CHAPTER

31

The DATA Step Graphics Interface

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.

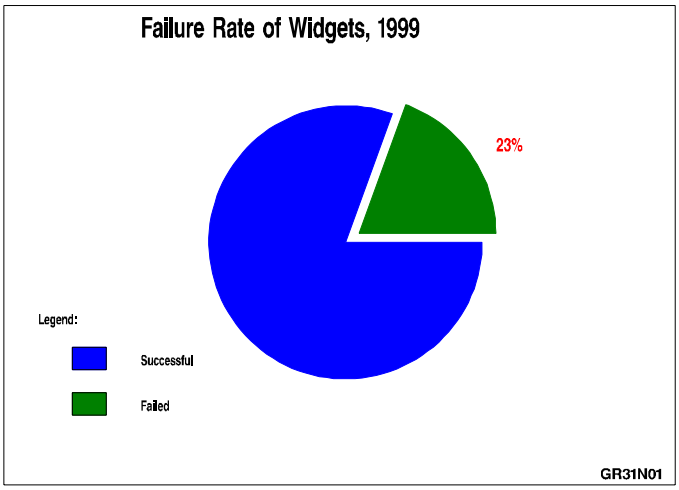
You should consider using the Annotate facility for enhancing procedure output and using DSGI for creating custom graphics without using a graphics procedure.

DSGI is based upon the Graphics Kernal 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.

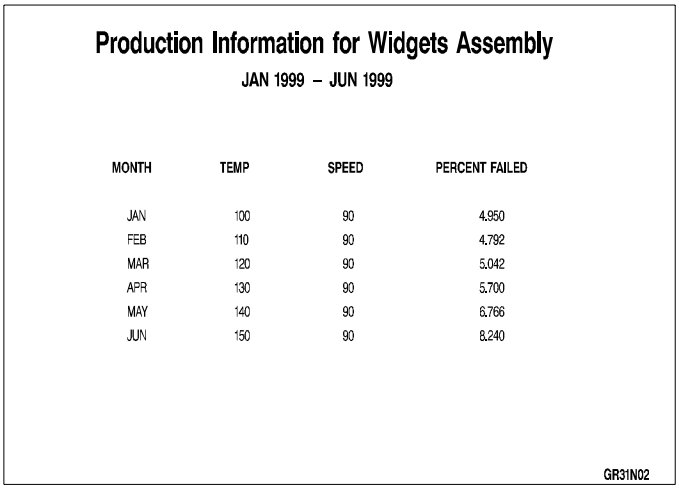
This chapter explains the concepts used to create graphics output with DSGI. The discussion provides an overview of the functions and routines used in DSGI. For complete details of each function and routine, see Chapter 32, “DATA Step Graphics Interface Dictionary,” on page 1075.

Display 31.1 on page 1029 shows a pie chart that was created entirely with DSGI functions. Display 31.2 on page 1029 is an example of a text slide that was created with DSGI statements.

Display 31.1 Exploded Pie Chart Generated with the DSGI



Display 31.2 Text Slide Created Using the DSGI



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.

return-code-variable is an arbitrary name and can be any numeric variable name. It will hold the return code upon execution of the function.

function-name is the DSGI command you want to execute and must be one of the following: GDRAW, GINIT, GPRINT, GRAPH, GSET, or GTERM.

operator 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, *operator* can take different values depending on *function-name*.

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 1033 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 32, "DATA Step Graphics Interface Dictionary," on page 1075 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 1050.

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 9, “Graphics Options and Device Parameters Dictionary,” on page 301 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 51 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 34 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 9, "Graphics Options and Device Parameters Dictionary," on page 301.

DSGI Function	Graphics Option That Is Overridden
GSET('CBACK', . . .)	CBACK=
GSET('COLREP', . . .)	COLORS=
GSET('DEVICE', . . .)	DEVICE=
GSET('HPOS', . . .)	HPOS=
GSET('HSIZE', . . .)	HSIZE=
GSET('VPOS', . . .)	VPOS=
GSET('VSIZE', . . .)	VSIZE=

DSGI Function	Graphics Option That Is Overridden
GSET('TEXCOLOR', . . .)	CTEXT=
GSET('TEXFONT', . . .)	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 1044 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 32, "DATA Step Graphics Interface Dictionary," on page 1075 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 1165 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 31.1 on page 1034 summarizes the types of operations available and the functions used to invoke them. Refer to Chapter 32, "DATA Step Graphics Interface Dictionary," on page 1075 for details about each function.

