - device color limitations 105
 - how to use 1241
 - managing catalog entries 1267
 - relaying graphics output in templates 1270, 1272
 - replaying catalog entries 1268
 - switching between line more and procedure windows 1252
 - syntax and options 1243
- TRANSLATE statement, GIMPORT procedure 976, 979
- graphics catalogs 53
 - accessing 53
 - converting 59
 - duplicate entry names 1240
 - listing and managing entries 55
 - replacing 70
 - specifying/assigning 1252, 1253
- graphics editor 55
- Graphics Editor window, printing from 52
- graphics elements, creating DSGI 119, 1446
- graphics files, saving and printing 51
- graphics options 14, 146, 261
 - animations, configuring 459
 - complete list of, alphabetical 262
 - displaying without description (example) 1079
 - ODS output with 493
 - overriding device parameters 46
 - resetting 348
- graphics output 34, 48, 584
 - adjusting character size in output (example) 1156
 - Annotate data sets 601
 - Annotate graphics with 601
 - appending strings to records 300
 - appending to or replacing catalogs 302
 - background images 318, 319
 - body files 491
 - controlling with device drivers 45
 - default destinations for 291
 - destination for 307
 - display size, in lines 308
 - displaying 49
 - displaying images in 319
 - editing 55
 - erasing after display 285, 287
 - exporting 59, 62
 - file formats 60
 - generating for ActiveX 391
 - generating output for Java 398
 - how written, specifying 306
 - positioning 39
 - prefixing records 309
 - previewing as if on different device 52, 356
 - protocol module, specifying 303
 - queuing for log messages 345
 - relaying graphics output in templates 1272
 - replaying in templates 1270
 - resizing 975
 - reversing black and white 353
 - size 1150
 - specifying color text in output (example) 1153
 - suppressing display of 284
 - transporting and converting 56
- graphics output, converting text files to 1147
 - adjusting character size in output (example) 1156
 - adjusting output size 1150
 - external text files, about 1148
 - fonts 1152
 - specifying color text in output (example) 1153
 - syntax and options, GPRINT procedure 1149

- TC statement 1258
 - GREPLAY procedure, statements of
 - ? statement 1246
 - BYLINE statement 1247
 - CC statement 1247
 - CCOPY statement 1248
 - CDEF statement 1249
 - CDELETE statement 1250
 - CMAP statement 1250
 - COPY statement 1251
 - DELETE statement 1251
 - DEVICE statement 1251
 - FS statement 1252
 - GOUT statement 1252
 - GROUP statement 1253
 - IGOUT statement 1253
 - LIST statement 1254
 - MODIFY statement 1254
 - MOVE statement 1255
 - NOBYLINE statement 1256
 - PREVIEW statement 1256
 - QUIT statement 1257
 - REPLAY statement 1257
 - TCOPY statement 1258
 - TDEF statement 1259
 - TDELETE statement 1262
 - TEMPLATE statement 1262
 - TREPLAY statement 1263
 - windows for, using 1264
 - GRID option
 - BUBBLE statement 1095
 - PLOT statement, G3D procedure 1303
 - PLOT statement, GCONTOUR procedure 893
 - PLOT statement, GPLOT procedure 1108
 - SCATTER statement, G3D procedure 1307
 - GRID statement, G3GRID procedure 1333
 - spline interpolation, partial (example) 1342
 - spline interpolation, with smoothed spline (example) 1339
 - spline interpolation (example) 1343
 - using default interpolation method (example) 1337
 - group brackets, bar charts 130
 - group heading
 - pie and donut charts 826, 833
 - star charts 839, 842
 - GROUP= option
 - BLOCK statement 790
 - HBAR and VBAR statements 805
 - PIE and DONUT statements 824
 - STAR statement 837
 - GROUP statement, GREPLAY procedure 1253
 - GROUP variable, Annotate facility 647
 - grouping abbreviations 141
 - grouping and subgrouping
 - block chart (example) 844
 - pie charts (example) 877
 - pie or donut chart (example) 848
 - vertical bar chart (example) 848
 - GRSEG catalog entries 1239
 - GSET functions, DSGI 1462
 - GSF (graphics stream file) 60, 64
 - assigning 72
 - closing 290
 - how output is written to 306
 - output format and destination 296
 - prompt messages to 308
 - protocol module, specifying 303
 - record length 305
 - saving graphics to files 64, 66, 68
 - where written, specifying 307
 - GSFLEN= graphics option 305
 - GSFMODE= graphics option 64, 306, 459
 - animated sequences, creating 458
 - GSFMODE= option, GDEVICE procedure 306
 - GSFNAME= graphics option 64, 307, 459
 - GSFNAME= option, GDEVICE procedure 307
 - GSFPROMPT= graphics option 308
 - GSIZE= graphics option 308
 - GSLIDE procedure 601, 1277
 - Annotate graphics, displaying 1278, 1283
 - data-dependent coordinates 1281
 - instead of GANNO procedure 707
 - producing text slides (example) 1282
 - syntax and options 1279
 - GSPACE= option, HBAR and VBAR statements 806
 - GSTART= graphics option 309
 - GSTART= option, GDEVICE procedure 309
 - Gstart window (GDEVICE) 933
 - GTERM function (DSGI) 1403
 - GTESTIT procedure 1285
 - LOG window 1289
 - managing colors list for device driver (example) 1291
 - syntax and options 1290
 - testing GOPTIONS statement (example) 1291
 - GTITLE= option, ODS HTML statement 492
 - GUNIT= graphics option 40, 310
 - GWAIT= graphics option 50, 310
 - GWRITER= graphics option 311
- ## H
- H= option
 - AXIS statement 136, 139
 - GFONT procedure 944
 - LABEL= option, DONUT statement 829
 - LEGEND statement options 158
 - POINTLABEL= specification 197
 - SYMBOL statement 187
 - TITLE, FOOTNOTE, and NOTE statements 218
 - HANDSHAKE= graphics option 311
 - HANDSHAKE= option, GDEVICE procedure 311
 - handshaking 311, 325
 - hardware fonts 76, 78
 - alternative 80
 - default 78, 268, 270
 - device map, specifying 280
 - scaling in graphics output 274, 350, 351
 - specifying for device 269, 292
 - when not found 351
 - hardware-generated graphics
 - circles and arcs 271
 - dashed lines 277
 - line thickness 323
 - pie filling 334
 - plot symbols 354
 - polygon-fill 336
 - rectangle-fill 292, 345
 - vertices, maximum drawn 324
 - hardware-oriented color schemes 95
 - hardware patterns 176
 - HAXIS= option
 - BUBBLE statement 1095
 - PLOT statement, GCONTOUR procedure 886, 893
 - PLOT statement, GPLOT procedure 1108
 - HBAR and HBAR3D statements
 - bar chart with sum statistic (example) 846
 - error bars in horizontal bar chart (example) 854
 - GAREABAR procedure 728
 - GCHART procedure 796, 1521
 - midpoints and statistics (example) 851
 - HBX= graphics option 312
 - HEADER= option, GDEVICE procedure 313
 - HEADER records 313
 - HEADERFILE= option, GDEVICE procedure 313
 - headers for animation sequences 458
 - HEIGHT= macro argument 537
 - HEIGHT= option
 - AXIS statement 136, 139
 - CHART statement, GRADAR procedure 1190
 - GFONT procedure 944
 - LABEL= option, DONUT statement 829
 - LEGEND statement options 158
 - POINTLABEL= specification 197
 - SYMBOL statement 187
 - TITLE, FOOTNOTE, and NOTE statements 218
 - HELPLOCATION= graphics option 476
 - HELPLOCATION= parameters, JAVA and ActiveX 432
 - HEX option, GFONT procedure 946, 951
 - hexadecimal character values 81
 - hexadecimal values for font characters, displaying 946, 951
 - high-low plots 8, 189
 - HINDIC= macro argument, DS2CSF macro 563
 - HITEXT= graphics option 225
 - HLABEL= option, CHART statement 1190
 - HLS color scheme 96, 97, 98
 - converting to RGB 103
 - %HLS2RGB macro 102
 - HM= option
 - BUBBLE statement 1095
 - PLOT statement, GCONTOUR procedure 894
 - PLOT statement, GPLOT procedure 1109
 - HMINOR= option
 - BUBBLE statement 1095
 - PLOT statement, GCONTOUR procedure 894
 - PLOT statement, GPLOT procedure 1109
 - HONORASPECT= parameter, JAVA 432
 - HORIGIN device parameter 35
 - HORIGIN= graphics option 314
 - HORIGIN= option, GDEVICE procedure 314
 - horizontal bar charts 5
 - basics 775
 - error bars in (example) 854
 - midpoints and statistics in (example) 851
 - statistics in, displaying 815
 - terms used with 778
 - horizontal resolution, device 35
 - host commands, executing
 - after driver initialization 286
 - after graph production 337

- before graph production 339
- Host Commands window (GDEVICE) 934
- Host File Options window (GDEVICE) 933
- HOSTSPEC= option, GDEVICE procedure 314
- HPLJxxxx drives, patterns 181
- HPOS function (DSGI) 1417, 1474
- HPOS= graphics option 36, 315, 1151
 - aspect ratio 1152
- HREF attribute 259
- HREF= option
 - BUBBLE statement 1095
 - PLOT statement, GCONTOUR procedure 894
 - PLOT statement, GPLOT procedure 1109
- HREVERSE option
 - BUBBLE statement 1096
 - PLOT statement, GCONTOUR procedure 894
 - PLOT statement, GPLOT procedure 1029
- HSIZE device parameter 35
- HSIZE function (DSGI) 1418, 1475
- HSIZE= graphics option 35, 316, 1150
- HSIZE= option, GDEVICE procedure 316
- %HSL macro 101
- HSPACE 537
- HSV color scheme 96, 98
- %HSV macro 102
- HSYS variable, Annotate facility 650
- HTEXT= graphics option 316
- HTITLE= graphics option 225, 317
- HTML character entities 582
- HTML destination, ODS 489
- HTML drill-down mode, Java 401, 410, 419, 572
- HTML files, creating with ODS HTML (example) 245
- HTML function (DSGI) 1419, 1476
- HTML= option
 - BAR statement 758
 - BLOCK statement, GCHART procedure 791
 - BLOCK statement, GMAP procedure 1013
 - CHART statement, GRADAR procedure 1190
 - CHORO statement 1020
 - data tips, adding 570
 - drop-down links 574
 - GCHART procedure 259
 - HBAR and VBAR statements 806
 - PIE and DONUT statements 824
 - PLOT statement 766
 - PLOT statement, GPLOT procedure 1109
 - PRISM statement 1026
 - STAR statement 837
- HTML pages
 - bar chart with drill-down (example) 255
 - combining graphs and reports (example) 248
- HTML variable, Annotate facility 651
- HTMLFILE= macro argument 544
- HTMLFREF= macro argument 544, 564
- HTML_LEGEND= option
 - BLOCK statement, GCHART procedure 791
 - BLOCK statement, GMAP procedure 1013
 - CHART statement, GRADAR procedure 1190
 - CHORO statement 1020
 - drop-down links 574
 - HBAR and VBAR statements 806
 - PIE and DONUT statements 824
 - PLOT statement, GPLOT procedure 1109
 - PRISM statement 1026
- hyperlinks
 - drill-down, ActiveX 392
 - drill-down, Java 400
 - GIF output hotspots (example) 452
 - tables of contents 495
 - tables of pages 496
 - titles and footnotes as 220
- HZERO option
 - BUBBLE statement 1096
 - PLOT statement, GPLOT procedure 1110
- I**
- I= option, SYMBOL statement 187
- IBACK= graphics option 113, 318
 - images in radar charts (example) 1205
- IBACKLOG= macro argument 548
- IBACKPOS= macro argument 548
- IBACKURL= macro argument 549
- IBACKX=, IVBACKY= macro arguments 549
- IBM printers
 - external writes with 311
 - JES form name 302
 - JES SYSOUT destination 300
 - output class 297
- ID= option, GDEVICE procedure 318
- ID statement
 - GMAP procedure 1009, 1527
 - GPROJECT procedure 1172
 - GREDUCE procedure 1218
 - REMOVE procedure 1228
- identification variables 1005
- IFRAME= option 115
 - BUBBLE statement 1096
 - CHART statement, GRADAR procedure 1190
 - GSLIDE procedure 1280
 - HBAR and VBAR statements 806
 - PLOT statement, GPLOT procedure 1110
- IGOUT option
 - ? statement, GREPLAY procedure 1246
 - LIST statement, GREPLAY procedure 1254
- IGOUT= option, GREPLAY procedure 1244
- IGOUT statement, GREPLAY procedure 1253
- IMAGE function, Annotate facility 1540
- IMAGE function, DSGI 1452
- image maps 259
- IMAGE= option, PATTERN statement 116, 171
- IMAGEMAP= option
 - GANNO procedure 709, 719
 - GBARLINE procedure 750
 - GCHART procedure 786
 - GMAP procedure 1008
 - GPLOT procedure 1089
 - GREPLAY procedure 1244
 - GSLIDE procedure 1280
- IMAGEPOSX= parameter, JAVA 432
- IMAGEPRINT GOPTIONS statement 319
- images 106
 - Annotate facility to draw 625
 - as graph background 318, 319
 - as pattern fills 171
 - background 113
 - backplane 115
 - bar charts 817
 - bar line charts 748, 765
 - disabling as output 319
 - displaying with Annotate facility 118
 - displaying with DSGI 119, 1452
 - file types, list of 106
 - in star charts (example) 1205
 - in text slides 1281
 - interlacing 320
 - on chart bars 116
 - reading 107
 - transparent 357
 - writing 107, 110
- IMAGESTYLE= graphics option 114, 319
- IMAGESTYLE= option
 - BUBBLE statement 1096
 - CHART statement, GRADAR procedure 1191, 1205
 - GSLIDE procedure 1280
 - HBAR and VBAR statements 806
 - PATTERN statement 171
 - PLOT statement, GPLOT procedure 1110
- IMGPATH variable, Annotate facility 652, 662
- importing graphics 969
 - adjusting graphics output (example) 979
 - basics of 970
 - creating and importing CGM (example) 977
 - mapping colors 971
 - resizing graphics 975
- INBORDER option, CHART statement 1191
- INCOMPLETE option, GCONTOUR procedure 887, 889
- INDICTYP= macro argument, DS2CSF macro 563
- INHEIGHT= option, CHART statement 1191
- initializing drivers, executing before 285
- input (user), enabling 360
- INSERT function (DSGI) 1460
- INSIDE= option
 - BAR statement 758
 - HBAR and VBAR statements 807
- installation of SAS/GRAPH software, testing 1285
 - managing colors list for device driver (example) 1291
 - testing GOPTIONS statement (example) 1291
- installing ActiveX controls 389
- installing Java plug-in 424
- integer-based font rendering 289
- interactive Metagraphics output 357, 469
 - character rotation angle 350
 - description string 318
 - enhancing Web presentations for 474
 - hardware text rotation angle 330
 - interactivity of 320
 - META2HTM macro with 471
 - negative handshaking response 325
 - ODS with 470
 - run-time controls 471
 - sample programs 478
 - TRAILER records 356
 - translating metafile into device commands 339
 - user-written part, files for 340, 341
- INTERACTIVE= option, GDEVICE procedure 320
- INTERCHART= option, CHART statement 1191
- interface drivers 42
- INTERLACED GDEVICE procedure 321

