# An Introduction to the DATA Step Graphics Interface

Earl Westerlund, Eastman Kodak Company, Rochester, NY

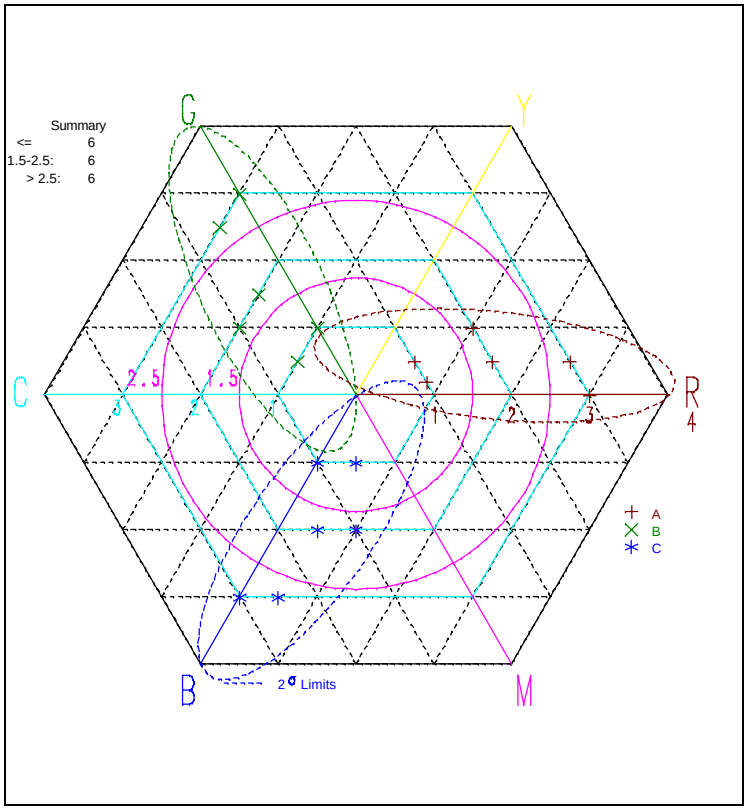
## INTRODUCTION

The DATA Step Graphics Interface (DSGI) is a component of SAS/GRAPH® software that enables you to generate graphics output from within a DATA step, macro or SCL application. It consists of low-level graphics routines that allow you to develop a custom graph, or add elements to an existing graph.

This paper will describe the steps involved in creating a graph in the DATA step, from opening a session to storing a graphic segment. A brief comparison of DSGI with other options for creating graphics will be given.

## WHY WOULD YOU WANT TO USE DSGI?

SAS/GRAPH software provides a set of procedures that allow you to produce a wide variety of standard graphics, including two-dimensional (XY) and three-dimensional (XYZ) plots, many kinds of business graphs, and maps. But, what if the graphic you need looks something like the following:




This is a kind of graphic we use at Kodak to evaluate the color balance of photographic materials. We wanted the ability to generate such a graph within the SAS® System, as part of a researcher's or engineer's analysis. None of the standard SAS/GRAPH procedures could easily produce this, however. Our solution was to write a DSGI program.

## OTHER WAYS TO PRODUCE GRAPHICS IN THE SAS SYSTEM

To solve our problem, we used DSGI. But there are other ways to generate graphs in the SAS System. Following is a quick review of the other ways to generate graphics, along with my opinion on their appropriate areas of use.

### SAS/GRAPH Procedures

These are by far the easiest to use, they are the fastest, and most are flexible – within the context of the kinds of plot they are designed to generate. For example, you can create many kind of plots with the GPLOT procedure, as long as they are two-dimensional plots within coordinate axes. Another advantage of SAS/GRAPH procedures is support for BY variables.

But desired options don't always exist: for example, the ability to display the value of a third variable on a scatter plot. The same is true for other SAS/GRAPH procedures, and procedures in other products that generate graphics.

### The Annotate Facility

The annotate facility uses instructions that are stored in a SAS data set with specifically named variables. It is very

similar in function to DSGI, in that both are collections of graphics primitives routines. The graph shown above could have been created using the annotate facility.

The annotate facility was designed to work with and complement SAS/GRAPH procedures, allowing for data-driven customization of output. It was not intended primarily to generate full custom graphics. It is slower than DSGI.

