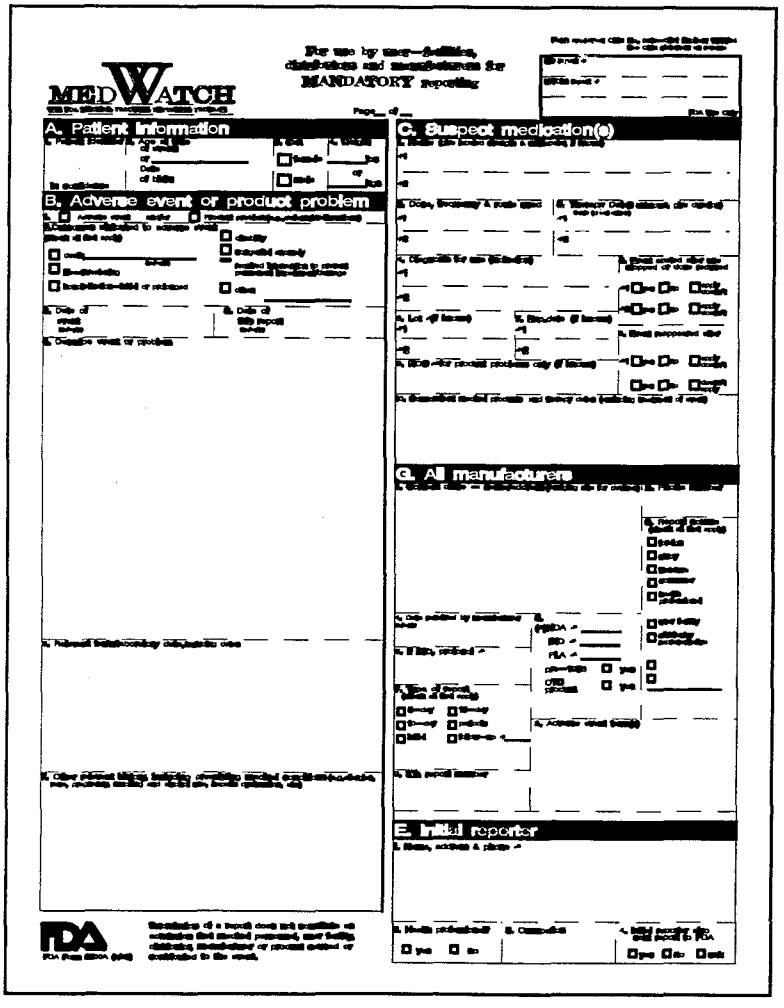
Creating Forms Using the Data Step Graphics Interface (DSGI)

Harvey Monder, Sandoz Pharmaceuticals Corporation

The display of data on standard forms is important in the meaningful transmission of information. Organizations depend on these forms to rapidly signal their purpose and content to those responsible for their review. However, it can be burdensome to the reporting agency to record data on these forms. Some of this burden can be alleviated by the automatic retrieval of data to printers that can complete pre-existing forms. There is a danger of loss of alignment of the form in the printer so that values may be misplaced. This would not be the case if the form and its contents were created atthe same time.

Many forms are made up of a variety of elements that cannot be combined easily using line-printer capabilities. Various sizes and styles of type face may be used, reverse colors and company logos as well as lines in different locations and orientations can exist on one form. The reproduction of these forms can be accomplished by the use of graphics, producing desktop publishing quality output. With the introduction of the Data Step Graphics Interface (DSGI) in the SAS® system this is now possible. Since the graphics elements are created as part of the data step, data to complete the form can be incorporated as graphic elements. Thus it is possible to place text anywhere on the form as integrated graphic text.



The FDA Adverse Event Reporting Form is an example of a graphically complex form that can be generated using DSGI. It uses a variety of text fonts and sizes, reverse color printing, and straight and curved lines in various locations on the form. This paper will describe an approach to the creation of a program to generate this form using DSGI. It should be noted that this program has not been used for any submission to the FDA and no effort has een made to determine if it would be acceptible to the agency. The form was programmed primarily as an example of a complex form that could be created as a DSGI graphics object.