INTERLACED GOPTIONS statement 321
interlacing images 320
internal coordinates, Annotate facility 598, 678
internationalization
 ActiveX and 392
 Java and 400
 Metaview Applet 474
INTERPOL= graphics option 321
INTERPOL= option, SYMBOL statement 185, 187
interpolation
 bar line charts 740
 box plots 185, 187, 201, 233
 connecting data points with straight lines 190
 contour plots 887
 data value inclusion 197
 default method, specifying 205
 default value for 321
 high-low plots 189
 language 190
 needle plots 191
 regression analysis 192
 regression analysis plots 192
 smoothing plot lines 190
 spline interpolation 194, 205, 1330
 spline interpolation, example 1339, 1342, 1343
 step plots 196
interpolation methods
 default interpolation method, GRID statement (example) 1337
 G3GRID procedure 1329
 plots 1085
 spline interpolation, partial (example) 1342
 spline interpolation, with smoothed spline (example) 1339
 spline interpolation (example) 1343
INTERTILE= option, CHART statement 1191, 1201
INTERVAL= option, AXIS statement options 127
INVISIBLE= option, PIE and DONUT statements 824
ITERATION= graphics option 321, 459

J

J= option
 AXIS statement options 137
 LABEL= option, DONUT statement 830
 LEGEND statement options 158
 POINTLABEL= specification 197
 TITLE, FOOTNOTE, and NOTE statements 218, 225
Java applets 370, 371, 385, 419
 authentication 583
 CLASSPATH environmental variables 583
 drill-down example (Local mode) 415
 drill-down example (Script mode) 416
 drill-down example (URL mode) 417
 drill-down links, configuring 400
 generating output for 398
 interactivity for 397
 internationalization 400
Java archive files
 location of 422

JAVA device driver 381, 398
 data tips for 568
 drill-down links in presentations 571
Java parameters and attributes 421
Java plug-in
 installing 424
 location of 424
Java support 1508
 Annotate functions 1539
 G3D procedure 1537
 GAREABAR procedure 1518
 GBARLINE procedure 1519
 GCHART procedure 1521
 GCONTOUR procedure 1526
 GMAP procedure 1527
 GOPTIONS statement 1510
 GPLOT procedure 1530
 GRADAR procedure 1535
 LEGEND statement 1514
 PATTERN statement 1515
 SYMBOL statement 1516
 TITLE and FOOTNOTE statements 1517
JAVAIMG device driver 377, 381
 data tips for 568
 drill-down links in images 571
 GIF, JPEG, PNG vs. 440
 Web presentations, developing 442
JAVAMETA device driver 382, 469, 470
 enhancing Web presentations for 474
 META2HTM macro with 471
 run-time controls 471
 sample programs 478
JavaScript callback method 407
JOIN option
 GRID statement, G3GRID procedure 1334
 PLOT statement, GCONTOUR procedure 894
joins, contour plots (example) 910
JPEG device driver 382
 ACTXIMG, JAVAIMG vs. 440
 data tips for 568
 developing web presentations 443
 drill-down links in images 571
 HTML files, generating 445
 names for image output files 445
JPEG presentations 377
JSTYLE option, PIE and DONUT statements 824
justification
 axis labels 127, 133, 135
 donut chart labels 830
 legend label 153
 legend text 158
 legend values 157
 plot print labels 197
 text in graphics output 218
JUSTIFY= option
 AXIS statement options 137
 LABEL= option, DONUT statement 830
 LEGEND statement options 158
 POINTLABEL= specification 197
 TITLE, FOOTNOTE, and NOTE statements 218, 225

K

kern data sets 958
 creating 959
 specifying 949
 variables for, list of 959
KERN= option, GFONT procedure 949
KERNDATA= option, GFONT procedure 949, 959
KEYMAP= graphics option 321, 986
KEYMAP option, GKEYMAP procedure 989
keymaps (mapping characters to keys) 983
 asymmetrical 986
 basics of 983
 creating and using 990
 GKEYMAP data sets 987
 GKEYMAP data sets, generating 990
 ignoring 945, 949
 modifying (example) 990

L

L option, GOPTIONS procedure 1078
L= option, SYMBOL statement 196, 207
LA= option, TITLE, FOOTNOTE, and NOTE statements 219
LABEL function, Annotate facility 1541
%LABEL macro, Annotate facility 685
LABEL= option
 AXIS statement 1509
 AXIS statement options 127
 DONUT statement 824, 1525
 LEGEND statement 1515
 LEGEND statement options 153
LABEL statement 28
 ActiveX 392
 Java applets, internationalization and 400
LABELPOS= macro argument, DS2CSF macro 563
labels
 axes 127, 133
 bubbles in bubble plots 1092
 BY lines 143
 contour lines 199
 contour plot lines (examples) 906
 contour plots 903
 donut charts 829
 legends 153
 pie chart slices 830, 873
 plot bubbles (example) 1122
 plot points 197
 size of 1190
 splines in star charts 1193
 star charts 840
LABELS= macro argument 538, 561
lakes, displaying in maps 1041
Lambert's conformal projections 1166, 1172
landscape orientation 34, 322, 323, 349
language, interpolation 190
language elements, SAS/Graph 26
LANGUAGE= option, TITLE, FOOTNOTE, and NOTE statements 219
languages
 ActiveX and 392
 Java and 400
LAST= option, CHART statement 1191

- LAT variable (map data sets) 1000
- latitude coordinates 1164
- LATMAX= option, GPROJECT procedure 1169
- LATMIN= option, GPROJECT procedure 1169
- LAUTOHREF= option
 - BUBBLE statement 1096
 - PLOT statement, GCONTOUR procedure 894
 - PLOT statement, GPLOT procedure 1110
- LAUTOREF= option
 - BAR statement 758
 - HBAR and VBAR statements 807
- LAUTOVREF= option
 - BUBBLE statement 1096
 - PLOT statement, GCONTOUR procedure 894
 - PLOT statement, GPLOT procedure 1110
- LAYOUT= macro argument 538
- LCOLFMT= macro argument 538
- LCOLOR= macro argument 538
- LCOLS device parameter 36, 1151
- LCOLS= option, GDEVICE procedure 322
- LCOLVAL= macro argument 538
- LDATA= macro argument 538
- LEFTMARGIN GDEVICE procedure 316
- LEFTMARGIN GOPTIONS statement 316
- LEGEND and LEGEND= options
 - BLOCK statement, GCHART procedure 791
 - BLOCK statement, GMAP procedure 1013
 - CHORO statement 1020
 - GOPTIONS procedure 1078
 - HBAR and VBAR statements 807
 - PIE and DONUT statements 825
 - PLOT statement, GCONTOUR procedure 894
 - PLOT statement, GPLOT procedure 1110
 - PRISM statement 1026
- LEGEND statement 27, 124, 151, 318
 - ActiveX and Java support for 1514
 - filling areas in overlay plot (example) 1134
 - generating overlay plot (example) 1131
 - legends for patterns and midpoints (example) 875
- LEGENDFONT= parameter, JAVA 432
- LEGENDHEIGHTPERCENT= parameter, JAVA 432
- LEGENDIT= parameter, JAVA 432
- LEGENDPERCENT= parameter, JAVA 432
- legends 39
 - drop shadows 326
 - formatting 150
 - offset 154, 163
 - origins 155, 163
 - pie chart patterns and midpoints (example) 875
 - plots with three variables 1118
 - spacing around 154, 163
- LEGENDWIDTHPERCENT= parameter, JAVA 432
- LENGTH= option, AXIS statement options 127
- LEVELOFDDETAIL= parameter, JAVA 432
- LEVELS= option
 - BAR statement 758
 - BLOCK statement, GCHART procedure 791
 - BLOCK statement, GMAP procedure 1014, 1047
 - CHORO statement 1020
 - HBAR and VBAR statements 808
 - PIE and DONUT statements 825
 - PLOT statement, GCONTOUR procedure 895, 899, 908
 - PRISM statement 1026
 - STAR statement 838
- LFACTOR= graphics option 323
- LFACTOR= option, GDEVICE procedure 323
- LFRAME= option, GSLIDE procedure 1280, 1281
- LFROM= macro argument 539
- LH= option
 - BUBBLE statement 1096
 - PLOT statement, GCONTOUR procedure 895
 - PLOT statement, GPLOT procedure 1111
- LHREF= option
 - BUBBLE statement 1096
 - PLOT statement, GCONTOUR procedure 895
 - PLOT statement, GPLOT procedure 1111
- LIBNAME statements 28, 29, 30
- librefs 30
- LIFO stack 603, 639
- light source coordinates, prism maps 1028
- LIGHTING= parameter, JAVA 433
- LINCOLOR function (DSGI) 1419, 1477
- LINE function (DSGI) 1453
- %LINE macro, Annotate facility 686
- LINE= option, SYMBOL statement 196, 207
- line plots 7
- line segments for drawing fonts 941, 956
- line smoothing 190
 - language, interpolation 190
 - spline interpolation 194, 205, 1330
 - spline interpolation, example 1339, 1342, 1343
- line types
 - axis 134
 - default line thickness 323
 - plots 196, 207
 - spokes in star charts 1191, 1208
- LINE variable, Annotate facility 653
- lines
 - dashed, hardware-generated 277
 - dashed, length of 278
 - displaying with DSGI 1453
 - in graphics output area 217
 - testing ability to draw 1286
- lines, drawing with Annotate facility 622
- LINESIZE= option 1150, 1151
- LININDEX function (DSGI) 1420, 1477
- LINK element (HTML) 555
- LINK= option, TITLE, FOOTNOTE, and NOTE statements 220
- link variables in Web presentations 574
- linking to output
 - Tables of Contents 495
 - Tables of Pages 496
- LINKTYPE= macro argument 539
- LINREP function (DSGI) 1421, 1478
- LINTYPE function (DSGI) 1422, 1479
- LINWIDTH function (DSGI) 1422, 1480
- LIST statement, GDEVICE procedure 926
- LIST statement, GREPLAY procedure 1254
- listing destination, ODS 489
- LLEVELS= option, PLOT statement 895, 903, 908
- LLX= and LLY= options, TDEF statement 1260
- LOADFUNC= parameter, JAVA 433
- Local drill-down mode, Java 400, 401, 572
 - default behavior 402
 - example 415
 - levels of, customizing 405
- local fonts 78
- local statements, RUN-group processing 33
- LOCALE= parameter, JAVA 433
- locking data sets automatically 31
- LODCOUNT= parameter, JAVA 433
- log, writing in (DSGI) 1454
- log messages, waiting to display 345
- LOG window, GTESTIT procedure 1289
- logarithmic axes 127, 229
 - block charts 815
 - plots 1087
- LOGBASE= option, AXIS statement 127, 128, 229
- LOGRESOURCES= graphics option 476
- LOGRESOURCES parameter, Metaview Applet 475
- LOGSTYLE= option, TITLE, FOOTNOTE, and NOTE statements 229
- LONG variable (map data sets) 1000
- longitude coordinates 1164
- LONGMAX= option, GPROJECT procedure 1169
- LONGMIN= option, GPROJECT procedure 1169
- looping animation 321
- LP variable (font data sets) 953
- LPT= macro argument 539
- LR= option
 - BAR statement 758
 - HBAR and VBAR statements 808
- LREF= option
 - BAR statement 758
 - HBAR and VBAR statements 808
- LROWS device parameter 36, 1151
- LROWS= option, GDEVICE procedure 323
- LRX= and LRY= options, TDEF statement 1260
- LS= option, TITLE, FOOTNOTE, and NOTE statements 221
- LSPACE= option, TITLE, FOOTNOTE, and NOTE statements 221
- text break and 225
- LSPOKE= option, CHART statement 1191
- LSTAR= and LSTARS= options, CHART statement 1191, 1208
- LSTIP= macro argument 539
- LSTIPFAC= macro argument 539
- LTIP= macro argument 539
- LTIPFMT= macro argument 539
- LTO= macro argument 539
- LV= option
 - BUBBLE statement 1096
 - PLOT statement, GCONTOUR procedure 896
 - PLOT statement, GPLOT procedure 1111
- LVALUE= macro argument 540
- LVREF= option
 - BUBBLE statement 1096
 - PLOT statement, GCONTOUR procedure 896
 - PLOT statement, GPLOT procedure 1111
- LWHERE= macro argument 540
- LWIDTH= macro argument 540

