DSGI: The OTHER Way To Do SAS Graphics
C. Michael Whitney - USAF Environmental Technical Applications Center

Abstract
This paper will discuss the use of the SAS® Data Step Graphics Interface (DSGI). DSGI provides a flexible means of creating graphics and is able to handle most of your special graphic needs. This paper will discuss the basics of DSG and show complex graphs can be created using simple shapes.

Introduction
"Windrose" is a USAF Environmental Technical Application Center (USAFETAC) program that creates a graphic representation of the average wind speeds and directions, and the frequencies of their occurrence for a given site. These graphs are called windroses, hence the name of the program. In early 1991, the graphics generation routine was being handled by PROC ANNOTATE, and only printed one graph for the surface winds per site. A request for customer specified heights was received, resulting in multiple graphs per site, which meant major changes in the code. After reviewing the options, it was determined that the SAS Datatstep Graphics Interface (DSGI) would be the best graphics generation method, since it allows the creation of graphics from within the data step, something not possible with the Annotate facility.

Comparison With the Annotate Facility
Both Annotate and DSGI allow you to enhance graphics output produced by other SAS/GRAPH procedures. Most of the SAS graphics tools allow for calling in an Annotate data set for this purpose, making modifications via Annotate a lot easier than doing it with DSGI.

DSGI is best used when creating custom graphics without another graphics procedure. With the Annotate facility, you must first create an Annotate data set and then submit a PROC step to display the graphics output. DSGI allows you to do all of this within the DATA step, eliminating the need for a separate PROC to be run. Since this extra step doesn’t need to be performed, DSGI graphic programs are generally faster than Annotate ones. DSGI functions and routines can be used in either the DATA step or in an SCL application. Thus, this paper will focus on creating, instead of modifying, graphics output using DSGI.

Using DSGI To Make Custom Graphics
DSGI allows for the creation of custom graphs using the following basic graph elements: arcs, bars, ellipses, elliptical arcs, lines, markers, pie slices, polygons (filled and unfilled areas), and text. The color, pattern, size, style and position of these elements can be specified by the programmer, allowing diversity in final output.

Components of DSGI
There are five basic steps to using DSGI: initializing DSGI, opening a graphic window, drawing your graph, closing the window, and ending DSGI. The initialization and closing steps must be done once per graphic screen. Multiple graphics can be created on one screen.

There are also five operating states: GKCL (graphics kernel closed), the initial state of DSGI, where no graphics resources have been allocated; GKOP (graphics kernel open), where you can check the current attribute settings; SGOP (segment open), where you can generate graphics; WSAC (workstation active), when active, you can issue commands; and WSOP (workstation open), which allows access to the graphics catalog.

There are five types of commands: utility functions that allow you to initialize DSGI; GRAPH commands to perform graphics library management functions; GASK routines, that allow you to check the current attribute settings; GDRAW functions, that allow you to create your graphic elements; and GSET functions that allow you to set the attributes of the graphics elements that your GDRAW commands will use.
The basic DSGI output area is the region of the screen, using a cartesian coordinate system of 100 x units by 100 y units. (0,0) is in the bottom left while (100,100) is in the top right. The center point is at the (50,50) coordinate location. These units are arbitrary, and can be set to some other amount if you wish, say -200 through 200 x, -50 through 50 y.

**Using DSGI**

For windroses a dataset containing the month, the wind directions, wind speeds in 5 categories, and the frequency of occurrence of the previous two is used. The dataset sometimes has the height and pressure levels as well.

DSGI commands are issued as function calls, in the form of

```plaintext
return_code = function(operator, arguments);
```

where return_code is a numeric variable that holds the return code from the function. These return codes are used in debugging, where a good run returns a 0, and a less than good returns a wide variety of numbers. The SAS/GRAPH Software Vol 1 book lists each commands possible return codes in the DSGI Interface Dictionary sections, and a complete list of what the error codes mean in the DSGI Return Codes section. Both of these sections are located in chapter 21.

**Initializing DSGI**

GINI() is used to initialize a DSGI session. The horizontal and vertical sizes of the graph output is set with the HSIZE and VSIZE GSET commands.

Here's a description of the commands used to do initialization a DSGI session:

GINI() - Initializes the DSGI system.

GSET(HSIZE, n); - Used to set the horizontal size of the graph, same as HSIZE= goption.

GSET(VSIZE, n); - Used to set the vertical size of the graph, same as VSIZE= goption.