### The Graphics Editor

The graphics editor has the standard tools of a simple drawing program. It is primarily for interactive modifications of graphics. For example, you can call attention to a point on a plot by drawing an arrow to the point and adding some explanatory text. If the annotation is repetitive and data-driven, it would be better to use the annotate facility.

### IML Graphics

SAS/IML® software contains graphics routines that are similar in nature to DSGI. If the rest of your program is written in IML, then you should probably use IML graphics.

## TERMINOLOGY

To understand DSGI, you must first understand the terminology. Following are some of the more important terms:

- A **graphics primitive** is a single graphic element, such as a line, bar, or string of text.
- A **segment** is essentially a plot. It is a collection of graphics primitives that make up a single catalog entry.
- A **workstation** is the device you are writing to, whether a monitor or hard copy device.
- A **viewport** defines what fraction of the available area of the workstation to use when generating the graph. It is somewhat analogous to a panel definition in the GREPLAY procedure. If not specified, the entire area is used: (0, 0) at the lower left corner to (1, 1) at the upper right corner.
- A **window** defines what coordinate units to use in the viewport. This allows you to work in the units that are native to your applications. If not specified, the coordinates range from (0, 0) at the lower left corner to (100, 100) at the upper right corner of the viewport.
- A **transformation** defines which viewport-window combination to use when generating your graph. Viewports and windows are linked by a transformation number. You can define up to 20 transformations in a DSGI program, beyond the default "transformation 0," which has the default viewport and window settings and cannot be altered.

## FUNCTIONS AND ROUTINES

There are only five functions and one call routine in DSGI. Two of the functions merely start and end DSGI, and the one call routine allows you to query settings. So there are only three functions that you use to control DSGI output. Of course, these three functions do a lot!

- The **GINIT** function starts DSGI and opens a workstation.

  Usage: **`rc = GINIT();`**

  Where `rc` is a return code variable that indicates whether the function was successfully carried out.

- The **GRAPH** function performs library management tasks:

  Usage: **`rc = GRAPH('task', other-args);`**

  The tasks are:

  - **`CLEAR`**: opens a graphics segment for output, clearing out existing segments
  - **`COPY`**: copies a graph within a catalog
  - **`DELETE`**: deletes a graph from a catalog
  - **`INSERT`**: inserts a graph into the currently open graphics segment
  - **`RENAME`**: renames a graph
  - **`UPDATE`**: closes the currently open graph and optionally displays it

- The **GDRAW** function is what actually draws graphic primitives.

  Usage: **`rc=GDRAW('primitive',other-args);`**

  There are only nine kinds of primitives.

  - **`ARC`**: circular arc
  - **`BAR`**: rectangle
  - **`ELLARC`**: elliptical arc
  - **`ELLIPSE`**: ellipse
  - **`FILL`**: filled polygon
  - **`LINE`**: polyline (empty polygon)
  - **`MARK`**: marks (points)
  - **`PIE`**: filled pie slice or circle
  - **`TEXT`**: text string

- The **GSET** function sets attributes, such as the current output catalog, output device, text font, line colors, and so on.

  Usage: **`rc=GSET('attribute', other-args);`**

- The **GASK** call routine queries DSGI about attribute settings, such as the name of. the current output catalog, output device, text font, and so on.

  Usage: **`CALL GASK('oper',other-args,rc);`**

There are too many GASK operators to be listed here. See the SAS/GRAPH Software Reference Guide (Chapter 1) for a complete list.

- The **GTERM** function closes DSGI. There are no arguments to the function.