- M**
- M= option, TITLE, FOOTNOTE, and NOTE statements 221, 225
 - macro variables, names for 566
 - macros, Web output 369
 - MAJOR= option, AXIS statement 129, 1509
 - major tick marks 129, 138
 - formatting 139
 - offset 130
 - scatter plots 1310
 - suboptions, list of 1509
 - surface plots 1304
 - with datetime values (example) 226
 - MAKEHTML= macro argument 544
 - MAKEXML= macro argument 545
 - Map applet 372, 419
 - drill-down tags 412
 - parameters for, list of 424
 - Script drill-down example 416
 - map areas 1005
 - clipping (example) 1178
 - combining by removing internal boundaries 1223, 1226, 1227, 12
 - emphasizing (example) 1177
 - unmatched, GREDUCE procedure with 1215
 - unmatched, GREMOVE procedure with 1215
 - map data sets 23, 999
 - creating 1041
 - customizing 1039
 - density values 1218
 - importing ESRI files as 1347, 1348, 1349
 - lakes, displaying 1041
 - merging feature tables with 1001, 1069
 - ordering observations 1227
 - outline map of Africa, creating (example) 1232
 - projecting 1040
 - reference information 1038
 - removing internal boundaries 1223, 1226, 1227, 12
 - removing U.S. state boundaries (example) 1228
 - response data sets 1001, 1003
 - specialty 1003
 - subsetting or reducing (clipping) 1039, 1173, 1178, 12
 - traditional map data sets 999
 - viewing 1001
 - map data sets, projecting coordinates from spherical to Cartesian 1161
 - basic usage of GPROJECT procedure 1172
 - clipping map areas (example) 1178
 - default projection specifications, using (example) 1174
 - emphasizing map areas (example) 1177
 - ID statement, GPROJECT procedure 1172
 - input map data sets 1163
 - projecting an Annotate data set (example) 1180
 - syntax and options, GPROJECT procedure 1168
 - types of map projections 1165
 - MAP element (HTML) 260
 - MAP= option, GMAP procedure 1009
 - MAP statement, GIMPORT procedure 974
 - MAPIMPORT procedure 1347, 1348
 - examples 1349
 - %MAPLABEL macro, Annotate facility 687
 - mapping characters to keyboard 983
 - asymmetrical 986
 - basics of 983
 - creating and using 990
 - GKEYMAP data sets 987
 - GKEYMAP data sets, generating 990
 - ignoring 945, 949
 - modifying (example) 990
 - mapping colors 971
 - mapping fonts 972
 - mapping patterns 971
 - maps 11, 996
 - assigning formats to response variables (example) 1049
 - creating maps using feature tables (example) 1069
 - default projection specifications, using (example) 1174
 - drill-down functionality, ActiveX 392
 - drill-down functionality in maps (example) 1054
 - emphasizing map areas (example) 1177
 - feature tables 1001
 - FIPS codes 1033
 - predefined formats for 1035
 - producing simple block map (example) 1045
 - producing simple choropleth map (example) 1053
 - producing simple prism map (example) 1063
 - producing simple surface map (example) 1066
 - response levels 1004
 - rotating and tilting surface map (example) 1068
 - SAS Maps Online 1006
 - specialty map data sets 1003
 - specifying midpoints in prism map 1065
 - specifying response levels in block map 1047
 - MARCOLOR function (DSGI) 1423, 1480
 - margins, graphics output area 34
 - MARINDEX function (DSGI) 1424, 1481
 - Marker font 87
 - MARREP function (DSGI) 1424, 1482
 - MARSIZE function (DSGI) 1425, 1483
 - MARTYPE function (DSGI) 1426, 1483
 - MATCHCOLOR option
 - PIE and DONUT statements 825
 - STAR statement 838
 - math font 85
 - MAXCOLORS device parameter 105
 - MAXCOLORS= option, GDEVICE procedure 324
 - MAXDISP function (DSGI) 1427
 - MAXHIDE suboption, AUTOLABEL= option 898
 - maximum, font 940
 - MAXIS= option
 - BAR statement 759
 - HBAR and VBAR statements 808
 - MAXNVERT= option, CHART statement 1192
 - MAXPOLY= option, GDEVICE procedure 324
 - MAXVERT= option, CHART statement 1192
 - MEAN option
 - BAR statement 759
 - HBAR and VBAR statements 808
 - mean statistic 746, 783
 - MEANLABEL option, HBAR and VBAR statements 808
 - memory, open software fonts 290
 - MENUREMOVE= parameter, JAVA 433
 - MERGE statement 1069
 - MESSAGE function (DSGI) 1485
 - message queuing 345
 - messages, writing in, DSGI for 1455
 - META2HTM macro 384
 - arguments of 478, 535
 - arguments of, applet behavior 565
 - arguments of, page formatting 552
 - arguments of, saving HTML files 564
 - embedding multiple instances of Metaview Applet (example) 483
 - producing Web presentation (example) 481
 - stylesheets, macro arguments for 554
 - metacodes 469
 - outputting with HTML from ODS (example) 478
 - page-selecting slider control 472
 - slide-show control 473
 - METACODES= graphics option 476
 - metacodes parameter, Metaview Applet 472
 - metacodes zoom control 473
 - METACODESLABEL= graphics option 476
 - Metagraphics device drivers 73
 - color space specification 273
 - header generation 313
 - metacode file format 294
 - patterns 181
 - Metagraphics output, interactive 357, 469
 - character rotation angle 350
 - description string 318
 - enhancing Web presentations for 474
 - hardware text rotation angle 330
 - interactivity of 320
 - META2HTM macro with 471
 - negative handshaking response 325
 - ODS with 470
 - run-time controls 471
 - sample programs 478
 - TRAILER records 356
 - translating metafile into device commands 339
 - user-written part, files for 340, 341
 - Metagraphics window (GDEVICE) 932
 - Metaview applet 375, 469, 470
 - data tips with 569
 - drill-down functionality 572
 - enhancing Web presentations for 474
 - META2HTM macro with 471
 - multiple instances on HTML page (example) 483
 - non-English resources and fonts 474
 - parameters, list of 475
 - run-time controls 471
 - sample programs 478
 - Microsoft Excel, embedded graphs in (example) 395
 - Microsoft Word, embedded graphics in (example) 393, 395
 - MIDPOINT variable, Annotate facility 654
 - midpoints, bar variables 741, 742, 759
 - selecting and ordering 744, 763

- midpoints, chart variables 778, 780
 - horizontal bar chart (example) 851
 - pie charts (example) 875
 - selecting and ordering 781, 816
 - MIDPOINTS= option
 - BAR statement 759
 - BLOCK statement, GCHART procedure 791
 - BLOCK statement, GMAP procedure 1014
 - CHORO statement 1021
 - GCHART procedure statements 782
 - HBAR and VBAR statements 809, 816
 - PIE and DONUT statements 826
 - PRISM statement 1027
 - STAR statement 838
 - MINILEGENDFONTSIZE= parameter,
 - JAVA 433
 - minimum, font 940
 - MINLNKWT= macro argument 540
 - MINOR= option
 - AXIS statement 129, 1509
 - BAR statement 760
 - HBAR and VBAR statements 810
 - PLOT statement 767
 - minor tick marks 129, 138
 - formatting 139
 - suboptions, list of 1509
 - with datetime values (example) 226
 - MISSING option
 - BAR statement 747, 760
 - BLOCK statement, GCHART procedure 792
 - BLOCK statement, GMAP procedure 1015
 - CHART statement, GRADAR procedure 1192
 - CHORO statement 1021
 - GCHART procedure statements 779
 - HBAR and VBAR statements 810
 - PIE and DONUT statements 826
 - PRISM statement 1027
 - STAR statement 838
 - missing values
 - GBARLINE procedure 747
 - GCHART procedure 779
 - GCONTOUR procedure 887, 889
 - in Annotate data sets 601
 - plot data sets 1087, 1112
 - star charts 1192
 - MISSINGCOLOR= parameter, JAVA 434
 - MODE= option
 - CHART statement, GRADAR procedure 1192, 1209
 - LEGEND statement 154, 163
 - SYMBOL statement 197
 - model number, output device 324
 - MODEL= option, GDEVICE procedure 324
 - MODIFY statement
 - custom device entry, creating (example) 936
 - GDEVICE procedure 926
 - GREPLAY procedure 1254
 - MODULE= option, GDEVICE procedure 325
 - MOVE function, Annotate facility 627, 1541
 - %MOVE macro, Annotate facility 687
 - MOVE= option, TITLE, FOOTNOTE, and NOTE statements 221, 225
 - MOVE statement, GREPLAY procedure 1255
 - MULTFONT option, GKEYMAP procedure 990
 - multiline
 - axis values 134
 - legend labels 159
 - multiple classification variables in radar charts (example) 1202
 - Music font 88
 - MWIDTH= option, GFONT procedure 949
- ## N
- N= option
 - AXIS statement options 140
 - GFONT procedure 943, 946
 - SURFACE statement 1032
 - N option, GOPTIONS procedure 1078
 - N1=, ..., N5= options, GREDUCE procedure 1217
 - NACTION= macro argument 540
 - NAK= option, GDEVICE procedure 325
 - NAME= macro argument 537
 - NAME= option 63
 - BLOCK statement, GCHART procedure 792
 - BLOCK statement, GMAP procedure 1015
 - BUBBLE statement 1097
 - CHART statement, GRADAR procedure 1192
 - CHORO statement 1021
 - GANNO procedure 709, 715
 - GFONT procedure 943, 946
 - GKEYMAP procedure 989
 - GPRINT procedure 1149
 - GSLIDE procedure 1280
 - HBAR and VBAR statements 729, 810
 - PIE and DONUT statements 826
 - PLOT statement, G3D procedure 1303
 - PLOT statement, GCONTOUR procedure 896
 - PLOT statement, GPLOT procedure 1111
 - PRISM statement 1027
 - SCATTER statement, G3D procedure 1307
 - STAR statement 838
 - SURFACE statement 1032
 - TREPLAY statement, GREPLAY procedure 1263
 - NAME= parameter, JAVA 434
 - names
 - anchors 168
 - Annotate facility 591, 599, 602
 - BY line catalog entries 143
 - catalog entries 55
 - colors 95, 99
 - data sets 30
 - device entries 928
 - device nicknames 299
 - executable driver modules 325
 - filename extensions 63, 288
 - fonts 76, 80, 81
 - graphics output 63
 - image output files 445
 - macro variables 566
 - paper type 330
 - predefined graph styles (ODS) 489
 - natural device drivers 42
 - NAVIGATERENDERMODE= parameter,
 - JAVA 434
 - NAXIS1= and NAXIS2= options, GRID statement 1334, 1336
 - NB option, GFONT procedure 944
 - NCOL= and NCOLS= options, CHART statement 1192
 - NCOLVAL= macro argument 541
 - ND option, GFONT procedure 949
 - NDATA= macro argument 541
 - NEAR= option, GRID statement 1334
 - needle plots 191
 - negative values, block charts 795
 - Netscape colors, troubleshooting 583
 - NEWFILE= option, ODS HTML statement 492
 - NEWNAME= option, RENAME statement 928
 - NFNTNAME= macro argument 541
 - NFNTSIZE= macro argument 541
 - NFNTSTYL= macro argument 541
 - NID= macro argument 541
 - NLABEL= macro argument 541
 - NLEVELS= option, PLOT statement 896
 - NLINES option, SURFACE statement 1032
 - NOADMGDF option 262
 - NOAUTOCOPY option 263
 - NOAUTOFEED option 264
 - NOAXES option
 - BUBBLE statement 1097
 - PLOT statement, G3D procedure 1303
 - PLOT statement, GCONTOUR procedure 897
 - PLOT statement, GPLOT procedure 1111
 - SCATTER statement, G3D procedure 1307
 - NOAXIS option
 - BAR statement 760
 - BUBBLE statement 1097
 - HBAR and VBAR statements 810
 - PLOT statement, G3D procedure 1303
 - PLOT statement, GCONTOUR procedure 897
 - PLOT statement, GPLOT procedure 1111
 - SCATTER statement, G3D procedure 1307
 - NOBASEREF option
 - BAR statement 760
 - HBAR and VBAR statements 810
 - NOBOT= option, ODS HTML statement 167
 - NO_BOTTOM_MATTER= option, ODS HTML statement 167
 - NOBRACKETS option, AXIS statement options 130
 - NOBUILD option, GFONT procedure 944
 - NOBYLINE option, GREPLAY procedure 1244, 1256
 - NOCC option, GPRINT procedure 1150
 - NOCELL option 268
 - NOCHARACTERS option 268
 - NOCIRCLEARC option 270
 - NOCOLLATE option 271
 - NOCONNECT option, STAR statement 839
 - NODASH option 277
 - node-link diagrams 373, 503, 513
 - chart with simple arcs (example) 518
 - chart with weighted arcs (example) 520
 - data tips with 569
 - drill-down functionality 572, 573
 - DS2CONST macro with 515
 - hotspots 524
 - when to use 514
 - XML written to external file (example) 522
 - NODEBDR= macro argument 549
 - NODESEP= macro argument 549
 - NODESHAP= macro argument 549
 - NODISPLAY option, GFONT procedure 949
 - NODISPLAY statement 284
 - NODRVQRY= option, GDEVICE procedure, executing before driver initialization 286
 - NOERASE= graphics option 287