GSET(CATALOG, libref, catalog-name); - Allows graphs to be copied into a specified library.

GRAPH(CLEAR, name, description, byline); - Opens a graph for output into current catalog. NAME, description and byline will appear in catalog listings and info.

GSET(LINCOLOR, color-index); - Sets the line color, using color-index. color-index is between 1 and 256.

GSET(TEXFONT, font); - Sets the font, where font is any valid SAS font name enclosed in quotes.

GSET(TEXHEIGHT, height); - Sets the height of your text, using a numeric value greater than 0.

Now, to demonstrate how all this goes together. Here is a macro that will be called later to initialize a DSGI session, and set up some defaults.

```plaintext
%MACRO INITDSGI;
   / Set graphic size to 16x11 */
   RC=GSET(HSIZE, 16);
   RC=GSET(VSIZE, 11);
   / Set up the graphics library & catalog to use. */
   RC=GSET(CATALOG, &GLIB", "&GCAT);
   / Initialize the DSGI system, and clear the graph screen, calling it the name held in &GNAME */
   RC=GINI();
   RC=GRAPH("CLEAR", ";
   / Initialize the text alignment, text height, text font, and colors here */
   RC=GSET(TEXALIGN, CENTER, "HALF";
   RC=GSET(TEXCOLOR, &COLOR);
   RC=GSET(TEXHEIGHT, 2); RC=GSET(TEXFONT, ";
   / Set the line color */
   RC=GSET(LINCOLOR, 3);
%MEND INITDSGI;
```

**Ending DSGI**

To end DSGI, we use the GTERMO() command.

The commands we will use are:

GRAPH(UPDATE, show); - Closes the graph and shows it, if the show keyword is included.

GTERMO(); - Ends the graph, and closes DSGI.

%MACRO ENDDSGI;
   / Close off graph & output */
Building a Windrose

Circles

The first thing we need is an inner circle, and to make one, we use this command:

```gdraw(, arc', x, y, radius, start, end); - Draws a circular arc.
```

x, y are the origin points; radius is the length of the radius; start is the starting angle, 0 is at the 3 o'clock position; end is the ending angle.

```data _null_;
/* Initialize DSGI */
%INITDSGI;
/* Make a circle centered at (50,50) with a radius of 3. Start at 0 and end at 360. */
rc=gdraw(,arc', 50, 50, 3, 0, 360);
/* End DSGI */
%ENDDSGI;
run;
```

This macro, when run, creates a circle centered on the screen, with a radius of 3 units. This is the basis of a Windrose.

Polygons

The next step in creating the graph is to draw the 'leaves' of the rose. The commands used are:

```gdraw(fill', n, x-vals, y-vals); - Draws a filled polygon. n is the number of sides the polygon has; x-vals is a list of the x coordinates for each vertex point; y-vals is the same for the y coordinates.
```

```gset(filltype', interior); - Sets the fill type for interior fills. interior can be set to hatch, hollow, pattern, or solid.
```

In order to determine the X and Y coordinates of the start and stop points of each leaf (a leaf being each rectangle in our graph), simple trigonometry is used to convert each 22.5 degree angle into its sine and cosine. These values are then put into 16 element arrays. In order to calculate the new X and Y, the initial distance from the initial XY is multiplied by the array element, and added to the initial XY. The distance between the XY start and endpoints is determined by the percent frequency of the particular windspeed in the direction indicated by the leaf.

```x_0, y_0
x_1, y_1
x_n, y_n
```

We start at our origin X_0, Y_0 and draw our lines connecting our calculated corner points, ending at the origin. After we're done, we set X_n, Y_n to be our new origin for the next leaf. The distance each corner X, Y is from the origin depends on how great the windspeed grouping is. The faster the winds, the thicker the leaf.

Here's some example code for creating a windrose, using the circle example from before and the filled polygon:

```%macro drawrose;
/* Our circle from the last example */
rc=gdraw(,arc', 50, 50, 3, 0, 360);
/* Set up arrays holding our trig values to calculate our leaf angles, and the corners of our leaf boxes. */
/* sin 0 = 0, sin 22.5 = .383, etc */
/* cos 0 = 1, cos 22.5 = .924, etc */
array dx(0:15) (.383 .707 .924 .924 .707 .383 0 -.383 -.707 -.924 -1 -.924 .707 .383 .707 .924);
array dy(0:15) (1 .924 .707 .383 0 -.383 -.707 -.924 -1 -.924 -.707 -.383 .707 .924);
/* Subtract 90° from each angle, get the sin of X and cosines of Y */
array px(0:15) (-1 -.924 -.707 -.383 0 .383 .707 .924 .924 .707 .383 0 -.383 -.707 -.924);
array py(0:15) (0 .383 .707 .924 1 .924 .707 .383 0 -.383 -.707 -.924 -1 -.924 .707 .383 .707 .924);
/* Array that holds the values of each leaf thickness */
array width(1:5) (.25 .5 .5 .75 .751);
/* Set the initial X-Y locations to work from */
inity = 50;
inix = 50;
/* In the real world, there would be 16 wind directions, but here a loop is used to simulate them. */
```