  Usage: **`rc = GTERM();`**

## CREATING A GRAPHICS SEGMENT

Summarizing the above, there are six steps to creating a graph using DSGI:

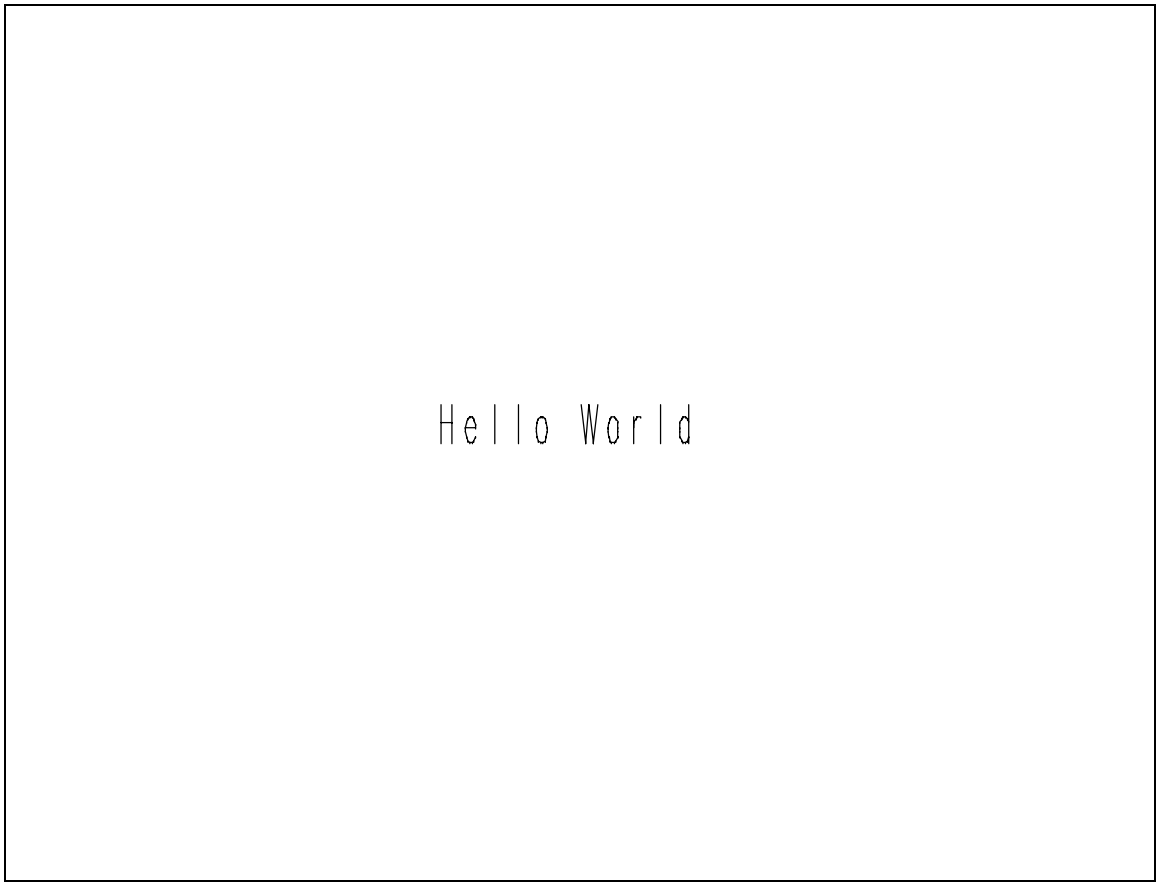
1. **Initialize DSGI:** Initialize the DSGI environment with the GINIT function, making the routines available for you to call.
2. **Open a new segment:** Use the GRAPH function to start a new graph or use a previously created one.
3. **Define the coordinate system and portion of the display area to use:** Use the GSET function to define what part of the display area to use, and what your coordinate system will be. (There are default values, so you do not need to do this explicitly.)
4. **Generate graphics elements:** Finally, you get to draw things!
5. **Close the segment, and store and display the graph:** Use the GRAPH function to store the graph, and display it (or not).
6. **End DSGI:** Use the GTERM function to close DSGI.

## CREATING A GRAPHICS SEGMENT: A SIMPLE EXAMPLE

This first example gives a simple example of writing a text primitive. We are using the default viewport (the entire plotting area) and window (X and Y coordinates of 0 to 100).

```
goptions reset=goptions;
data _null_;
  * Step 1: start DSGI;
  rc = ginit();

  * Step 2: open a new segment named TEXT;
  rc = graph('clear', 'text');

  * Step 3: generate an element (a text
   string);
  rc = gdraw('text', 50, 50, 'Hello
   World');

  * Step 4: store and display graph;
  rc = graph('update');

  * Step 5: end DSGI
  rc = gterm();
run;
```

The result of this program follows:

