ABSTRACT
SAS has long been valued for its strength in statistical computing. Indeed, statistical analysis is the basis for the creation of SAS over 25 years ago. Today, in the clinical research industry, SAS is almost always used in conjunction with a separate word processing package in order to present output in a richly formatted manner that will be pleasing to the eye of the client. However, SAS itself can produce the same aesthetically pleasing output by utilizing the Data Step Graphics Interface. The examples shown in this paper are based on SAS version 6.12 under Windows 98, but the principles presented apply to all operating systems and subsequent releases of SAS.

WHY USE DATASETP GRAPHICS?
With the advent of Desktop Publishing, it is becoming more and more accepted and expected to have customized client documents that look professionally done. The SAS way of producing output via the Proc Print procedure and/or data _null_ and “put” statements just do not measure up in the world of publishing software. This is true even in the Pharmaceutical Industry where the numbers are our primary concern. DSGI gives SAS the boost it needs to present more professional looking output.

WHAT EXACTLY IS DATASETP GRAPHICS?
The Datastep Graphics Interface (DSGI) enables you to create graphics output in a base SAS datastep or from SCL code. The power of SAS/GRAPH can be leveraged by calling graphics routines. Combine these with the power of the datastep and you have a custom graphics application. When using the term “graphics”, it can refer to actual SAS generated graphs or graphic objects of your making. A very experienced person can actually produce company logos using this tool.

GETTING STARTED

VISUALIZE
The first step is to visualize what you want your output to look like, even making a sketch of it. The actual graphing part comes in here because you need to visualize it as coordinates on a graph. Use an X and Y axis with any range of coordinates. The only limitations there are depend upon the dimensions of your devices windowing system. You are generally guaranteed a range of at least 0 through 100 on either axis. You can find out what your system default is with the GASK routine;

```
call GASK('WINDOW',1,llx,lly,urx,ury,rc);
```

Doing the above call in a data step would result in a dataset containing coordinates for the lower left x-axis, lower left y-axis, upper right x-axis and upper right y-axis, as well as a return code for the call procedure. If you want to use coordinates other than the system default, you can set them up by using the GSET command

```
rc=GSET('window',1,0,0,100,100);
```

The previous command would give you a window with both the x and y axis having coordinates from 0 to 100. Experimentation is the best way to find the best coordinates for your custom graphics.

SETTING UP THE BASICS
There are a few basic steps that need to be taken in order to ensure correct functioning of DSGI. They are as follows:

1. Initialize DSGI
2. Open a graphics segment
3. Generate graphics elements
4. Close the graphics segment
5. End DSGI

The following source code presents these steps in a very simple example:

```
data _null_;  /* Initialize DSGI */  rc=ginit();  /* Open graphics segment */  rc=graph('clear');  /* Generate graphics */  rc=gdraw('text', 40, 93, 'Simple Graphics Output');  rc=gdraw('line', 2, 0, 100, 0, 100);  rc=gdraw('line', 2, 0, 100, 100, 0);  /* Close graphics segment */  rc=graph('update');  /* End DSGI */  rc=gterm();  run;
```

The above writes some text and draws two intersecting lines using all default values. The resulting graphics output will look like this:

CUSTOMIZE
The simple code above can be customized in several ways. Note the changes in the graphic when just a few lines are added:

```
data _null_;  /* Initialize DSGI */  rc=ginit();  /* Open graphics segment */  rc=graph('clear');  /* Generate graphics */  rc=gdraw('text', 40, 93, 'Simple Graphics Output');  rc=gdraw('line', 2, 0, 100, 0, 100);  rc=gdraw('line', 2, 0, 100, 100, 0);  /* Close graphics segment */  rc=graph('update');  /* End DSGI */  rc=gterm();  run;
```

The above writes some text text and draws two intersecting lines using all default values. The resulting graphics output will look like this:
/* Initialize DSGI */
rc=ginit();
rc=gset('window',1,0,0,100,100);
rc=gset('transno',1);

/* Open graphics segment */
rc=graph('clear', 'Mygraph');

/* Generate graphics */
rc=gset('texfont','CENTB');
rc=gset('transno',1);
rc=gset('window',1,0,0,100,100);
rc=ginit();
rc=gdraw('text', 30, 93, 'Customized Graphics Output');

/* Close graphics segment */
rc=graph('update');
/* End DSGI */
rc=gterm();

This is what your output will look like after adding the following enhancements:
Define a window size of 0 to 100
Name the graph (Mygraph)
Select a more attractive text font to use (CENTB)
Specify a text height (4.5)
Make the intersecting lines wider

There are many more customization techniques that can be utilized using DSGI.