- NOERASE= option, GDEVICE procedure 287
 NOFASTTEXT= graphics option 288
 NOFILEONLY= graphics option 291
 NOFILL= graphics option 292
 NOFILL= option, GDEVICE procedure 292
 NOFR option
 BAR statement 757
 BUBBLE statement 1095
 HBAR and VBAR statements 804
 PLOT statement, GPLOT procedure 1108
 NOFRAME option
 BAR statement 757
 BUBBLE statement 1095
 CHART statement, GRADAR procedure 1190, 1192
 HBAR and VBAR statements 728, 804
 PLOT statement, GCONTOUR procedure 897
 PLOT statement, GPLOT procedure 1108
 NOFS option
 GDEVICE procedure 921
 GREPLAY procedure 1241, 1244
 NOGFOOTNOTE= option, ODS HTML statement 492
 NOGRAPHRC= graphics option 305
 NOGROUPHEADING option
 PIE and DONUT statements 826, 833
 STAR statement 839, 842
 NOGTITLE= option, ODS HTML statement 492
 NOHEADING option
 BLOCK statement 792
 PIE and DONUT statements 827, 832
 STAR statement 839, 842
 NOHEX option, GFONT procedure 945
 NOIMAGEPRINT GOPTIONS statement 120, 319
 NOIMAGEPRINT graphics option 120, 319
 NOJSOBJECT= parameter, JAVA 434
 NOKEYMAP option, GFONT procedure 945, 949
 NOLABEL option
 PLOT statement, G3D procedure 1303
 SCATTER statement, G3D procedure 1307
 NOLEGEND option
 BLOCK statement, GCHART procedure 793
 BLOCK statement, GMAP procedure 1015
 CHORO statement 1022
 HBAR and VBAR statements 810
 PIE and DONUT statements 827
 PLOT statement, GCONTOUR procedure 897
 PLOT statement, GPLOT procedure 1111
 PRISM statement 1028
 NOLINE option, PLOT statement 767
 NOLIST option, GOPTIONS procedure 1078, 1079
 NOLOG option, GOPTIONS procedure 1078
 NOMARKER option, PLOT statement 767
 non-roman alphabet fonts 85
 NONE font 77
 NONEEDLE option, SCATTER statement 1308
 NONINTERLACED GDEVICE procedure 321
 NONINTERLACED GOPTIONS statement 321
 NOPIEFILL GDEVICE procedure 334
 NOPIEFILL GOPTIONS statement 334
 NOPLANE option, AXIS statement options 130
 NOPROMPT= graphics option 50, 310
 NOROMAN option, GFONT procedure 945
 NOROMHEX option, GFONT procedure 945
 NOSCALE option, GRID statement 1335
 NOSTATS option, HBAR and HBAR3D statements 810
 NOTE statement 26, 124, 210, 224
 BY statement with 145
 RUN-group processing 33
 notes 211, 224
 angle of rotation 213, 219, 222
 boxes around 215, 216
 colors for 215, 216, 276
 default characteristics, setting 225
 defining text of 222, 226
 fonts for 217
 justification 218
 positioning 221
 size of 218, 316
 spacing around 221
 text breaks 225
 underlining 223
 NOTOP= option, ODS HTML statement 167
 NO_TOP_MATTER= option, ODS HTML statement 167
 NOTRANSAPRENCY GOPTIONS statement 357
 NOTRANSAPRENCY= graphics option 459
 NOTSORTED= option, BY statement 142, 1227
 NOUSERINPUT= graphics option 459
 NOZERO option
 BAR statement 760
 HBAR and VBAR statements 810
 NOZEROREF option, CHART statement 1192
 NPARENT= macro argument 542
 NPW= macro argument 542
 NR option, GFONT procedure 945
 NROW= and NROWS= options, CHART statement 1193
 NSCBACK= macro argument 542
 NSCTEXT= macro argument 542
 NSDATA= macro argument 542
 NSFNTNAM= macro argument 542
 NSFNTSIZ= macro argument 542
 NSFNTSTY= macro argument 542
 NSHAPE= macro argument 542
 NSID= macro argument 542
 NSIZE= macro argument 543
 NSPW= macro argument 543
 NSTYLE= macro argument 543
 NSWHERE= macro argument 543
 NTEXTCOL= macro argument 543
 NTIP= macro argument 543
 NTIPFMT= macro argument 543
 NUMBER= option, AXIS statement options 140
 numeric bar variables 742, 743
 numeric chart variables 779, 780, 781
 charting in star chart (example) 881
 numeric map variables 1004
 NUMGRAPH function (DSGI) 1428
 NURL= macro argument 543
 NVALUE= macro argument 544
 NWHERE= macro argument 544
 NX=, NY= macro arguments 544
- O**
 O option, GPRINT procedure 1150
 OBJECT element (HTML) 421
 observations
 in Annotate data sets 589
 ordering for input map data sets 1227
 ODS destinations 489
 ODS HTML statement 124, 164, 167
 bar chart with drill-down (example) 255
 destination, specifying 168
 multiple graphs and reports in Web page (example) 248
 Web page, creating (example) 245
 ODS output 487
 ACTXIMG device driver with (example) 447
 body files 491
 frames for 497
 GIF output, generating (example) 450
 graphic options with 493
 JAVAMETA driver with 470
 metacodes (example) 478
 non-graphics output on Web pages 494
 RUN-group processing 490
 static graphics 440
 Table of Contents 495
 Tables of Pages 496
 titles and footnotes, controlling 492
 ODS statements 28
 generating presentations 382
 JAVA and ActiveX parameters and attributes 421
 ODS RTF statement, graphics in Microsoft Word (example) 393
 ODS USEGOPT statement 493
 PARAMETERS= statement for applet parameters 477
 ODS styles 94, 488
 offset
 angle of rotation 222
 axes 130
 between Bitstream font letters 295
 between display area and graphic 360
 between displayed area and graph 314
 between fill lines 292
 between graphs and display 314, 360
 contour plot labels 904
 fonts 948, 951, 960, 961
 legend 154, 163
 legends 154, 163
 text in graphics output 216, 221
 OFFSET= option
 AXIS statement options 130
 LEGEND statement options 154, 163
 OFFSHADOW= graphics option 326
 online help, locations for 385
 open destinations, ODS 489
 OPENGRAPH function (DSGI) 1428
 OPENMODE= argument, META2HTM macro 564
 OPENMODE= macro argument 545
 OPTION= option, GOPTIONS procedure 1078
 OPTIONS statement 28
 ORDER= option
 AXIS statement options 130, 135, 747, 816,
 LEGEND statement options 155
 ORDERACROSS= option, CHART statement 1193
 ordering
 axis values 130

- legend values 155
- midpoints, bar variables 744, 763
- midpoints, chart variables 781, 816
- slices in pie charts (example) 873
- ORIGIN= option, AXIS statement options 133
- ORIGIN= option, LEGEND statement options 155, 163
- origins
 - axes 133
 - legends 155, 163
- OTHER= option
 - CHART statement, GRADAR procedure 1193
 - PIE and DONUT statements 827
- OTHERCOLOR= option, PIE and DONUT statements 827
- OTHERLABEL= option, PIE and DONUT statements 827
- out-of-range plot variables 747, 1087
- OUT= option
 - G3GRID procedure 1332
 - GKEYMAP procedure 989
 - GPROJECT procedure 1169
 - GREDUCE procedure 1217
 - GREMOVE procedure 1226
 - MAPIMPORT procedure 1348
- outline fonts 941
- OUTLINECOLOR= parameters, JAVA and ActiveX 434
- outlines
 - bar charts 816
 - block charts 794
 - colors 180
 - default 178
 - GBARLINE procedure 748, 764
 - GCHART procedure 784
 - outline map of Africa, creating (example) 1232
 - slice colors and patterns 831
 - star charts 840
- OUTLINES= parameter, JAVA 434
- output names 63
- output printer bins 326
- OUTSIDE= option
 - BAR statement 760
 - HBAR and VBAR statements 811
- OUTTRI= option, G3GRID procedure 1332
- overdrawing graphs 348
- OVERFLOWCOLOR= parameters, JAVA and ActiveX 434
- OVERLAY= option
 - CHART statement, GRADAR procedure 1193, 1199
 - PLOT statement, GPLOT procedure 1111, 1131, 1134
- overlay plots 1111, 1131, 1134
- overlying graphics, Annotate facility 602
- overlying radar charts (example) 1199
- OVERLAYVAR= option, CHART statement 1193
- overriding
 - colors list 94
 - device parameters 46
- overwriting files 70

