The DATA Step Graphics Interface
Jade Walker, SAS Institute Inc., Cary, NC

ABSTRACT
The DATA Step Graphics Interface, or DSGI, enables you to call the graphics subroutines used by SAS/GRAPH software during the DATA Step or in a screen control language (SCL) application. You access the graphics system by using function and subroutine calls. In the DATA step, you may have already used the SUBSTR function to collect a substring from within a string. DSGI uses function calls similar to the SUBSTR function to accomplish its goals. You can use DSGI to write a custom graphics application or prototype in conjunction with all the power of the programming statements available by the DATA Step. This paper describes the DATA Step Graphics Interface and the commands used within it. Some terminology of the Graphics Kernel System (GKS) is covered because DSGI is based on GKS. This paper also shows some examples using the DATA Step Graphics Interface and explains certain features and techniques used in the examples.

INTRODUCTION
Why would you use the DATA Step Graphics Interface? You may have or need a graphics application that is very specific to your work and that has not been addressed by SAS/GRAPH software. The functions and subroutines of the DATA Step Graphics Interface enable you to build a custom application or design a prototype for a graphics application. Since DSGI is contained within the DATA step, the application can be compiled and run in one step. The functions and subroutines of DSGI are also available in SCL with the same function and subroutine names and parameter lists. These two methods let you make the graphics application you want, as shown in Figure 1.

USING DSGI
Here is the usual order of logic flow when using DSGI:
1. Initialize the graphics environment.
2. Open a segment that collects the graphics primitives that you are going to issue.
3. Set attributes to change the graphical elements to effect a desired appearance.
4. Issue graphics primitives.
5. Close the segment and display it.
6. Terminate the graphics environment.

Initialize Graphics Environment
To initialize the graphics environment, use the GINIT function. This function reads the interface so that additional DSGI functions can be issued.

Open Segment
To open a segment, use the GRAPH function with the subcommand CLEAR. CLEAR is the only parameter that you need to specify to open a segment, but you can also add the name and description of the segment as additional parameters so that you can distinguish among graphs when running the GREPLAY procedure. The default name for a segment created by the DATA Step Graphics Interface is DSGI, and the default description is Graph from DATA Step Graphics Interface.

Set Attribute
To set an attribute, use the GSET function. An attribute is a user-controlled value that affects the appearance of graphical elements. Examples of this are text color, line width, and marker type. Figure 2 shows examples of setting the line type attribute and the line width attribute.

Figure 1 How DSGI Is Used
The GSET function has many subcommands, each referring to an attribute that you can change. While some subcommands control options within DSGI, others alter the appearance of graphical elements. For instance, if you want to change the line color to the color referred to by color index 3, use the GSET('LINCOLOR', 3) command. If you want to activate clipping around viewports, use the GSET('CLIP', ...) command.

**Issue Primitives**

To issue a graphics primitive, use the GDRAW function. The subcommands for the GDRAW function each refer to a shape that you can draw. Four types of shapes are defined in the GKS standard as shown in Figure 3: polylines, polymarkers, fillareas, and text.

**Close Segment**

To close a segment, use the GRAPH function with the subcommand UPDATE. There is an optional parameter to this command that allows you to control whether you want to see the graph at that time or whether you want to display it later in the session, perhaps during an invocation of PROC GREPLAY.<

**Terminate Graphics Environment**

To terminate the graphics environment, use the GTERM function. This function closes and displays any graphs that were not completed and then ends the interface.

**Query Graphics Environment**

There is an additional subroutine, GASK, that is used to query the values of attributes within DSGI. Suppose, for instance, that you want to set the line color to the color represented by index 3, but you don't know what the color name is because it is dependent upon the data fed into the application or because you have constructed a color-independent macro. You use the GASK('COLREP', ...) command to find out what color name is associated with index 3. GASK is a subroutine call rather than a function call because it may return many pieces of information to the DATA step. For example,

```main
CALL GASK('COLREP', x, y, rc);
```

returns three pieces of information: the character vector in the x direction, the character vector in the y direction, and the code returned from invoking the subroutine.