PRACTICAL APPLICATION
How can this apply to our jobs in the pharmaceutical industry? I can speak to the usefulness of DSGI in a Contact Research Organization. It is inevitable for a CRO working with many different clients to find that there will always be at least one client who will not be pleased with standardized output. They will want custom output to reflect the nature of their company. Take, for example, the Data Clarification (or Query) form.

The CRO has a standard query form generated by using “put” statements in a “data _null_” block. The client wants something better looking. Here is an example of what could be done using DSGI.

goptions reset=all display device=win
rotate=landscape;
libname querylib 'C:\DSGI\Queries';
data _null_;
/* Terminate DSGI */
rc=gterm();
run;

This code will produce the following customized query template:

The final step will be to add a dataset containing the actual data you want printed in the template. Leave the template outline in place and add a set statement for the dataset and place the actual data values where you want them on the template:

goptions reset=all display device=win rotate=landscape;
libname querylib 'C:\DSGI\Queries';
data _null_;
set querylib.qdata end=lastobs;
  if _n_=1 then do;
    /* Define a catalog to store graphic segments in */
    gset('catalog','querylib','gqueries');
  end;
/* Initialize DSGI */
rc=ginit();
end;

/* Define a window to work in */
rc=gset('window',1.0,0.1,100,100);
rc=gset('transno',1);
rc=graph('clear','Mygraph');

/* Draw border around page */
rc=gset('linwidth',3);
rc=gdraw('line',2,1,99,87,87);
rc=gdraw('text',4,85,'Pt');
rc=gdraw('text',8,85,'Pt');
rc=gdraw('text',13,85,'Site');
rc=gdraw('text',19,85,'Query');
rc=gdraw('line',2,1,99,82,82);

/* Write column headers */
rc=gset('texfont','CENTX');
rc=gset('texheight',1.8);
rc=gset('texalign','LEFT','BASE');
rc=gdraw('text',4,83.4,'#');
rc=gdraw('text',8,83.4,'Init');
rc=gdraw('text',13,83.4,'#');
rc=gdraw('text',19,83.4,'#');
rc=gdraw('text',26,83.4,'Issue');
rc=gdraw('text',61,83.4,'Resolution');
rc=gdraw('line',2,1,99,82,82);

/* Write out data values */
rc=gdraw('text',4,80,id);
rc=gdraw('text',8,80,init);
rc=gdraw('text',13,80,site);
rc=gdraw('text',19,80,qrynum);
rc=gdraw('text',26,80,issue1);
rc=gdraw('text',61,80,resolut1);
rc=gdraw('text',26,78,issue2);
rc=gdraw('text',61,78,resolut2);
rc=gdraw('text',26,76,issue3);
rc=gdraw('text',61,76,resolut3);
rc=gdraw('line',2,25,25,15,82);
rc=gdraw('line',2,60,60,15,82);

/* Draw bottom Internal Use box */
rc=gdraw('line',2,1,99,13,13);
rc=gdraw('text',40,13.7,'For Internal Use Only');
rc=gdraw('line',2,1,99,15,15);

/* Close the graphics segment */
rc=graph('update');

/* Terminate DSGI */
rc=gterm();
run;

The resulting graphics output will look like this:
With less than 100 lines of code, we can generate custom output for our clients.

**AREAS OF CONCERN**
As with any shift to a different technology, there are a few areas that may be cause for concern.

1. **Editing Ability** - Although SAS/GRAPH contains a graphics segment editor, it is not easy to use. Especially so for non-technical SAS users and for graphics that contain a lot of text. The most efficient way to modify is regeneration.

2. **Printing** - Since DSGI produces graphic objects, it takes them longer to print. Even text is considered a graphic. Depending upon the size of your study, printing a batch of queries could be tediously slow.

**CONCLUSION**
To meet the need for a fast and professional looking customized client document, I would highly recommend the SAS Dataspace Graphics Interface. It is not hard to learn to use and is very flexible regarding the output that can be produced by using it.

**ACKNOWLEDGMENTS**
Special thanks to my very knowledgeable friend and former colleague at SAS Consulting Services, Dave Berger, who introduced me to DSGI.

**CONTACT INFORMATION**
Your comments and questions are valued and encouraged. Contact the author at:
Elaine Dempsey  
Rho Inc.  
100 Eastowne Drive  
Chapel Hill, NC 27514  
Work Phone: (919)408-8000  
Fax: (919)408-0999  
Email: edempsey@rhoworld.com  
Web: www.rhoworld.com