Table 31.1 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('*xxx*INDEX', . . .)	selects the bundle of attributes to use
	GSET('*xxx*REP', . . .)	assigns attributes to a bundle
Setting Attributes That Affect Graphics Elements		
color index	GSET('COLREF'), . . .)	assigns a color name to color index

DSGI Operations	Associated Function	Function Description
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
	GSET('CATALOG', . . .)	selects the catalog to use

DSGI Operations	Associated Function	Function Description
	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(*code*)	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 31.2 on page 1038 summarizes the types of values that the GASK routines can check. Refer to Chapter 32, "DATA Step Graphics Interface Dictionary," on page 1075 for details about each routine.

Table 31.2 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('*xxx*INDEX', . . .)	returns the index of the active bundle
	GASK('*xxx*REP', . . .)	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.

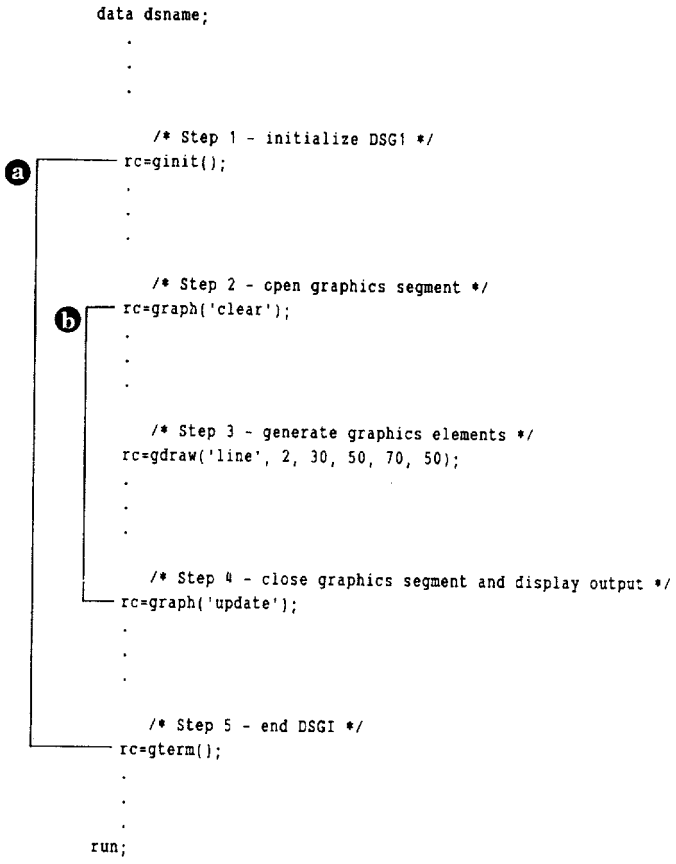
5. End DSGI.

   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 31.1 on page 1042 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 31.1 Basic Steps Used in Creating DSGI Graphics Output



Notice that there are two pairs of functions that work together within a DSGI DATA step (shown by a and b in Figure 31.1 on page 1042). 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 1054 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 1044.

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 31.3 on page 1043 lists the operators used with GDRAW functions to generate graphics elements and the attributes that control them.

Table 31.3 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 32, "DATA Step Graphics Interface Dictionary," on page 1075 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 → WSAC → SGOP → WSAC → 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 → WSAC
GRAPH('CLEAR', . . .)	WSAC → SGOP
GRAPH('UPDATE', . . .)	SGOP → WSAC
GTERM()	WSAC → 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 1076 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.

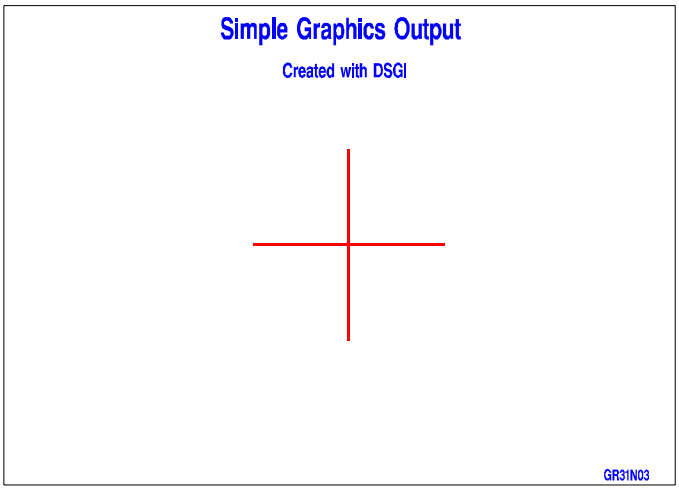
Notice that you can set attributes for the graphics primitives in several places. As long as the functions that set the attributes are executed before the graphics primitives, they will affect the graphics output. If you execute them after a graphics primitive, the primitive is not affected. See "Setting Attributes for Graphics Elements" on page 1042.