**GKS TERMINOLOGY**

**Operating Systems**

Because DSGI follows the GKS standard and follows GKS terminology, a few terms ought to be explained. One is the concept of operating states. GKS has specified phases at which only cer-
tain types of commands can be executed. The concept of operating states has also been adopted by DSGI. The operating states are defined as follows:

**GKCL (GKS closed)**
This is the initial state of GKS. No graphical resources have been allocated.

**GKOP (GKS open)**
The GKS attributes can be inquired.

**MOP (workstation open)**
In this implementation, the graphics catalog, either default or specified through the `GSET('CATALOG:...')` command, is opened or created.

**WSAC (workstation active)**
An active workstation can receive GRAPH commands.

**SGOP (segment open)**
Graphics output primitives can be generated.

For each command in the DSGI Interface documentation, each function lists the operating states during which that function may be invoked. If you use a command in the wrong operating state, the function returns a special return code and the SAS® System issues a message to that effect.

**Bundles**
Another way to handle attributes is through bundles. A bundle is a specified collection of attributes. It is referenced via a bundle index. DSGI has the following four types of bundles:

- **LINE** used in polyline, elliptical arc, and arc output primitives. The LINE bundle contains the attributes LINWIDTH (thickness of line), LINTYPE (style of line), and LINCOLOR (color index of line).
- **MARKER** used in the polymarker output primitive. The MARKER bundle contains the attributes MARSIZE (size of symbol), MARTYPE (type of symbol), and MARCOLOR (color of symbol).
- **FILL** used in fillarea, bar, pie, and ellipse output primitives. The FILL bundle contains the attributes FILTYPE (type of fill), FILSTYLE (style of pattern used when FILTYPE is PATTERN or HATCH), and FILCOLOR (color of fill interior).
- **TEXT** used in the text output primitive. The TEXT bundle contains the attributes TEXFONT (SAS/GRAPH font for the string) and TEXCOLOR (color of text).

**Aspect Source Flags**
For each attribute there is an aspect source flag (ASF). An aspect source flag indicates to DSGI whether to use an attribute in the current bundle or an attribute specified individually. For example, the command

```
BC = GSET('ASF', 'LINCOLOR', 'BUNDLED');
```
tells DSGI to get the line color attribute from the line bundle currently specified. The command

```
BC = GSET('ASF', 'TEXFONT', 'INDIVIDUAL');
```
tells DSGI to get the text font attribute from the text font currently set. Therefore, activating a bundle is a three-step process. First, define a bundle for a particular index. Second, set the aspect source flags for the attributes in that bundle to BUNDLED. Third, set the bundle index to the bundle that you have just defined.

**Text Operations**
Special terms used in drawing text are also used by DSGI. Two terms that may be unfamiliar are text path and text up vector. The text path of a text string indicates in which direction a text string "reads." In DSGI, the four values of text path are LEFT, RIGHT, UP, and DOWN. In both DSGI and the GKS standard, the default value is LEFT. Figure 5 shows examples of all four text paths that you can set.

![Figure 5: Example of Text Paths](image)

**Text Up Vector**
The text up vector of a text string is equivalent in function to the `ANGLE=` options in TITLES, FOOTNOTES, and NOTES. The vector, however, has two components, a base vector and an up vector. The arctangent of the ratio between the up vector and the base vector is equal to the angle at which the text string is drawn. The default value is (0,1), where 0 is the base vector and 1 is the up vector. This draws the text string with the baseline angle of 0, as shown in Figure 6.