Defining the Page

One of the powerful features of DSGI is the ability to define the scale of the screen. In this particular instance the screen should be defined as the same measure as the original form. The original form measures 205mm by 270mm. This is set by use of the 'WINDOW' function. In DSGI one defines a 'VIEWPORT' which determines the area of the screen that the graphic will occupy. The 'WINDOW' sets the scale within the VIEWPORT:

```
rc=gset('VIEWPORT',1,0,0,1,1);
rc=gset('WINDOW',1,0,0,205,270);
rc=gset('TRANSNO',1);
```

The first numeric parameter defines the

numeric ID of the VIEWPORT and WINDOW. The VIEWPORT can have horizontal and vertical ranges from 0 to 1, so that the screen is filled from the lower left corner of (0,0) to the upper right corner of (1,1). Thus this graphic will occupy the entire display. The default size of the WINDOW is 100 units in each direction. However, for our purposes, the WINDOW will be defined with the horizontal scale as 205 units and the vertical scale as 270 units. This corresponds to the metric size of the form. The lower left is (0,0) and the upper right is (205,270). the TRANSNO command tells DSGI which VIEWPORT/WINDOW combination to use. The advantage to the use of this approach becomes clear when the form is laid out. For example, to draw the lines on the form that separate the various sections, measure the location of each line and its length using a metric ruler then entered those values as line coordinates. So that a line that started at the left edge of the form and was 100 mm long, located 55 mm from the bottom of the form is defined as:

```
rc=gdraw('LINE',.,0,100,55,55);
```

A peculiarity of the syntax of this command is that all the 'X' values are listed first, followed by all the 'Y' values. The missing value of the second parameter tells DSGI to determine how many line segments there are by counting the number of X-Y pairs.

Two colors are defined for use in this form, Black is defined as color '1' and White is color '2':

```
rc=gset('COLREP',1,'BLACK');
rc=gset('COLREP',2,'WHITE');
```

These color numbers are used to assign color values to text, lines, backgrounds, etc.

Many of the environmental graphics options can be set either by use of a GOPTIONS statement or by using DSGI functions. The DSGI functions will override any GOPTIONS that have been set. In generating the Adverse Events form, most of the text uses the SWISS font. Since there are changes in font in different parts of the form it is more efficient to change fonts and font characteristics using DSGI.

```
rc=gset('TEXFONT','SWISS');
```

In order to use the lower left corner of the text string to set its page location the alignment of the text is set by:

```
rc=gset('TEXALIGN','LEFT','BASE');
```

Each section heading on the form is printed in reversed color - white text on a black background. In order to create this a black bar was drawn then white text was entered so that the text was drawn on top of the bar. You have a choice of using either 'BAR' or 'FILL'. Both will provide the black background for this entry. BAR draws rectangles by specifying the lower left and upper right corners. FILL is used to draw polygons. For this example I used FILL. Items are generated in the sequence in which they are encountered in the program. If the text is drawn first it is covered by the bar and is not visible. So it is important to remember to draw the background first, then insert the text. The 'Patient Information' header was generated by:

```
/* set the color and fill type */
rc=gset('FILCOLOR',1);
rc=get('FILTYPE','SOLID');

/* set the text color and size */
rc=gset('TEXCOLOR',2);
rc=gset('TEXHEIGHT',4);


/* draw the black bar - note that all
   the 'X' values precede all the 'Y'
   values . The syntax for FILL is
  the same as for LINE. The
  second parameter is missing
  signalling that the number of
 X-Y pairs are used to determine
 the number of line segments*/

rc=gdraw('FILL',.,0,100,100,0,0,
        245,245,240,240,245);

/* Now enter the text */
rc=gdraw('TEXT',0,241,
        'A. Patient Information');
```

Check boxes were generated by creating 2.5 mm squares. Since a number of such boxes were needed a macro was used to facilitate coding.

Some creativity was needed in recreating the logo and specialized text. For example, existing text fonts in the font library do not include a font to match the 'W' in 'MEDWATCH'. This line of text uses the CENTX font. The 'W' was created by overlapping two 'V's:

```
rc=gset('TEXFONT','CENTX');
rc=gset('TEXHEIGHT',6);
rc=gdraw('TEXT',1,250,'MED');
rc=gdraw('TEXT',32,250,'ATCH');
```

```
rc=gset('TEXHEIGHT',15);
rc=gdraw('TEXT',16,250,'V');
rc=gdraw('TEXT',21,250,'V');
```

The FDA logo at the bottom left was reproduced by combining lines and arcs:

```
rc=gset('LINWIDTH',5);
rc=gdraw('LINE',.,1,1,8,4,11,11);
rc=gdraw('ARC',8,7.5,3.5,270,90);
rc=gdraw('LINE',.,8,5,5,4,4,9);
rc=gdraw('LINE',.,2,2,4,4,8,8);
rc=gdraw('LINE',.,4,2,2,8,9,9,10,10);
rc=gdraw('ARC',8,7.5,2.5,270,90);
rc=gdraw('LINE',.,8,6,6,5,5,9);
rc=gdraw('LINE',.,11.5,14,15,19,
        19,14,7.5,11,11,5,4,4);
rc=gdraw('LINE',.,10.5,14.5,
        17.5,13,4,10,5,5);
```

ARC draws an arc of a circle. If you have the need or, in this case, if you feel adventurous, you can draw a segment of an ellipse using 'ELLARC'.