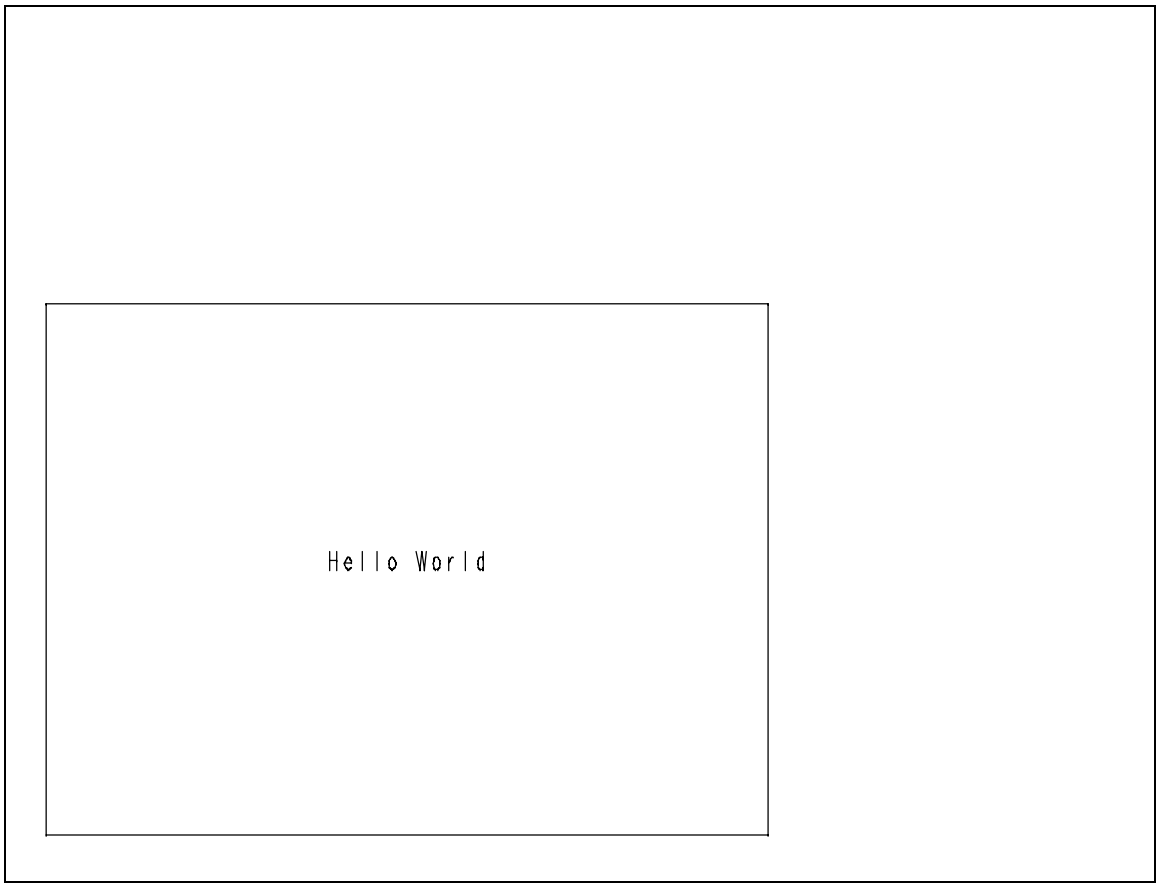


## USING A TRANSFORMATION

This example shows the use of windows and viewports. The coordinate system (window) has been redefined so that the (0, 0) coordinate is the center of the plot, and we are only using the lower left corner of the available display area (viewport).

```
data _null_;

  * Step 1: start DSGI;
  rc = ginit();

  * Step 2: open a new segment named TEXT;
  rc = graph('clear', 'text');
  * Use lower left corner only;
  rc = gset('viewport', 1, 0, 0, .7, .7);
  * Units are 0-50 in both directions
  rc = gset('window', 1, -50, -50, 50,
   50);
  * Use this viewport/window combination;
  rc = gset('transno', 1);

  * Step 3: generate graphics elements;
  rc = gset('texheight', 4);
  * Center the text;
  rc = gset('texalign', 'center', 'base');
  * Draw the text;
  rc = gdraw('text', 0, 0, 'Hello World');
  * Draw a box around the text;
  rc = gdraw('bar', -45, -45, 45, 45);

  * Step 4: display and store graph;
  rc = graph('update');

  * Step 5: end DSGI;
  rc = gterm();
run;
```