![Figure 6: Examples of Text Up Vector](image)
Transformations

Transformations are part of DSGI. Transformations are composed of a viewport/window pair. A viewport is a definition of the boundaries in which drawing can occur. The default viewport of both SAS/GRAPH software and DSGI is the entire screen. The specification of the viewport boundary is made in terms of the viewport previously defined. This means if you want a viewport covering only the left half of the screen, you would define a viewport like this:

\[ \text{RC} = \text{GSET(} ' \text{VIEWPORT'}, 1, 0, 0, .5, 1); \]

A window is a definition of the coordinate system used in the current viewport. Every window is tied to a viewport so that if the viewport is not currently active, neither is its window. A window is defined in terms of the minimum and maximum values in both the X and Y direction. If you want only the X values from 20 to 50 and the Y values from 6 to 10 to be visible, define a window like this:

\[ \text{RC} = \text{GSET(} ' \text{WINDOW'}, 1, 20, 6, 50, 10); \]

EXAMPLES

Using the CALL GASK('TEXEXTENT'...) Routine

Suppose you want to draw an underline for a text string exactly the length of the text string. To determine the length of the text string, you need to set the font and height you are going to print the text string in, using the GSET('TEXFONT'...) and GSET(TEXHEIGHT...) commands. Then you use the CALL GASK('TEXEXTENT'...) routine, which returns information about the height and width of a text string. This is an example of a macro that draws a specified text string and then underline it:

\[
\text{%UNDERLINE}(x, y, \text{STRING}): \quad \text{x = GGROUND('TEXT', x2, y2, STRING)}; \\
\text{CALL GASK('TEXEXTENT', x2, y2, STRING, XEND, YEND, x1, x2, x3, y1, y2, y3);} \\
\text{RC = GSET('LIIMNIGHT');} \\
\text{RC = GDSRA('LINE', 2, x2, XEND, YEND, YEND);} \\
\text{RETURN UNERLINE(x, y, STRING);} \\
\]

This is an example of how to invoke the macro using DSGI:

\[
\text{DATA _MALL...}: \\
\text{RC = GROUND('TEXT', x2, y2, STRING);} \\
\text{CALL GASK('TEXEXTENT', x2, y2, STRING, XEND, YEND, x1, x2, x3, y1, y2, y3);} \\
\text{RC = GSET('LIIMNIGHT');} \\
\text{RC = GDSRA('LINE', 2, x2, XEND, YEND, YEND);} \\
\text{RETURN UNERLINE(x, y, STRING);} \\
\]

Figure 7 shows the picture resulting from the execution of the DATA step listed above.

Now underline this!

Figure 7 Using the %UNDERLINE Macro

The GASK('TEXEXTENT'...) routine returns a great deal of useful information about a text string. Not only does it give the starting and ending points of the baseline of the string, but it also returns the coordinates of the box that surrounds the text string, as shown in Figure 8. This can help you verify placement of text strings to avoid exceeding the boundaries of your device.

### Inserting Graphics Segments

Suppose you want to draw a picture of a person pointing out some feature on a map of the United States. To construct a map, you use the GMAP procedure, saving the output in a permanent graphics catalog. Next, bring up DSGI and set up a transformation to put the map output into so that it does not take up the entire screen. Insert the segment containing the U.S. map into the graphics segment currently open. Deactivate the transformation, and then draw the stick figure on the appropriate place on the graph.

One of the current limitations of DSGI is that the segment currently open and the segment to be inserted must be in the same graphics catalog.

If the segment is in a graphics catalog other than WORK.GSEG, use the GSET('CATALOG...' command to set the catalog to the one in which the graph output from PROC GMAP is located. The GSET('CATALOG...' command must appear before the GINIT function. To make the map take up only a small portion of the screen, create a transformation using the subcommands VIEWPORT and TRANSNO of the function GSET. The VIEWPORT and WINDOW subcommands only define viewports and windows. The GSET('TRANSNO...' command actually activates the specified transformation.

To include the graph of the map into a graphical segment produced by DSGI, use the GRAPH(INSERT...) command, supplying the name of the map of the graph as the other parameter. Turn off the transformation by using the GSET('TRANSNO', 0) command in order to access the entire device.

Then draw the stick figure instructor using the ARC and LINE subcommands of the GDSRA function. The pointer of the stick figure is drawn over the map because that line is drawn after the transformation using the GSET('TRANSNO', 0) command. Figure 9 shows the picture resulting from the execution of the DATA step listed above.

Figure 8 Values Returned from CALL GASK('TEXEXTENT'...)