The following program statements illustrate a more complex DSGI program that produces Display 31.3 on page 1047 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 1076 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, 'GR31N03  ');

      /* 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 31.3 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 31.4 on page 1047 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 31.4 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, TEXFONT

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 1134 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. △

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 1133.

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 Viewpoints":

```
/* 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 1062 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. △

Activating Transformations

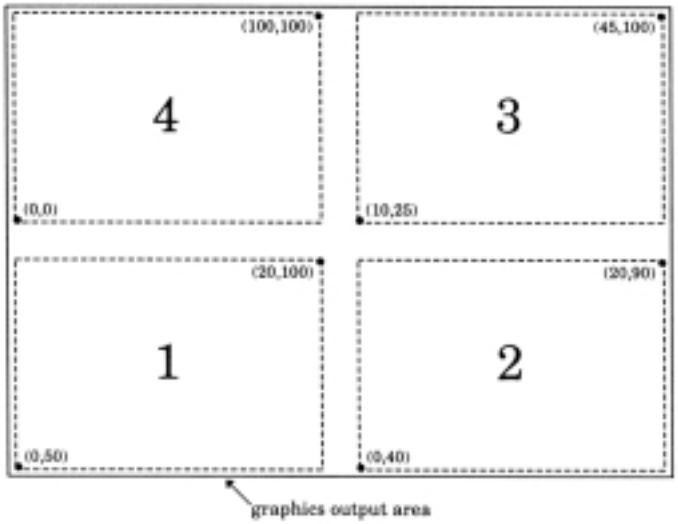
Once you have defined a viewport or window, you must activate the transformation in order for DSGI to use the viewport or window. To activate the transformation, use the GSET('TRANSNO',*n*, . . .)function where *n* has the same value as *n* in GSET('VIEWPORT',*n*, . . .)or GSET('WINDOW',*n*, . . .).

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 31.2 on page 1053.

Figure 31.2 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', . . .). △
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 31.1 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. △

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 31.1 on page 1042.) 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 32, "DATA Step Graphics Interface Dictionary," on page 1075 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=

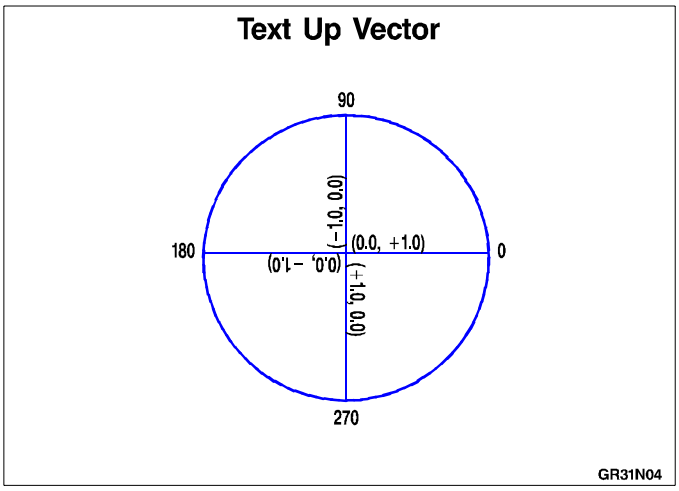
graphics option provides a colors list from which the colors referenced in GSET('*xxx* COLOR', . . .)functions are selected.

The following program statements produce Display 31.4 on page 1057:

```
/* 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 */
footnote1 j=r 'GR31N04  ';
title1 '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 31.4 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.