And here is the graphic segment that results:



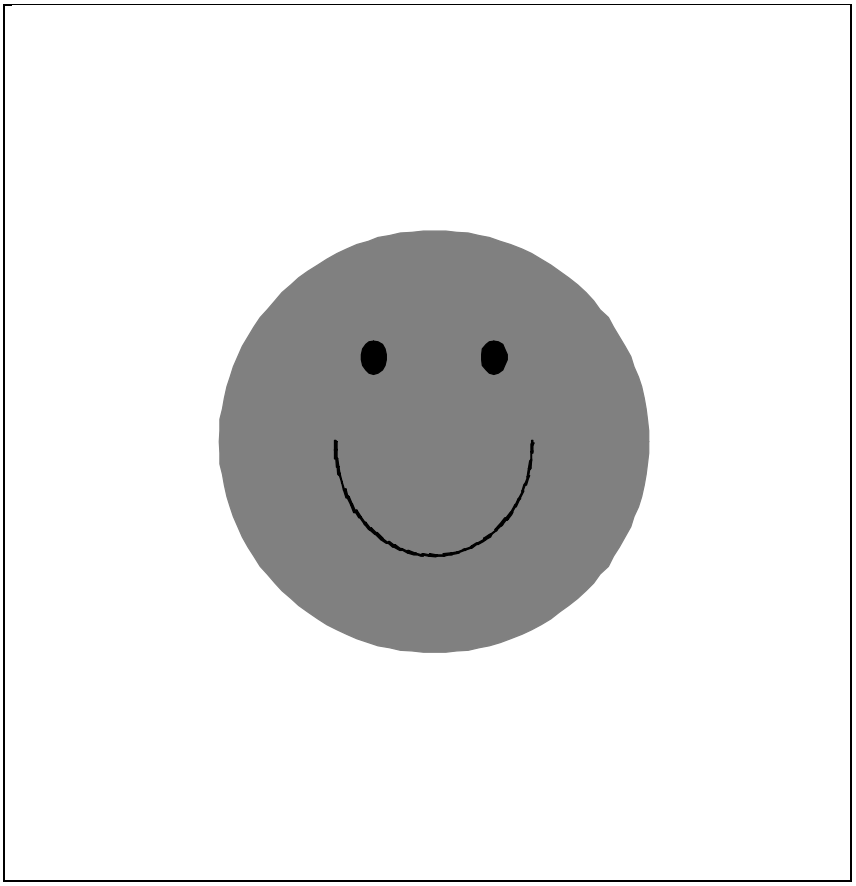

## SETTING ATTRIBUTES

This final example illustrates setting attributes such as colors and line widths.

```
data _null_;

  * Step 1: start DSGI;
  rc = ginit();

  * Step 2: open a new segment GRYNNE;
  rc = graph('clear', 'grynne');

  * Step 2.5: set color indices;
  * Color 1 will be yellow,
    color 2 will be black;
  rc = gset('colrep', 1, 'yellow');
  rc = gset('colrep', 2, 'black');

  * Step 3: generate graphics elements;
  * Set the fill color to 1 (yellow);
  rc = gset('filcolor', 1);
  * Make a solid fill;
  rc = gset('filtype', 'solid');
  * Draw a 360 degree pie slice (circle);
  rc = gdraw('pie', 50, 50, 25, 0, 360);

  * Reset the fill color to 2 (black);
  rc = gset('filcolor', 2);
```

```
* Draw an ellipse at (43, 60);
rc = gdraw('ellipse', 43, 60, 4, 3, 0,
 360, 90);
* Draw another one at (57, 60);
rc = gdraw('ellipse', 57, 60, 4, 3, 0,
 360, 90);

* Set the line color to 2 (black);
rc = gset('lincolor', 2);
* Set the line width to 2 (double thick-
 ness);
rc = gset('linwidth', 2);
* Draw an ellipse arc;
  rc = gdraw('ellarc', 50, 50, 27, 23, 90,
   270, 90);

  * Step 4: display graph;
  rc = graph('update');

  * Step 5: end DSGI;
  rc = gterm();
run;
```

The resulting plot is below. I apologize to anyone who is offended by its content:



## OTHER (POSSIBLY) USEFUL THINGS YOU CAN DO WITH DSGI

### Adding to an existing graph

You can open a graph that was created in a SAS/GRAPH procedure and add graphics primitives to it later. While you would normally use the annotate facility to annotate output from a graphics procedure, you can do "post hoc" annotation of a graph by inserting the graph into an open segment, then adding graphics primitives. Consider using the graphics editor if it is available to you.