P

- P option, GOPTIONS procedure 1078
- PAGE= argument, ODS HTML statement 166
- page files 496
- PAGE= option, ODS statements 497
- page-selecting slider control 472
- PAGECONTROLENABLED= graphics option 476
- PAGECTL= argument, META2HTM macro 565
- PAGEPART= macro argument 553
- PAGESIZE= option 1150, 1151
- paints, plot 190
- paper feed 264, 327
- paper size 328
- paper type, specifying 330
- PAPERDEST= graphics option 326
- PAPERFEED= graphics option 327
- PAPERFEED= option, GDEVICE procedure 327
- PAPERLIMIT= graphics option 328
- PAPERSIZE= graphics option 328
- PAPERSOURCE= graphics option 329
- PAPERTYPE= graphics option 330
- PARADIV= option, GPROJECT procedure 1170, 1172
- PARALEL1= and PARALEL2= options, GPROJECT procedure 1170, 1172
- parallels
 - calculating 1172
- parameters
 - JAVA and ActiveX 421, 427
- PARAMETERS= option, ODS statements 421
- PARAMETERS= statement, ODS statement 477
- Parameters window (GDEVICE) 930
- parametric language interpolation 190
- PARTIAL option, GRID statement 1335
- partial spline interpolation (example) 1342
- PATH= option, GDEVICE procedure 330
- PATH= option, ODS HTML statement 491
- PATREP function (DSGI) 1498, 1501
- PATTERN definitions, BY statement with 145
- PATTERN option
 - GOPTIONS procedure 1078
 - PLOT statement, GCONTOUR procedure 897, 899
- PATTERN statement 27, 124, 169
 - ActiveX and Java support for 1515
 - altering/canceling 177
 - GBARLINE procedure 748, 764
 - GCHART procedure 784
 - GPLOT procedure 1120
 - images on bar chart bars 116
- PATTERNID= option
 - BAR statement 761, 764
 - BLOCK statement 793, 795
 - BY line 144
 - HBAR and VBAR statements 811, 817
- patterns and fills 169
 - bar charts 171, 816
 - block charts 794
 - block maps 1016
 - built-in pie-fill capability 334
 - built-in polygon-fill capability 336
 - built-in rectangle capability 345
 - built-in rectangle-fill capability, device 292
 - color for 274
 - contour plots 173
 - contour plots (example) 910
 - default 177
 - fill color 170
 - filling area between plot lines (example) 236
 - for symbol plots 191
 - GBARLINE procedure 748, 764
 - GCHART procedure 784
 - hardware patterns 176, 181
 - images as fill elements 171
 - images on bar chart bars 116
 - mapping 971
 - outline colors 180
 - pattern sequences 182
 - pie and donut chart slices 831
 - pie and donut charts 875
 - pie and star charts 174
 - plots 1120
 - spacing between fill lines 292
 - star charts 840
 - PATTERNSTRIP applet parameter 410, 414
 - PATTERNSTRIP= parameters, JAVA and ActiveX 434
 - PCLIP= graphics option 331
 - PCOL device parameter 36
 - PCOLS device parameter 1151
 - PCOLS= option, GDEVICE procedure 332
 - PCT option
 - BAR statement 761
 - HBAR and VBAR statements 811
 - PEMPTY variable, Annotate facility 662
 - pen speed, plotters 352
 - PENMOUNTS= graphics option 105, 333
 - pens, active 333
 - PENSORT= graphics option 333
 - PENSORT= option, GDEVICE procedure 333
 - PERCENT option
 - BAR statement 761
 - HBAR and VBAR statements 811
 - PIE and DONUT statements 827
 - STAR statement 839, 840
 - percentage statistic 746, 782
 - percentiles, box plots 185, 187, 201, 233
 - PERCENTLABEL= option, HBAR and VBAR statements 812
 - permanent data sets 29
 - PIC= option, GTESTIT procedure 1290
 - PICTURE= option, GTESTIT procedure 1290
 - PIE and PIE3D statements, GCHART procedure
 - ActiveX and Java support for 1521
 - detail pie chart, creating (example) 883
 - grouping and arranging pie charts (example) 877
 - legends for patterns and midpoints (example) 875
 - pie chart with sum statistic (example) 842
 - subgrouping in pie or donut chart (example) 848
 - syntax and options 818
 - pie charts 6
 - angling text in (example) 1381
 - basics 776
 - detail pie chart, creating (example) 883
 - grouping and arranging (example) 877
 - legends for patterns and midpoints (example) 875
 - patterns 174

- slice colors and patterns 831
- slice labels and formatting 830, 873
- statistic and group headings 827, 832
- subgrouping in (example) 848
- sum statistic, specifying (example) 842
- terms used with 778
- pie-fill capability, device 334
- PIE function, Annotate facility
 - ActiveX and Java support for 1542
 - pie slices, drawing with Annotation facility 628
- PIE statement, BY statement with 144
- PIECNTR function, Annotate facility 1542
- PIEFILL GDEVICE procedure 334
- PIEFILL GOPTIONS statement 334
- PIEXY function, Annotate facility 1543
- %PIEXY macro, Annotate facility 688
- PLAY function (DSGI) 1460
- plot data sets 1086
- plot lines 196, 207
 - filling area between plot lines (example) 236
 - type of 196
- PLOT statement
 - contour levels, specifying (example) 908
 - labels for contour lines (example) 906
 - patterns and joins in contour plots (example) 910
 - simple contour plot, generating (example) 904
- PLOT statement, G3D procedure
 - ActiveX and Java support for 1537
 - generating default surface plot (example) 1314
 - rotating surface plot (example) 1316
 - tilting surface plots (example) 1317
- PLOT statement, GBARLINE procedure 765
 - ActiveX and Java support for 1519
 - syntax and options 765
- PLOT statement, GCONTOUR procedure
 - ActiveX and Java support for 1526
 - syntax and options 889
- PLOT statement, Gplot procedure 1101
 - ActiveX and Java support for 1530
 - connecting plot data points (example) 1129
 - different scales of values (example) 1138
 - filling areas in overlay plot (example) 1134
 - generating overlay plot (example) 1131
 - matching PLOT2 statements 1117
 - plots with drill-down for Web (example) 1141
 - plotting three variables (example) 1135
 - plotting two variables (example) 1126
- plot symbols 1114, 1120
 - altering or canceling 203
 - bar line charts 768
 - built-in drawing capability 354
 - colors for 93, 185, 206, 275
 - colors for, rotating through (example) 231
 - default 209
 - displaying with DSGI 1454
 - fonts of 186
 - in Annotate graphics output 640
 - interpolation 321
 - scatter plots 1308, 1320
 - size of 187, 201
 - specifying for plot points 205
- plot variables 741, 745
 - chart statistics 745
 - out of range 747
- PLOT2 statement, Gplot procedure 1115
 - ActiveX and Java support for 1530
 - different scales of values (example) 1138
 - matching PLOT statements 1117
- plots
 - basics of 1085
 - box plots 185, 187, 201, 233
 - classification variables with 1083
 - connecting plot data points (example) 1129
 - different scales of values in (example) 1138
 - drill-down functionality (example) 1141
 - filling areas in overlay plot (example) 1134
 - generating overlay plot (example) 1131
 - generating simple bubble plots (example) 1121
 - high-low plots 8, 189
 - interpolation methods 1085
 - labeling and sizing plot bubbles (example) 1122
 - missing values 1087, 1112
 - needle plots 191
 - out-of-range variables 747
 - overlay plots 1111, 1131, 1134
 - patterns 1120
 - plotting three variables (example) 1135
 - plotting two variables (example) 1126
 - regression analysis 192
 - regression analysis plots 192
 - right vertical axis to bubble plot (example) 1124
 - standard deviations 194
 - step plots 196
 - symbols in 1114, 1120
 - three variables and legend 1118
 - two variables 1082
 - two vertical axes 1084, 1119, 1124
 - with multiple variables 1113
- plotters
 - active pens or colors 333
 - drawing elements in color order 333
 - paper size 328
 - pen speed 352
- PNG device driver 382
 - ACTXIMG, JAVAIMG vs. 440
 - data tips for 568
 - developing web presentations 443
 - drill-down links in images 571
 - HTML files, generating 445
 - names for image output files 445
- PNG presentations 377
 - developing Web presentations 442
- POINT function, Annotate facility 1543
- POINTLABEL= option, SYMBOL statement 197, 1517
- points, drawing with Annotate facility 633
- points, plot
 - labels for 197
 - specifying for plot points 199
 - symbols for, specifying 199, 205
- POLELAT= option, GPROJECT procedure 1170, 1177
- POLELONG= option, GPROJECT procedure 1170, 1177
- %POLY, %POLY2 macro, Annotate facility 689
- POLY function, Annotate facility 1543
- POLYCONT function, Annotate facility 1544
- %POLYCONT macro, Annotate facility 689
- polygons
 - clipped (intersecting) 331, 335
 - drawing with Annotate facility 634
 - drawing with DSGI 1451
 - map data sets, creating 1041
 - testing ability to draw 1287
 - vertices, maximum drawn 324
- %POP macro, Annotate facility 690
- portability 40
- portrait orientation 34, 332, 344, 349
- ports, how output is written to 306
- POSITION= option
 - AXIS statement options 138
 - LEGEND statement options 155, 159, 162, 163
 - POINTLABEL= specification 198
- POSITION variable, Annotate facility 656
- positioning
 - Annotate graphics 596, 597
 - axis labels 127, 133, 135, 138
 - BY lines 143
 - donut chart labels 830
 - graphics element 39
 - graphics output, imported 976
 - legend label 153
 - legend text 159
 - legends 155, 162
 - pie and donut chart slice labels 830, 873
 - plot point labels 198
 - prism map light sources 1028
 - star chart slice labels 840
 - text in graphics output 221
 - titles and footnotes, ODS output 492
- POSTGEPILOG= graphics option 336
- POSTGRAPH= graphics option 337
- POSTGRAPH= option, GDEVICE procedure 337
- pound sign #, variables as plot point labels 198
- PPD file, location of 338
- PPDFILE= graphics option 338
- predefined color names 97, 99
- predefined graph styles (ODS) 489
- PREGEPILOG= graphics option 338
- PREGPROLOG= graphics option 339
- PREGRAPH= graphics option 339
- PREGRAPH= option, GDEVICE procedure 339
- PRESENTATION option, GREPLAY procedure 1245
- PRESENTATION window (GREPLAY) 1265
- PREVIEW statement, GREPLAY procedure 1256
- previewing device output 356
- previewing output 52
- printing 51
 - automatic 263
 - collating output 271
 - copies to print 298
 - duplex 287
 - duplex, binding edge for 265
 - flow control 311
 - graph orientation 349