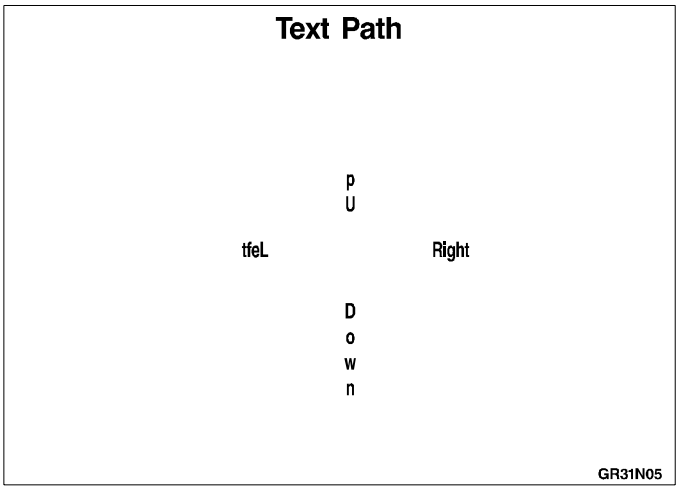
Changing the Reading Direction of the Text

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 31.5 on page 1059:

```
    /* 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 */
footnote1 j=r 'GR31N05  ';
title1 '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 31.5 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

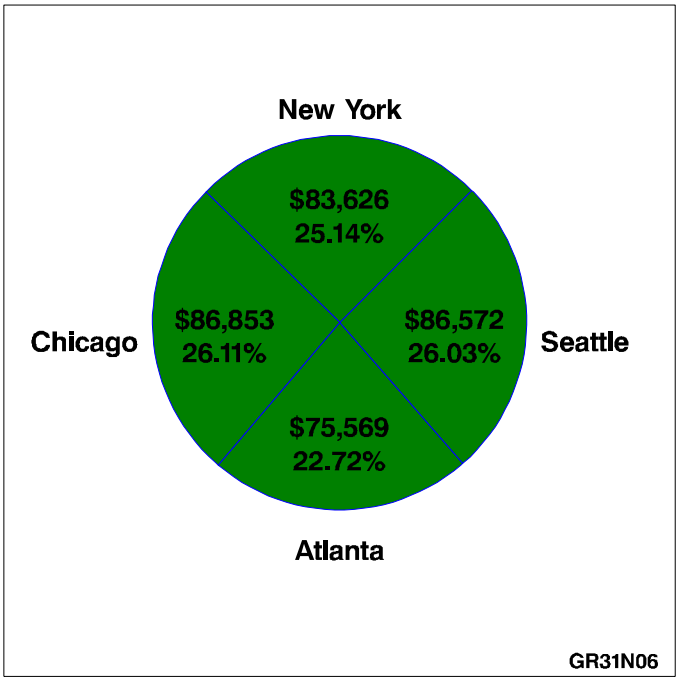
This example uses the GCHART procedure to generate a graph, defines a viewport in which to display it, and inserts the GCHART graph into the graphics output being created by DSGI. Display 31.6 on page 1061 shows the pie chart created by the GCHART procedure. Display 31.7 on page 1062 shows the same pie 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;
```

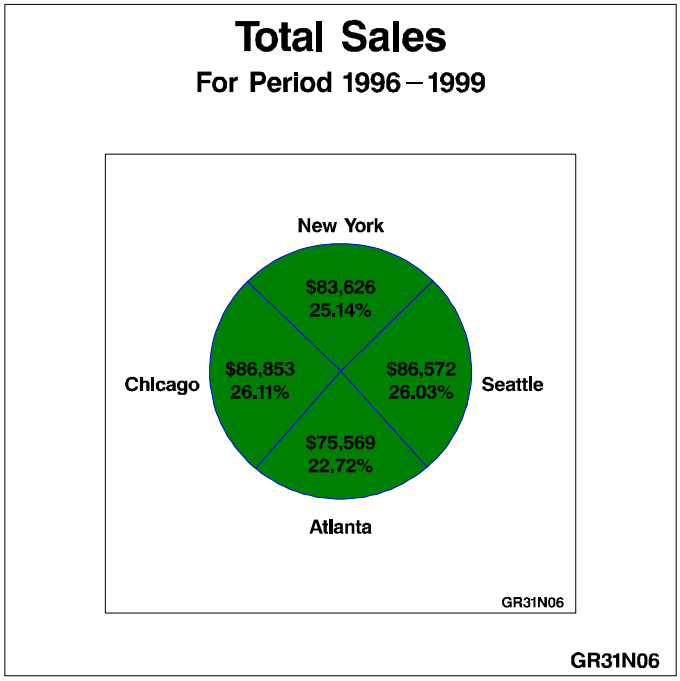
```
   /* create data set TOTALS */
data totals;
   length dept $ 7 site $ 8;
   do year=1996 to 1999;
      do dept='Parts','Repairs','Tools';
         do site='New York','Atlanta','Chicago','Seattle';
            sales=ranuni(97531)*10000+2000;
            output;
         end;
      end;
   end;
run;

   /* define the footnote */
footnote1 h=3 j=r 'GR31N06  ';

   /* 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='gr31n06';
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', 'gr31n06');

      /* display graph and end DSGI */
   rc=graph('update');
   rc=gterm();
run;
```