### Displaying multiple graphs

You can create multiple viewports and insert different segments into each viewport. This would normally be done with templates in the GREPLAY procedure, but it is possible to do with DSGI.

### Using DSGI to query graphic options

There is no dictionary table for you to query GOPTIONS settings, as there is for system options. But you can find out many of these settings through the GASK routine of DSGI.

For example, we needed to be able to define a square plotting area for plots that required the X axis to be exactly the same length as the Y axis. In other words, the HSIZE= and VSIZE= option settings needed to be the same. But we did not want to hard code the settings, as we have many different display and hard copy devices to support. To solve the problem, we wrote a macro that uses DSGI calls to find the maximum default horizontal and vertical lengths and put the minimum into a macro variable. Then we used the macro variable in the GOPTIONS statement. This macro is in the appendix.

## SUMMARY

DSGI is a useful tool in situations where a complex graph is needed, and other options are not practical or feasible. You should be aware of its capabilities if you create complex graphs in SAS.

## APPENDIX A: WAYS TO GET GRAPHS

| Method | Advantages | Disadvantages |
|---|---|---|
| Graphics procedures | • Fastest<br>• Easiest to use<br>• Support for BY variables | • Least flexible |
| Annotate facility | • Best used with graphics procedures<br>• Data driven customization of graphics<br>• Support for BY variables | • Slower than DSGI<br>• Requires creation of data set with specific variable names |
| Graphics editor | • For editing of generated graphics<br>• Quick one-shot editing | • Not for repeated modification or annotation |
| SAS/IML Graphics | • Similar to DSGI<br>• Integrated into IML – use if rest of program is IML | • Must use IML<br>• No native support for BY variables |
| DSGI | • For custom graphs<br>• Callable from DATA step, SCL, macro language | • No native support for BY variables |

## APPENDIX B: THE SETSQ MACRO

The following macro uses DSGI calls to find the current HSIZE and VSIZE settings, then uses the lesser of those values to reset the HSIZE= and VSIZE= options. The result is a square plotting area.

```
%macro setsq;

  /************************************/
  /* Macro to set plotting area square */
  /************************************/

  /*** Reset to defaults ***/
  GOPTIONS HSIZE= VSIZE=;

  DATA _NULL_;

    /*** Invoke DSGI ***/
    RC = GINIT();
    RC = GRAPH('CLEAR', '_DUMMY');

    /*** Get HSIZE and VSIZE ***/
    CALL GASK('HSIZE', HSIZE, RC);
    CALL GASK('VSIZE', VSIZE, RC);
    PSIZE = MIN(HSIZE, VSIZE);
    CALL SYMPUT('PSIZE', PUT(PSIZE,
   BEST12.));

    RC = GRAPH('UPDATE', 'NOSHOW');
    RC = GRAPH('DELETE', '_DUMMY');
    RC = GTERM();
  RUN;

  /*** Fix a square plotting area ***/
  GOPTIONS HSIZE=&psize VSIZE=&psize;

%mend setsq;
```

## REFERENCE

SAS Institute Inc., *SAS/GRAPH Software: Reference, Version 6, First Edition, Volume 1,* Cary, NC: SAS Institute Inc., 1990.

## ACKNOWLEDGMENTS

Thanks to Karen Crandall, Mary Maggio and Julie Strickland for their valuable suggestions for improving this paper.



## AUTHOR

Mail: Earl Westerlund
Eastman Kodak Company
1669 Lake Avenue, Mail Code 24608
Rochester, NY 14652-4608
Phone: 716-722-1983
Fax: 716-722-4415
Email: earlw@kodak.com