- graphics files 51
 - IBM printers 297, 300, 302, 311
 - output bin, specifying 326
 - paper feed 264, 327
 - paper size 328
 - paper tray, specifying 329
 - paper type, specifying 330
 - PPD file, location of 338
 - previewing output 52, 356
 - protocol module, specifying 303
 - redrawing (overdrawing) graphs 348
 - reverse printing 349
 - prism maps 11, 998, 1023
 - identification variables 1005
 - predefined formats for 1035
 - producing simple prism map (example) 1063
 - response levels 1004
 - specifying midpoints in prism map (example) 1065
 - PRISM statement, GMAP procedure
 - ActiveX and Java support for 1527
 - producing simple prism map (example) 1063
 - specifying midpoints in prism map (example) 1065
 - syntax and options 1023
 - PROC GREPLAY window 1265
 - PROC statement 26
 - procedure output area 39
 - Annotate facility 598
 - procedure termination, step code at 305
 - procedures 26
 - product codes for 21
 - PROCESSINPUT= option, GDEVICE procedure 340
 - PROCESSOUTPUT= option, GDEVICE procedure 340
 - product codes for procedures 21
 - program mode, GDEVICE procedure 918
 - switching to 925, 1252
 - programs, SAS/Graph 25
 - running 31
 - PROJECT= option, GPROJECT procedure 1171, 1177
 - projecting coordinates from spherical to Cartesian 1161
 - basic usage of GPROJECT procedure 1172
 - clipping map areas (example) 1178
 - default projection specifications, using (example) 1174
 - emphasizing map areas (example) 1177
 - ID statement, GPROJECT procedure 1172
 - input map data sets 1163
 - projecting an Annotate data set (example) 1180
 - syntax and options, GPROJECT procedure 1168
 - types of map projections 1165
 - projecting map data sets 1040
 - PROJECTION= parameter, JAVA 434
 - PROJECTIONRATIO= parameter, JAVA 434
 - PROMPT= graphics option 341
 - prompt messages to GSF 64, 308, 459
 - PROMPTCHARS= graphics option 343
 - PROMPTCHARS= option, GDEVICE procedure 343
 - prompting to install ActiveX control 389
 - prompts
 - characters for, specifying 343
 - specifying if used 341
 - proportional fonts 940
 - protocol module, specifying 303
 - PROWS device parameter 36, 1151
 - PROWS= option, GDEVICE procedure 344
 - PTYPE variable (font data sets) 954
 - %PUSH macro, Annotate facility 690
 - PUT statement 409
 - animated GIF (example) 460, 463
 - PW= macro argument, DS2CSF macro 562
- Q**
- QMSG= option, GDEVICE procedure 345
 - QUIT statement 28, 33
 - GDEVICE procedure 927
 - GREPLAY procedure 1257
 - RUN-group processing 490
- R**
- R= option
 - GFONT procedure 950
 - PATTERN statement 171, 183
 - POINTLABEL= specification 198
 - SYMBOL statement 210
 - TITLE, FOOTNOTE, and NOTE statements 222
 - radar charts (star charts) 6, 1183
 - ActiveX and Java support for 1535
 - assigning axis definitions to axis spokes (example) 1210
 - basics 777
 - changing star type (example) 1207
 - CHART statement, GRADAR procedure 1185
 - color and line styles in (example) 1208
 - discrete numeric variables, charting (example) 881
 - filling stars (example) 1204
 - generating data set for (example) 1196
 - images in (example) 1205
 - labels 840
 - mode for, specifying (example) 1209
 - multiple classification variables in (example) 1202
 - overlying (example) 1199
 - patterns 174
 - patterns and outlines 840
 - producing basic (example) 1198
 - statistic and group headings 839, 842
 - sum statistic, specifying (example) 879
 - syntax and options, GRADAR procedure 1184
 - tiling (example) 1201
 - RANGE= macro argument, DS2CSF macro 563
 - RANGE option
 - BAR statement 761
 - HBAR and VBAR statements 812
 - Rangeview applet
 - drill-down functionality 573
 - DS2CSF macro 527, 528
 - hotspots 532
 - sample diagrams 530
 - RAXIS= option
 - AXIS statement 747
 - BAR statement 761
 - HBAR and VBAR statements 812
 - HBAR statement 144
 - PLOT statement 767
 - VBAR statement 144
 - RBSIZING= macro argument 549
 - RC= option, GFONT procedure 945
 - reading direction of text, changing (example) 1384
 - reading image file types 107
 - record length, GSF, origins 305
 - %RECT macro, Annotate facility 691
 - rectangle-fill capability, device 292, 345
 - rectangles, drawing with Annotate facility 616
 - RECTFILL= option, GDEVICE procedure 345
 - redrawing graphs 348
 - reducing map data sets 1039, 1213, 1220
 - reducing map of Canada (example) 1220
 - REF= option
 - BAR statement 761
 - HBAR and VBAR statements 812
 - REFCOL= option, GFONT procedure 945
 - reference-line labels, axis 133, 136
 - reference lines
 - colors 892, 893, 1093, 1094
 - REFLABEL= option, AXIS statement options 133, 136, 1509
 - REFLINES option, GFONT procedure 945
 - REGEQN option, PLOT statement 1112
 - regression analysis plots 192
 - regression plots 7
 - RENAME function (DSGI) 1461
 - RENAME statement, GDEVICE procedure 928
 - RENAME statement, MAPIMPORT procedure 1349
 - RENDER= graphics option 346
 - rendering fonts 82, 289
 - Bitstream fonts 295, 346
 - resolution, setting 293
 - software fonts 353
 - storing font files 347
 - RENDERLIB= graphics option 347
 - RENDERMODE= parameter, JAVA 435
 - RENDEROPTIMIZE= parameter, JAVA 435
 - RENDERQUALITY= parameter, JAVA 435
 - REPAINT= graphics option 348
 - REPAINT= option, GDEVICE procedure 348
 - REPEAT= option, PATTERN statement 171, 183
 - REPEAT= option, SYMBOL statement 210
 - repeating animation loops 321
 - replacing external files 70
 - REPLAY statement, GREPLAY procedure 1257
 - replaying graphs, device color limitations 105
 - reserved names, macro variables 566
 - RESET= graphics option 348
 - resetting graphics options 150, 348
 - RESOL= option, GFONT procedure 950
 - resolution
 - display device 363, 364, 365
 - fonts 950
 - image interlacing 320
 - software fonts 293
 - RESOURCES= graphics option 476

- RESOURCESFONTNAME= graphics option 477
- response axis, bar charts 815
- response axis, bar line charts 763
- response data sets 1003
- assigning formats to response variables, block map (example) 1049
 - identification variables 1005
 - merging feature tables with 1001
- response levels
- maps 1004
 - specifying in block maps (example) 1047
 - specifying midpoints in prism map (example) 1065
- RESPSTAT= option, HBAR and VBAR statements 729
- RETAIN statement 600
- return characters at record ends 300
- return codes 305, 1501
- REVEAL suboption, AUTOLABEL= option 898
- REVERSE= graphics option 349
- reversing black and white 353
- RF= option, GFONT procedure 945
- RGB color scheme 95
- converting to HLS 103
- %RGB macro 102
- %RGB2HLS macro 103
- RH= option, GFONT procedure 946
- RIGHTMARGIN= option, GDEVICE procedure 316
- RIGHTMARGIN= option, GOPTIONS statement 316
- roles 411
- Roman alphabet text fonts 84
- ROMCOL= option, GFONT procedure 945
- ROMFONT= option, GFONT procedure 945
- ROMHEX= option, GFONT procedure 946, 951
- ROMHT= option, GFONT procedure 946
- ROTATE= graphics option 349
- ROTATE= option
- GDEVICE procedure 349
 - LABEL= option, DONUT statement 830
 - PLOT statement, G3D procedure 1303
 - SCATTER statement, G3D procedure 1308, 1312
 - SURFACE statement 1032, 1068
 - TDEF statement, GREPLAY procedure 1260
 - TITLE, FOOTNOTE, and NOTE statements 222
- ROTATE variable, Annotate facility 659
- rotating graphs for printing 349
- rotating plot symbols through colors (example) 231
- ROTATION= option, GDEVICE procedure 350
- rows
- in graphics output area 36, 323, 350, 361
 - legends 153
- ROWS= option, GDEVICE procedure 350
- RSTAT= option, HBAR and VBAR statements 729, 735
- RUN-group processing 33
- GSLIDE procedure 1281
 - ODS and 490
- RUN statement 26, 144
- run-time controls in presentations 471
- RUNMODE= macro argument 545, 565
- running programs 31
- ## S
- S option, GOPTIONS procedure 1078
- sample programs 21
- SAS Color Naming Scheme (CNS) 97, 99
- SAS/GRAPH software installation, testing 1285, 1291
- SAS Maps Online 1006
- SAS output 48, 1150
- SASPOWER= macro argument 554
- SCALABLE= option, GDEVICE procedure 351
- %SCALE macro, Annotate facility 692
- SCALE statement, GIMPORT procedure 975, 979
- %SCALET macro, Annotate facility 693
- SCALEX= and SCALEY= options, TDEF statement 1261
- scaling
- dash length in lines 278
 - graphs with DSGI windows 1388
 - hardware fonts 274, 350, 351
- scatter plots 7, 9, 1296
- appearance of points 1310
 - axes, controlling 1299
 - axes, reversing values on 1312
 - connecting plot data points (example) 1129
 - data ranges 1298
 - generating simple scatter plot (example) 1318
 - input data sets 1298
 - plotting three variables (example) 1135
 - plotting two variables (example) 1126
 - rotating and tilting 1299
 - rotating (example) 1323
 - shapes in (example) 1320
 - simulating overlaid scatter plots 1310
 - three-dimensional, syntax for 1305
- SCATTER statement, G3D procedure 1305
- ActiveX and Java support for 1537
 - generating simple scatter plot (example) 1318
 - rotating scatter plot (example) 1323
 - shapes in (example) 1320
- SCLNKWT= macro argument 550
- SCLWIDTH= macro argument 550
- SCNSIZE= macro argument 550
- Script drill-down mode, Java 400, 413, 572
- example 416
- searching device catalogs 917
- SEGMENT variable (font data sets) 956
- SEGMENT variable (map data sets) 1000
- SELECT statement, MAPIMPORT procedure 1349
- SELIFUNC= macro argument 550
- SELLFUNC= macro argument 550
- SELUFUNC= macro argument 551
- SEPCLASS= macro argument 554
- SEPLoc= macro argument 554
- SEPTYPE= macro argument 554
- %SEQUENCE macro, Annotate facility 695
- server graphs vs. client graphs 584
- shadow color, legends 153
- shadowing, legend frames 326
- shape, legend values 156
- SHAPE= option
- BLOCK statement, GMAP procedure 1015
 - HBAR and VBAR statements 812
 - LEGEND statement options 156
 - SCATTER statement, G3D procedure 1308, 1310, 1320
- shapes in scatter plots 1308, 1320
- SHORT option, GOPTIONS procedure 1078, 1079
- SHOWALL option, GFONT procedure 946
- SHOWBACKDROP= parameter, JAVA 435
- SHOWLEGEND= parameter, JAVA 436
- SHOWLINKS= macro argument 551
- SHOWROMAN option, GFONT procedure 946, 950, 962
- SIDE option, PLOT statement 1303
- SIMFONT= graphics option 351
- simple plot lines 7
- SIMPLEDEPTHSORT= parameter, JAVA 436
- SIMPLETHRESHOLD= parameter, JAVA 436
- SIMULATE font 77
- SINGULAR= option, POINTLABEL= specification 199
- singularities, checking for 199
- size
- aspect ratio 263, 1152
 - axis labels 127, 133, 135, 136
 - axis tick marks 129, 139, 140
 - axis values 136
 - boxes in box plots 185
 - bubbles in bubble plots 1092
 - BY lines 143, 312
 - character cells 37, 38, 265
 - contour labels 187
 - contour lines 201
 - contour plot labels 903
 - dash length in lines 278
 - display, in lines 308
 - donut chart labels 829
 - enlarging graph areas with DSGI windows (example) 1391
 - errors in sizing 40
 - fonts 269
 - GRAPH window 50
 - graphics output 1150
 - graphics output, imported 975
 - graphics output text 316
 - legend frame 153
 - legend label 153
 - legend values 156, 157
 - line thickness, default 323
 - paper 328
 - paper feed increments 327
 - plot bubbles (example) 1122
 - plot print labels 197
 - plot symbols 187, 201
 - record length, to GSF 305
 - scatter plot points 1309, 1310
 - splines in star charts 1196
 - text in graphics output 218
 - titles and footnotes 317, 493
 - units of measurement 38, 310
- size, graphics output area 316, 361, 363, 364,
- columns in 36, 274, 315, 322, 3
 - rows in 36, 323, 350, 361
- SIZE= option, SCATTER statement 1309, 1310
- SIZE variable, Annotate facility 660
- SKIPMISS option, PLOT statement 1112