Display 31.6 Pie Chart Produced with the GCHART Procedure



Display 31.7 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, 'GR31N06', to be inserted.
- The GSET('VIEWPORT', . . .)function defines the section of the graphics output area into which GR31N06 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, 'GR31N06', 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

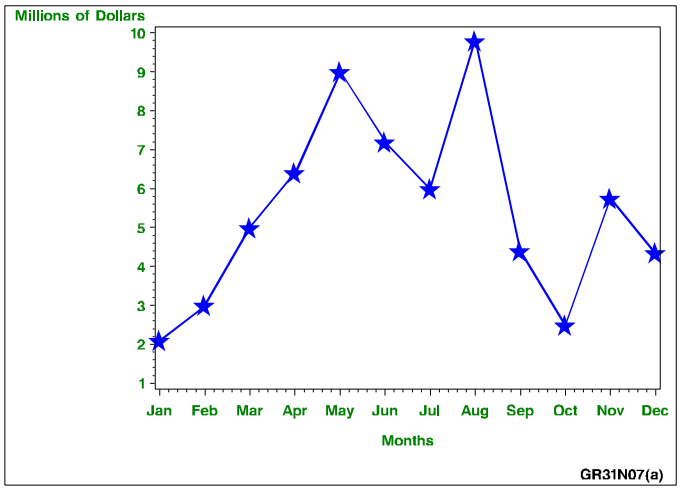
This example uses the GPLOT procedure to generate a plot of AMOUNT*MONTH and store the graph in a permanent catalog. DSGI then scales the graph by defining a window in another DSGI graph and inserting the GPLOT graph into that window. Display 31.8 on page 1064 shows the plot as it is displayed with the GPLOT procedure. Display 31.9 on page 1065 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
```

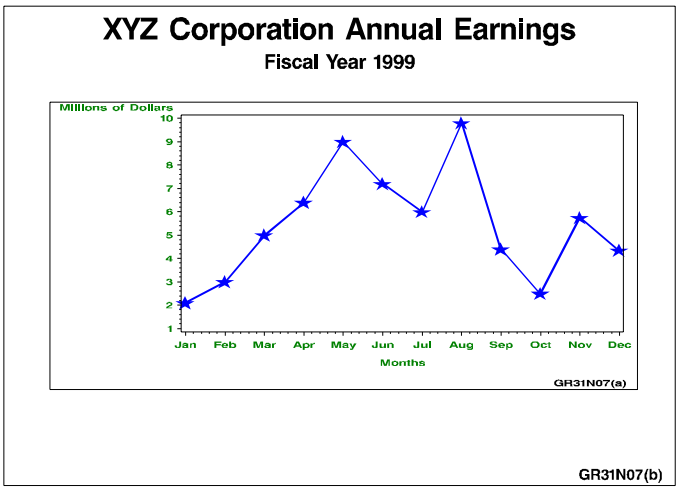
```
         ftext=swissb htitle=6 htext=3
         colors=(black blue green red)
         hsize=7 in vsize=5 in
         targetdevice=pscolor;

   /* 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 */
footnote1 j=r 'GR31N07(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 GR31N07                 */
proc gplot data=earn;
   plot amount*month
        / haxis=axis2
          vaxis=axis1
          name='gr31n07';
run;

   /* define the footnote and titles for       */
   /* second graph, which will scale output */
footnote1 j=r 'GR31N07(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', 'gr31n07');