Figure 9 Inserting a Graphics Segment
One thing to note about GDRAW commands that take a variable number of vertices, such as the LINE and FILL subcommands, is that the number of vertices comes first and then all the x and y values, followed by all the y values, instead of alternating x and y values. This adheres more closely to the GKS standard, which represents the vertices as a correspondence between an x array and a y array. If you supply a missing value for the number of vertices argument, DOOI calculates the number of elements in the x and y arrays based on the number of arguments passed by the GDRAW function.

Using the SAS Macro Facility with DSGI

You also can design custom graphs. An example is a graph showing airlines in business (see Figure 10). The plane symbol is generated by the SAS macro with the GDRAW function. The example code, the plane color (PCOLOR) and the trail color (TCOLOR) are set. Then the fill color is set in turn to the plane color to draw the plane and to the trail color to draw the trail by using the GSET('FILCOLOR') command.

Using GDRAW('TEXT') with Variables

The example of local housing starts, Figure 11, shows two methods of printing text from variables in SAS data sets. If the variable is a character variable, the variable itself can be used as a parameter to the GDRAW('TEXT') command. In this case, YEAR, a character variable, is printed as a list of values below the houses.

Using the GASK Subroutine

The next example, Figure 12, shows a clock that displays the time at which the graph was created. One of this program's features is that it uses the GASK('ASPECT') subroutine call to determine how to draw circles round. With this section of code, drawing of round circles becomes device independent:

To insert a hex string into a regular text string, use the concatenation operator to gather three text strings into one variable. Then use the variable as a parameter to the GDRAW('TEXT') command.

Local Housing Starts

Using the GASK Subroutine

The next example, Figure 12, shows a clock that displays the time at which the graph was created. One of this program's features is that it uses the GASK('ASPECT') subroutine call to determine how to draw circles round. With this section of code, drawing of round circles becomes device independent:

CALL GASK('ASPECT', ASPPECT);
ELSE DO;
XADJUST = 1:
YADJUST = 1 / ASPPECT;
END;
"SYSTEM"
END;
RC = GVM('TEXT', 50, 65, TITLE);
The last example is written in screen control language (SCL). A simple screen requiring entry of the graduate's name, course completed, and date on which the course was completed has been defined. Diploma graphics are sent to a graphics stream file to be printed later. GOPTIONS statements to initialize graphics output destination need to be submitted using the SUBMIT IMMEDIATE command. These should occur in the INIT section of the program.

INIT:
SUBMIT IMMEDIATE;
GOPTION OUT=PSL NOSHOW SUCCESS='SASdSGD-GRADS.OUT' AREA=60X60 DIPLOMA(insert), 'DIPLOMA';
ENDINIT;
RETURN;

All DSGI functions need to be contained within one section. You cannot do the GINIT function in the INIT section, the GTERM in the TERM section, and the rest of the functions in the MAIN section. This causes confusion in the sequencing of DSGI statements and results in messages about incorrect operating states.

The MAIN section uses an already created segment called DIPLOMA to contain the constant values of the diploma and any graphics decoration needed. It inserts the DIPLOMA segment and then completes the diploma by formatting the name, course, and date (see Figure 13).

This is to certify that Jade Walker has successfully completed the course Remedial Oboe on February 8, 1989.

This paper has given an overview of the DATA Step Graphics Interface. By using the functions and subroutines discussed, you can create graphic applications not yet covered by SAS/GRAPH software. You can integrate graphics with SCL applications. You can query the attributes and status of the DATA Step Graphics Interface to produce an application for which your users can supply default values for attributes. DSGI is not meant to replace the ANNOTATE facility; each has its strengths and weaknesses. For programmers accustomed to GKS, DSGI may prove easier to learn and use. The DATA Step Graphics Interface was created because of an entry listed in the SASware Ballot for a graphics toolkit. This paper has shown how DSGI can be used to fill that need.

SAS, SAS/GRAPH, and SASware Ballot are registered trademarks of SAS Institute Inc., Cary, NC, USA.