- `%SLICE` macro, Annotate facility 695
- `SLICE=` option
 - PIE and DONUT statements 828
 - STAR statement 839, 840
- `SLIDECTL=` option, META2HTM macro 473, 565
- `SLIDESHOWCONTROLENABLED=` graphics option 477
- `SLIDESHOWENABLED` parameter, ODS statements 473
- smooth line fit 193
- `SMOOTH=` option, GRID statement 1329, 1331, 1335, 13
- smoothing plot lines 190
- software fonts 76, 351
 - listing available 76
 - open at one time 290
 - rendering 353
 - resolution 293
 - where stored 82
- sorting
 - grouped observations 142
 - map data set observations 1227
 - plot data set observations 1087
- space data sets 960
 - variables for, list of 960
- `SPACE=` option
 - BAR statement 762
 - GFONT procedure 951
- `SPACEDATA=` option, GFONT procedure 951, 961
- spacing
 - angle of rotation 222
 - between Bitstream font letters 295
 - between display area and graphic 360
 - between displayed area and graph 314
 - between fill lines 292
 - contour plot labels 904
 - fonts 948, 951, 960, 961
 - legends 154, 163
 - text in graphics output 216, 221
- `SPLCLASS=` macro argument 554
- special characters 81
 - HTML entities 582
- Special font 89
- special plot symbols 200
- `SPEED=` graphics option 352
- speed of plotter pens 352
- `SPEED=` option, GDEVICE procedure 352
- spherical coordinates, converting to Cartesian 1161
 - basic usage of GPROJECT procedure 1172
 - clipping map areas (example) 1178
 - default projection specifications, using (example) 1174
 - emphasizing map areas (example) 1177
 - ID statement, GPROJECT procedure 1172
 - input map data sets 1163
 - projecting an Annotate data set (example) 1180
 - syntax and options, GPROJECT procedure 1168
 - types of map projections 1165
- `SPIDER` option, CHART statement 1193
- `SPIDERWEB` option, CHART statement 1193
- `SPKLABEL=` option, CHART statement 1193
- spline interpolation 194, 205, 1330
 - GRID statement (example) 1339, 1342, 1343
- `SPLINE` option, GRID statement 1330, 1335, 1339
- spline smoothing 1331, 1339
- splines in star charts
 - axis definitions 1210
 - colors 1188, 1208
 - labels for 1193
 - line types 1191, 1208
 - size 1196
- `SPLIT=` option, AXIS statement options 134
- `SPREAD=` macro argument 551
- SR option, GFONT procedure 946, 950
- `SSFILE1=`, ..., `SSFILE5=` macro arguments 555
- `SSFREF1=`, ..., `SSFREF5=` macro arguments 555
- `SSHREF1=`, ..., `SSHREF5=` macro arguments 555
- `SSMEDIA1=`, ..., `SSMEDIA5=` macro arguments 555
- `SSREL1=`, ..., `SSREL5=` macro arguments 555
- `SSREV1=`, ..., `SSREV5=` macro arguments 556
- `SSTITLE1=`, ..., `SSTITLE5=` macro arguments 556
- `SSTYPE1=`, ..., `SSTYPE5=` macro arguments 556
- `STACKED=` parameter, JAVA 436
- `STACKPERCENT=` parameter, JAVA 436
- standard deviations 194
- STAR statement, GCHART procedure
 - BY statement with 144
 - discrete numeric variables, charting (example) 881
 - star chart with sum statistic (example) 879
 - syntax and options 833
- `STARAXES=` option, CHART statement 1193
- `STARAXIS=` option, CHART statement 1193, 1210
- `STARCIRCLE=` option, CHART statement 1194
- `STARCIRCLES=` option, CHART statement 1194
- `STARFILL=` option, CHART statement 1194, 1204
- `STARINRADIUS=` option, CHART statement 1194
- `STARLEGEND=` option, CHART statement 1194, 1201
- `STARLEGENDLAB=` option, CHART statement 1194, 1201
- `STARMAX=` option, STAR statement 839
- `STARMIN=` option, STAR statement 839
- `STAROUTRADIUS=` option, CHART statement 1194
- stars, drawing circle of (example) 609
- `STARSTART=` option, CHART statement 1195
- `STARTYPE=` option, CHART statement 1195, 1207
- STATE function (DSGI) 1430
- state map data (U.S.) 1003
 - removing U.S. state boundaries (example) 1228
- statement options 46
- statements, SAS/GRAPH 123
 - global 14, 27, 33, 123
- static graphics 439
 - creating with ODS 440
 - developing presentations with GIF, JPEG, PNG 443
 - presentations developed with ACTXIMG, JAVAIMG 442
 - sample programs for 447
- static images in presentations 370, 371, 376
- statistic heading
 - pie and donut charts 827, 832
 - star charts 839, 842
- step codes 305
- `STEP=` option, SYMBOL statement 199, 904
- step plots 196
- stock market high, low, close data 189
- storing
 - Annotate graphics (example) 713
 - clipped polygons 331
 - DSGI graphs 1357
 - fonts 77
 - graphics catalogs 53, 55, 59, 70
 - graphics files 51, 60
 - graphics output as files 64, 66, 68, 291
 - Java archive 399
 - Java plug-in 424
 - online help 385
 - PPD file 338
 - rendered font files 346, 347
- strings
 - appending to graphics data records 300
 - prefixing output records 309
 - sending to devices or files 301, 303
- stroked fonts 940
- `STYLE=` option, AXIS statement options 134
- `STYLE=` option, ODS statements 489
- style variable, annotate facility 660
- styles, ODS 488
- stylesheets, macro arguments for 554
- `SUBGROUP=` option
 - BLOCK statement 793, 795
 - HBAR and VBAR statements 729, 733, 735, 813
 - PIE and DONUT statements 828
- `SUBGROUP` variable, Annotate facility 664
- subgrouping
 - block chart (example) 844
 - pie charts (example) 877
 - pie or donut chart (example) 848
 - vertical bar chart (example) 848
- subsetting map data sets 1039, 1173, 1178, 12
 - example of 1178
 - reducing map of Canada (example) 1220
- substitution strings
 - drill-down tags as 411
 - removing blanks from data values 414
 - variables as 410
- success factor diagrams
 - drill-down functionality 573
 - DS2CSF macro 527, 528
 - hotspots 532
 - sample diagrams 530
- SUM option
 - BAR statement 762
 - HBAR and VBAR statements 813
- sum statistic 746, 783
 - bar chart (example) 846
 - block chart (example) 842
 - pie chart (example) 842
 - star chart (example) 879

- SUMLABEL= option, HBAR and VBAR statements 813
- SUMVAR= option
- bar chart with sum statistic (example) 846
 - BAR statement 762
 - block chart with sum statistic (example) 842
 - BLOCK statement 793
 - CHART statement, GRADAR procedure 1195
 - HBAR and VBAR statements 814
 - PIE and DONUT statements 828
 - pie chart with sum statistic (example) 842
 - PLOT statement 745, 767
 - star chart with sum statistic (example) 879
 - STAR statement 839
- suppressing axes
- BAR statement 760
 - BUBBLE statement 1097
 - HBAR and VBAR statements 810
 - PLOT statement, G3D procedure 1303
 - PLOT statement, GCONTOUR procedure 897
 - PLOT statement, GPLOT procedure 1111
 - SCATTER statement, G3D procedure 1307
- surface maps 12, 998, 1030
- axes, controlling 1299
 - identification variables 1005
 - predefined formats for 1035
 - producing simple surface map (example) 1066
 - response levels 1004
 - rotating and tilting 1299
 - rotating and tilting surface map (example) 1068
- surface plots 9, 1295
- appearance of surface 1304
 - data ranges 1298
 - input data sets 1298
 - rotating (example) 1316
 - three-dimensional, syntax for 1301
 - tilting (example) 1317
- surface plots, three-dimensional, processing data for 887, 1327
- controlling observations in output data set 1336
 - GRID statement, G3GRID procedure 1333
 - interpolation methods 1329
 - simple contour plot, generating (example) 905
 - spline interpolation, partial (example) 1342
 - spline interpolation, with smoothed spline (example) 1339
 - spline interpolation (example) 1343
 - syntax and options, G3GRID procedure 1332
 - using default interpolation method (example) 1337
- SURFACE statement, GMAP procedure 1030
- producing simple surface map (example) 1066
 - rotating and tilting surface map (example) 1068
- SURFACESIDECOLOR= parameter, JAVA 436
- SWAP function, Annotate facility 639
- SWAP= graphics option 353
- %SWAP macro, Annotate facility 696
- SWAP= option, GDEVICE procedure 353
- SWFONTRENDER= graphics option 353
- SYMBOL definitions, BY statement with 145
- symbol fonts, creating figures for 964
 - SYMBOL function, Annotate facility 1544
 - SYMBOL= graphics option 354, 1078
 - SYMBOL= option, GDEVICE procedure 354
- SYMBOL statement 27, 124, 183
- ActiveX and Java support for 1516
 - altering or canceling 203
 - box plots, modifying (example) 233
 - GBARLINE procedure 740, 768
 - GCONTOUR procedure 903
 - GPLOT procedure 1114, 1120
 - internationalization 392
 - Java applets, internationalization and 400
 - rotating plot symbols through colors (example) 231
- SYMBOLS= option, GDEVICE procedure 354
- syntax conventions 17
- system fonts 76
- %SYSTEM macro, Annotate facility 696
- ## T
- T= option, AXIS statement options 138
- T option, GOPTIONS procedure 1078
- T= option, LEGEND statement options 160
- tables of contents 495, 497
- tables of pages 496, 497
- TARGETDEVICE= graphics option 52, 72, 105, 354
- TC option
- ? statement, GREPLAY procedure 1246
 - LIST statement, GREPLAY procedure 1254
- TC= option, GREPLAY procedure 1245
- TC statement, GREPLAY procedure 1258
- TCOPY statement, GREPLAY procedure 1258
- TDEF statement, GREPLAY procedure 1259
- TDELETE statement, GREPLAY procedure 1262
- TEMPLATE DESIGN window (GREPLAY) 1265
- TEMPLATE option
- ? statement, GREPLAY procedure 1246
 - LIST statement, GREPLAY procedure 1254
- TEMPLATE= option, GREPLAY procedure 1245
- TEMPLATE statement, GREPLAY procedure 1262
- templated graphs 13
- templates 1239
- copying or duplicating 1258
 - creating 1268, 1270
 - defining or modifying in catalogs 1259
 - deleting 1262
 - panel outlines for, displaying 1256
 - printing contents of 1254
 - relaying graphics output in templates 1272
 - replaying graphics output in 1270
 - specifying/assigning 1258, 1262
 - transporting 58
- temporary data sets 29
- terminating drivers 286
- test pattern, GTESTIT procedure 1286
- testing installation of SAS/GRAPH software 1285
- managing colors list for device driver (example) 1291
 - testing GOPTIONS statement (example) 1291
- TEXALIGN function (DSGI) 1430, 1486
- TEXCOLOR function (DSGI) 1431, 1488
- TEXEXTENT function (DSGI) 1433
- TEXTFONT function (DSGI) 1434, 1489
- TEXHEIGHT function (DSGI) 1435, 1490
- TEXINDEX function (DSGI) 1436, 1491
- TEXPATH function (DSGI) 1437, 1491
- TEXREP function (DSGI) 1437, 1492
- text 1456
- adjusting character size in output (example) 1156
 - angle of 330
 - as axis values 131, 134
 - as legend values 155, 157
 - axis text, formatting 135
 - block charts 795
 - BY lines 143
 - contour plot labels, specifying 904
 - donut chart labels 829
 - HTML entities 582
 - in Annotate graphics output 626
 - reading direction, changing (example) 1384
 - specifying color text in output (example) 1153
- text color 276
- text files, converting to graphics output 1147
- adjusting character size in output (example) 1156
 - adjusting output size 1150
 - external text files, about 1148
 - fonts 1152
 - specifying color text in output (example) 1153
 - syntax and options, GPRINT procedure 1149
- text slides for presentations 12, 13, 1277
- Annotate graphics, displaying 1278, 1283
 - multiple graphs on same slide 13
 - producing (example) 1282
 - slide-show control 473
- TEXT variable, Annotate facility 666
- TEXUP function (DSGI) 1438, 1493
- three-dimensional plots 9
- tick marks, axes 129, 138
- formatting 139
 - offset 130
 - scatter plots 1310
 - suboptions, list of 1509
 - surface plots 1304
 - with datetime values (example) 226
- TICK= option, AXIS statement options 138
- TICK= option, LEGEND statement options 160
- TILELEGEND= option, CHART statement 1195
- TILELEGLABEL= option, CHART statement 1195
- tiling radar charts (example) 1201
- TILT= option
- PLOT statement, G3D procedure 1303
 - SCATTER statement, G3D procedure 1309
 - SURFACE statement 1032, 1068
- tilting
- surface and scatter plots 1299, 1317
 - surface maps (example) 1068
- TIPBACKCOLOR= parameter, JAVA 436
- TIPBORDERCOLOR= parameter, JAVA 436
- TIPMODE= parameters, JAVA and ActiveX 437
- TIPS= macro argument 551
- TIPS= parameters, JAVA and ActiveX 437
- TIPSTEMSIZE= parameters, JAVA and ActiveX 437