735
DO ANGLE = 0 TO 15;
/* Set the origin for each leaf set. The multiplier of three sets the origin at the edge of the circle. This gives the angle from our DX and DY array. */
ORIGX = INITX + DX(ANGLE) * 3;
ORIGY = INITY + DY(ANGLE) * 3;
/* Normally, there'd be 4 or 5 wind speed categories, but for this example a loop is used instead. */
DO LEAF = 1 TO 5;
/* And, the length of each leaf is controlled by the frequency of observations matching the direction and speed category group. However, a random length is used instead for this example. */
RANDLGTH = RANUNI(ANGLE) * LEAF);
END;
END;
/* Now, set the origin coords to the end coords so that the next leaf starts where the first left off. */
ORIGX = ENDX;
ORIGY = ENDY;
/* End the loops. */
END;
END;
%MEND DRAWROSE;
The following datastep pieces it together, and draws a windrose:
DATA _NULL_;
%INITDSGI;
/%DRAWROSE;
%ENDDSGI;
RUN;
Here's the output from this step:

Figure 2

Text

Looks pretty good, but some text is needed! Text is handled by the following command:

GDRAW(TEXT, x, y, string); - Prints text string at x-y coordinates. The string can be either a quoted text, or a variable that equates to a string of text.

And it used like this:
%MACRO DRAWTEXT;
/ Normally the number of calm winds is put in
 the middle of our rose. For this example, a
 a zero in simply placed there. */

RC=GDRAW('TEXT, 50, 50, '0');
/ And all good graphs have a title of some sort.
 Here a variable is set to hold a text string and
 place it below the graph. */

WRTITLE = 'AN EXAMPLE WINDBESE';
RC=GDRAW('TEXT, 50, 20, WRTITLE);
%MEND DRAWTEXT;

After the text macro to the previous code examples, the
 output looks like this:

Figure 3

Lines

The line command is used to draw the lines in the legend. No
 example of it is given here, because the rose is redrawn with
 a series of leaves eminating at the 3 o'clock direction, with the
 legend tick marks placed beneath it. Here is the syntax to
 use it:

GDRAWLINE, n, x-vals, y-vals); - Draws a line, or series of
 connected lines. n is the number of corner points used; x-vals
 is a list of the x coordinates for each vertex point; y-vals is
 the same for the y coordinates.

In the windrose program, the position for each little segment
 in the legend scale is calculated by looping through and
 moving a bit over each time, drawing a segment each time.

Other Graphs Created by DSGI

Here's a couple other graphs that are created in the Windrose
 program, using DSGI, and a brief description of how they are
 created using these graphic basics.

Multiple Roses On One Chart

In Figure 5, the DSGI session is not closed between graphs,
 but rather the origin point of each graph is changed. Also, the
 origin of the legend has been shifted to the right of each rose,
 instead of under it.

Figure 4

Engineering Rose

The rose shown in Figure 5 was requested by an engineering
 client, who wanted hard numbers instead of the graphical
 rose. For this, the bulls-eye was created using the Arc and
 Line commands, and the numbers were placed using the Text
 command.

Note that this graph is for the same data as the graph shown
 in Figure 1.

Conclusions

Hopefully, this is enough to get you started using the Data
 Step Graphics Interface. These basic building blocks can be
 combined in many ways in order to build complex graphs.
 DSGI is a very useful tool when creating custom graphs.

737
References

Cary, NC: SAS Institute, Inc.

SAS and SAS/GRAPH are registered trademarks of SAS Institute, Inc in the USA and other countries.

Contacting the Author

C. Michael Whitney, SSgt, USAF
USAFETAC/SYS
Bldg. 859
Scott AFB, IL 62225-5116
(618) 256-5323 DSN 576-5323
Internet: syscmw@lightning.safb.af.mil

Figure 5