Entering the Data

Data is entered into a field on a form only as character values. The TEXT parameter is the way data is normally entered into a field. Thus any such data must be converted using the PUT function. The alignment of the text is such that the lower left corner of the text string can be used to locate the value on the form. Numeric data might be entered as:

```
rc=gdraw('TEXT',105,136,
        put(lotno,3.));
```

The mode of data entry, of course, depends on the structure of your data set. If there is one observation per form, then the code for the data entry can be intermixed with the code for the form generation. However, this would limit the utility of the program. Placing the form code in a labeled section allows the code to be called conditionally. The data entry program lines are in a separate section of the program. DSGI is started with the first observation:

```
if _n_=1 then rc=ginit();
```

This loads the DSGI graphics library. To generate the form with each new Adverse Event:

```
if first.AETEXT then link FORM;
```

The data entry lines follow this:

```
rc=gset('TEXT','SWISS');
rc=gset('TEXHEIGHT',1.3);
rc=gdraw('TEXT',...
```

Text strings that are too long for the width of the entry field must be split programmatically. Since the Y origin is at the bottom of the form for each line of text generated Y must be decremented. For example:

```
stext=substr(AETEXT,1,50);
do while(length(stext) gt 1);

   /* evaluate each character for */
  /* first blank from end of string */
 ind=substr(stext,length(stext),1);
 c=0;
 do until(ind eq ' ');
       c+1;
       ind=substr(stext,
          length(stext)-c,1);
 end;

if c=50 then c=1;
rc=gdraw('TEXT',c,y,stext);

y+(-4); /* move down 4 units */
AETEXT=substr(AETEXT,
    length(stext)+2,50);
stext=substr(AETEXT,1,50);
end;
```

Conclusions

DSGI offers a solution for the printing of completed forms. Using this technique, SAS data sets are accessed and printed in the appropriate form without the need for third party software. As data becomes available forms can be created. This system can be automated and removes the necessity to maintain a stock of blank forms or the use of other software packages to create the forms.

The programming of forms using DSGI is also facilitated by the ability to define the 'drawing surface' with the same dimensions as those used by the original form. In this particular case, the Adverse Events Reporting Forms was measured in millimeters and those measurements were translated to the screen. Lines and text could be located easily by use of a ruler to determine their start and stop coordinates.

In examining the program that generates the form, the number of lines of code are initially daunting. However, remember that graphics can address every part of the screen and unlimited

numbers of shapes can be defined. The requirement of preciseness of location increases the number of parameters that are needed to locate an object. Thus a line is drawn from a particular set of coordinates to another set. Polygons can have many sides each of which must be defined. Text must be located precisely and the characteristics of the text must be defined (i.e., font, size, color, etc.). A careful examination of the code will show that it is quite basic. Each line of code defines an object to the display or sets the characteristics of an object. If a command starts with 'gset' it sets a characteristic, much the same way that the GOPTIONS statement sets the graphics environment. The actual objects are drawn by use of 'gdraw' commands.

Much more complex graphics can be created using DSGI. It is possible to incorporate other graphics, such as those created by the various graphics procedures. These can be incorporated within 'viewports' that can be sized and located anywhere on the output[1].

With DSGI, submission ready forms and publication quality graphics can be created without leaving the SAS system or use of third party software. Data can be directly accessed and output to forms as needed. There is a large dictionary of commands that are available providing for great flexibility in the design of graphics[2]. In addition, those familiar with the GKS standards would have little difficulty adapting to the DSGI command structure, as a modified GKS standard is followed[3].

Bibliography

[1] Monder, H. Creating Tables Using the Data Step Graphics Interface (DSGI), Observations, Third Qtr., No. 4, Vol. 4, 1995.

[2] SAS/GRAPH® Software Reference, Vol. 1, Version 6, First Edition, 1990.

[3] Enderle, G., Kansy, K., and Pfaff, G. Computer Graphics Programming, Berlin: Springer-Verlag, 1984.

Harvey Monder, Ph.D.
Building 419/1180
Sandoz Pharmaceuticals Corporation
59 Route 10
East Hanover, New Jersey 07936-1080
Phone: 201-503-5822
E-Mail: harvey.monder@sandoz.com