- TIPTEXTCOLOR= parameters, JAVA and ActiveX 437
- TIPTYPE= argument, META2HTM macro 565
- TIPTYPE= macro argument 551
- TITLE graphics option 1078, 1079
- TITLE statement 27, 124, 210, 224
- ActiveX and Java support for 1517
 - BY statement with 145
 - displaying with GOPTIONS procedure (example) 1079
 - enhancing titles (example) 238
- titles 211, 224
- angle of rotation 213, 219
 - boxes around 215, 216
 - colors for 215, 216, 276
 - default characteristics, setting 225
 - defining text of 222, 226
 - enhancing (example) 238
 - fonts 295
 - fonts, color, and size (ODS output) 493
 - fonts for 217
 - hyperlinks for 220
 - justification 218
 - ODS output 492
 - positioning 39, 221
 - size of 218, 316, 317
 - spacing around 221
 - text breaks 225
 - underlining 223
- titles macro, arguments for 556
- TO variable (GKEYMAP data set) 987
- tokens, GDDM 299
- TOLANGLE= suboption, AUTOLABEL= option 899
- TOLEN variable (GKEYMAP data set) 988
- traditional map data sets 999
- creating 1041
 - identification variables 1005
 - lakes, displaying 1041
 - projecting 1040
 - response data sets with 1003
 - subsetting or reducing (clipping) 1039, 1173, 1178, 12
- traditional map data sets, projecting coordinates from spherical to Cartesian 1161
- basic usage of GPROJECT procedure 1172
 - clipping map areas (example) 1178
 - default projection specifications, using (example) 1174
 - emphasizing map areas (example) 1177
 - ID statement, GPROJECT procedure 1172
 - input map data sets 1163
 - projecting an Annotate data set (example) 1180
 - syntax and options, GPROJECT procedure 1168
 - types of map projections 1165
- TRAILER= option, GDEVICE procedure 356
- TRAILER records 356, 357
- TRAILERFILE= option, GDEVICE procedure 357
- trailers, animation 458
- TRANLIST= macro argument 561
- TRANS function (DSGI) 1439
- transformations, DSGI 1378
- TRANSLATE statement, GIMPORT procedure 976
- adjusting graphics output (example) 979
- translation table, ASCII-to-EBCDIC 358
- TRANSNO function (DSGI) 1439, 1496
- transparency, image 357
- TRANSPARENCY GOPTIONS statement 357
- TRANSPARENCY= graphics option 459
- transporting and converting graphics output 56
- TRANTAB= graphics option 358
- TRANTAB= option, GDEVICE procedure 358
- tray, paper 329
- TREEDIR= macro argument 551
- TREESPAN= macro argument 552
- Treeview applet 372, 503
- data tips with 569
 - drill-down functionality 572
 - DS2TREE macro with 505
 - enhancing presentations for 506
 - hotspots 510
 - when to use 504
 - XML embedded in HTML file (example) 507
 - XML written to external file (example) 509
- TREPLAY statement, GREPLAY procedure 1263
- relaying graphics output in templates 1270, 1272
- troubleshooting
- Annotate data sets 604
 - SAS/GRAPH software installation 1285
 - Web output 579
- trueness of color 105
- TTAG= macro argument 556
- two-dimensional bar charts 116
- two-sided printing 265, 287
- TXT2CNTL function, Annotate facility 642
- %TXT2CNTL macro, Annotate facility 697
- TYPE= option
- BAR statement 762
 - BLOCK statement 794
 - GDEVICE procedure 358
 - GKEYMAP procedure 989
 - HBAR and VBAR statements 814
 - PIE and DONUT statements 828
 - PLOT statement 767
 - STAR statement 839
- ## U
- U option, GFONT procedure 951
- U= option, TITLE, FOOTNOTE, and NOTE statements 223, 225
- UCC= graphics option 359
- UCC= option, GDEVICE procedure 359
- UCC values 359
- ULX= and ULY= options, TDEF statement 1261
- unclipped polygons, storing 331
- UNDERFLOWCOLOR= parameters, JAVA and ActiveX 437
- UNDERLIN= option, TITLE, FOOTNOTE, and NOTE statements 223, 225
- underlining in titles, footnotes, and notes 223
- Unicode references for character data 561
- uniform fonts 940, 951
- UNIFORM option
- GFONT procedure 951
 - GPlot procedure 144, 1089
- uninstalling ActiveX Control 390
- unit area (maps) 1005
- units of measurement 38, 310
- Annotate graphics 597
- unmatched area boundaries
- GREDUCE procedure and 1215
 - REMOVE procedure and 1225
- UPDATE function (DSGI) 1461
- URL drill-down mode, Java 401, 409, 572
- example 417
- URL= option, ODS HTML statement 167
- URX= and URY= options, TDEF statement 1261
- U.S. city map data 1003
- U.S. state map data 1003
- removing U.S. state boundaries (example) 1228
- user-defined control characters, device 359
- user input, enabling 360
- USERFMT= parameters, JAVA and ActiveX 437
- USERINPUT= graphics option 459
- ## V
- V= option, SYMBOL statement 199, 205
- V6COMP graphics option 362, 749, 785
- patterns 181
- VALUE= option
- AXIS statement 134, 1509
 - LEGEND statement 1515
 - PATTERN statement 171
 - PIE and DONUT statements 829
 - STAR statement 840
 - SYMBOL statement 199, 205, 904
- VALUEPOS= macro argument, DS2CSF macro 564
- values on axes 134
- order of 130
 - splitting (multiline) 134
- values on legends
- order of 155
 - size and shape of 156
- VAR= macro argument, DS2CSF macro 562
- variable roles 411
- variables
- Annotate facility 591, 599, 602, 642
 - as substitution strings 410
 - bar variables 741, 742, 745
 - chart variables 778, 779, 780
 - classification, plotting 1083
 - contour variables 885
 - declaring as plot point labels 198
 - font data sets 952
 - GKEYMAP data sets 987
 - identification variables 1005
 - identification variables, maps 1005
 - kern data sets 959
 - link and enhancement variables in presentations 574
 - macro variable names 566
 - multiple classification variables in radar charts (example) 1202
 - plot variables 741, 745, 747

plotting three variables (example) 1135
 plotting two variables (example) 1126
 space data sets 960
 variance 195
 VAXIS= option
 BUBBLE statement 1097
 PLOT statement, GCONTOUR procedure 886, 897
 PLOT statement, GPLOT procedure 1112
 VBAR and VBAR3D statements
 drill-down functionality in bar chart (example) 856
 GAREABAR procedure 728
 GCHART procedure 796, 1521
 subgrouping in vertical bar chart (example) 848
 vector graphics files, rendering software fonts 353
 Version 6, SAS/GRAPH
 defaults for programs 362
 patterns 181
 vertical axes, multiple in plots 1084, 1119, 1124
 vertical bar charts 5, 775
 BAR statement, GBARLINE procedure 751, 1519
 statistics in, displaying 815
 subgroup labels (example) 607
 subgrouping in (example) 848
 terms used with 778
 vertical resolution, device 35
 vertices, maximum drawn 324
 VIEW2D= parameters, JAVA and ActiveX 437
 VIEWPOINT=2D= parameter, JAVA 438
 VIEWPORT function (DSGI) 1441, 1497
 viewports, DSGI 1376, 1385
 VM= option
 BUBBLE statement 1097
 PLOT statement, GCONTOUR procedure 897
 PLOT statement, GPLOT procedure 1112
 VMINOR= option
 BUBBLE statement 1097
 PLOT statement, GCONTOUR procedure 897
 PLOT statement, GPLOT procedure 1112
 VORIGIN device parameter 35
 VORIGIN= graphics option 360
 VPOS function (DSGI) 1442, 1498
 VPOS= graphics option 36, 1151, 1152
 VREF= option
 BUBBLE statement 1097
 PLOT statement, GCONTOUR procedure 897
 PLOT statement, GPLOT procedure 1113
 VREVERSE option
 BUBBLE statement 1097
 PLOT statement, GCONTOUR procedure 898
 PLOT statement, GPLOT procedure 1113
 VSIZE device parameter 35
 VSIZE function (DSGI) 1443, 1499
 VSIZE= graphics option 35, 361, 1150
 VSIZE= option, GDEVICE procedure 361
 VSPACE= macro argument 537
 ZERO option
 BUBBLE statement 1097
 PLOT statement, GPLOT procedure 1113

W

W= option, AXIS statement options 140
 W= option, SYMBOL statement 201
 WAXIS= option, CHART statement 1195
 Weather font 89
 Web browsers
 installing ActiveX control 389
 Netscape colors, resolving 583
 Web output
 Annotate facility for 499
 Annotate variables for 601
 developing for Metaview Applet 469
 developing with ACTXIMG and JAVAIMG drivers 442
 enhancing with GIF, JPEG, PNG drivers 443
 generating presentations 382
 HTML files, generating with ODS 445
 multiple instances of Metaview Applet (example) 483
 naming conventions for image files 445
 ODS styles for 488
 page formatting, macro arguments for 552
 presentation features 379
 presentation types 370, 378
 producing with META2.HTM macro (example) 481
 run-time controls 471
 static graphics 439
 stylesheets, macro arguments for 554
 troubleshooting 579
 Web pages
 bar chart with drill-down (example) 255
 combining graphs and reports (example) 248
 creating with ODS HTML (example) 245
 WEIGHT= option, CHART statement 1196
 weighted statistics
 GBARLINE procedure 746, 770
 GCHART procedure 783
 WFRAME= option
 CHART statement, GRADAR procedure 1195
 GSLIDE procedure 1281
 WHEN variable, Annotate facility 602, 667
 WHERE= data set option, subsetting map data sets 1039
 WHERE statement 28, 1039
 RUN-group processing 33
 white and black, reversing 353
 WIDTH= macro argument 537
 WIDTH= option
 AXIS statement options 140
 BAR statement 762
 HBAR and VBAR statements 814
 SYMBOL statement 201
 WIDTHSTAT= option, HBAR and VBAR statements 729
 WINDIC= macro argument, DS2CSF macro 564
 WINDOW function (DSGI) 1444, 1500
 windowing mode, GDEVICE procedure 918, 928
 switching to program mode 925, 1252
 windowing mode, GREPLAY procedure 1241
 windows, DSGI 1376
 enlarging graph areas with DSGI windows (example) 1391
 scaling graphs with (example) 1388
 WORK data library 30

WOUTLINE= option
 BLOCK statement, GMAP procedure 1015
 CHORO statement 1022
 PIE and DONUT statements 829
 PRISM statement 1028
 STAR statement 840
 writing image files types 107, 110
 WSACTIVE function (DSGI) 1445
 WSOPEN function (DSGI) 1445
 WSPOKE= and WSPOKES= options, CHART statement 1196
 WSTAR= option, CHART statement 1196
 WSTARCIRCLE= and WSTARCIRCLES= options, CHART statement 1196
 WSTARS= option, CHART statement 1196
 WSTAT= option, HBAR and VBAR statements 729
 chart with subgrouping and variable percentages (example) 735

X

X= option
 SCALE statement 975
 TRANSLATE statement 976
 X variable, Annotate facility 668
 X variable, map data sets 1000
 XADJ variable (kern data sets) 959
 XBINS= parameter, JAVA 437
 XC variable, Annotate facility 669
 XLAST variable, Annotate facility 678
 XLATEX= and XLATEY= options, TDEF statement 1261
 XLIGHT= option, PRISM statement 1028
 XLSTT variable, Annotate facility 678
 XMAX device parameter 34, 35, 1150
 XMAX= graphics option 363
 XMAX= option, GDEVICE procedure 363
 XMLFILE= macro argument 545
 XMLFREF= macro argument 545
 XMLTYPE= macro argument 545
 XMLURL= macro argument 545
 XPIXELS device parameter 35
 XPIXELS= graphics option 35, 364, 459
 XPIXELS= option, GDEVICE procedure 364
 XSIZE= option
 BLOCK statement, GMAP procedure 1015
 CHORO statement 1022
 PRISM statement 1029
 SURFACE statement 1032
 XSYS variable, Annotate facility 670
 XTICKNUM= option
 PLOT statement, G3D procedure 1304
 PLOT statement, GCONTOUR procedure 898
 SCATTER statement, G3D procedure 1310
 XVIEW= option
 BLOCK statement, GMAP procedure 1016
 CHORO statement 1022
 PRISM statement 1029
 XYTYPE= option, PLOT statement 1304

Y

Y= option
 SCALE statement 975
 TRANSLATE statement 976
 Y variable, Annotate facility 673
 Y variable, map data sets 1000
 YBINS= parameter, JAVA 437
 YC variable, Annotate facility 674
 YLAST variable, Annotate facility 678
 YLIGHT= option, PRISM statement 1028
 YLSTT variable, Annotate facility 678
 YMAX device parameter 34, 35, 1150
 YMAX= graphics option 365
 YMAX= option, GDEVICE procedure 365
 YPIXELS device parameter 35
 YPIXELS= graphics option 35, 365, 459
 YPIXELS= option, GDEVICE procedure 365
 YSIZE= option
 BLOCK statement, GMAP procedure 1015
 CHORO statement 1022

PRISM statement 1029
 SURFACE statement 1032
 YSYS variable, Annotate facility 675
 YTICKNUM= option
 PLOT statement, G3D procedure 1304
 PLOT statement, GCONTOUR procedure 898
 SCATTER statement, G3D procedure 1310
 YVIEW= option
 BLOCK statement, GMAP procedure 1016
 CHORO statement 1022
 PRISM statement 1029

Z

z/OS operating environment, JAVAIMG driver
 with 443
 Z variable, Annotate facility 676
 zero values, block charts 795

ZMAX= option
 PLOT statement, G3D procedure 1304
 SCATTER statement, G3D procedure 1310
 ZMIN= option
 PLOT statement, G3D procedure 1304
 SCATTER statement, G3D procedure 1310
 zoom controls 473
 ZOOM= macro argument 552
 ZOOMCONTROLENABLED= graphics option 477
 ZOOMCONTROLMAX= graphics option 477
 ZOOMCONTROLMIN= graphics option 477
 ZOOMCTL= argument, META2HTM
 macro 566
 ZSYS variable, Annotate facility 677
 ZTICKNUM= option
 PLOT statement, G3D procedure 1304
 SCATTER statement, G3D procedure 1310
 ZVIEW= option
 BLOCK statement, GMAP procedure 1016
 CHORO statement 1022
 PRISM statement 1029

Your Turn

If you have comments or suggestions about *SAS/GRAPH® 9.1 Reference*, please send them to us on a photocopy of this page, or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
E-mail: yourturn@sas.com

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
E-mail: suggest@sas.com