      /* display graph and end DSGI */
   rc=graph('update');
   rc=gterm();
run;
```

Display 31.8 Plot Produced with the GPLOT Procedure



Display 31.9 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.

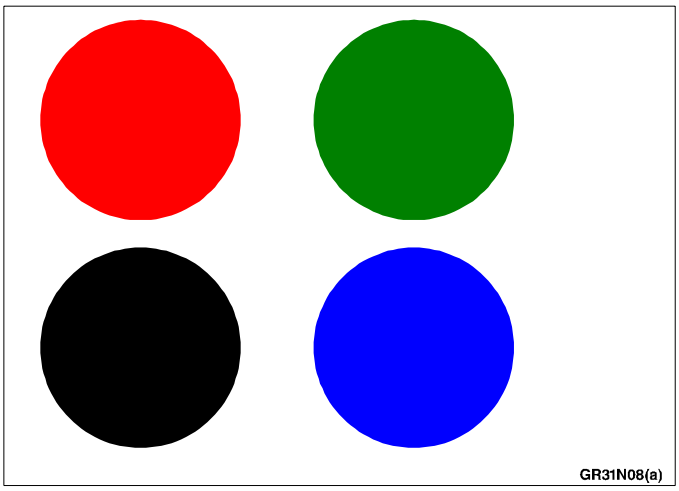
Enlarging an Area of a Graph by Using Windows

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 'GR31N08' into that window. The following program statements produce Display 31.10 on page 1067 from the first DATA step, and Display 31.11 on page 1067 from the second DATA step:

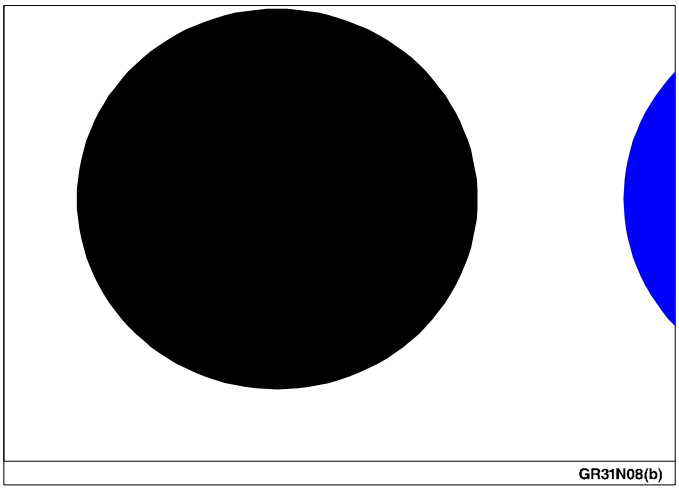
```
   /* set the graphics environment */
goptions reset=global gunit=pct border
         ftext=swissb htext=3
         colors=(black blue green red)
         hsize=7 in vsize=5 in
         targetdevice=pscolor;

   /* define the footnote for the first graph */
footnote1 j=r 'GR31N08(a)  ';

   /* execute DATA step with DSGI */
data plot;

      /* prepare SAS/GRAPH software */
      /* to accept DSGI statements  */
   rc=ginit();
   rc=graph('clear', 'gr31n08');

      /* 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 */
footnote1 j=r 'GR31N08(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', 'gr31n08');

      /* display graph and end DSGI */
   rc=graph('update');
   rc=gterm();
run;
```

Display 31.10 Four Pie Charts Generated with DSGI



Display 31.11 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 31.1 on page 1068 shows how the value appears in the log. Output 31.2 on page 1069 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 31.1 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 31.2 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 accross 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 5, "Bringing SAS/GRAPH Output to the Web," on page 71. For specific information about drill-down graphs, see "About Drill-down Graphs" on page 90. △

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 31.12 on page 1071. To drill down to the graph shown in Display 31.13 on page 1072, 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 tile and footnote for graph */
title1 'Drill-down Graph';
footnote1 j=l '  Click in pie chart'
          j=r 'GR31N10  ';

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;
footnote1;
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 31.12 Drill-down Graph Generated with DSGI

Display 31.13 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= specifes 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 49
for an explanation of graphics catalogs and catalog entries

Chapter 9, "Graphics Options and Device Parameters Dictionary," on page 301
for complete information about graphics options

"TITLE, FOOTNOTE, and NOTE Statements" on page 251
for details of using the TITLE and FOOTNOTE statements

"GOPTIONS Statement" on page 182
for details of using the GOPTIONS statement

Chapter 10, "The Annotate Data Set," on page 403
for an explanation of the Annotate facility

Chapter 32, "DATA Step Graphics Interface Dictionary," on page 1075
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

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/GRAPH® Software: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS/GRAPH® Software: Reference, Version 8

ISBN 1-58025-525-6

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

1st printing, October 1999