Paper 112-2007

An Introduction to SAS/GRAPH® Step-by-Step

Deb Cassidy

ABSTRACT

SAS/GRAPH® is an extremely powerful tool. In addition to producing common graphs like pie charts and line graphs, it has the power to create any custom graph that you could imagine. Another great benefit to SAS/GRAPH is its ability to programmatically create your graphs without adjustments. You may have a few daily or monthly graphs to create, hundreds or thousands of graphs by patient or customer or product, or on-demand graphs where someone specifies the subset criteria to get a standardized graph.

However, before you can harness the real power, you need to know the basics. This workshop will walk you through the creation of a graph and exploring some of the many ways you can make a "picture worth a thousand words." You'll start simple using all the defaults. Then we'll explore some of the many options that can enhance a graph. This will include some of the recent enhancements to SAS/GRAPH such as the availability of ODS styles to have a consistent look between your graphs and reports. We'll even save the graph in a non-SAS file such as an RTF or PDF. Since this is a hands-on workshop, you'll get to pick from several choices so you easily tell what changes in the graph when you specify an option. This workshop assumes you have no SAS/GRAPH experience. It assumes you do have a basic understanding of using SAS procedures and statements.

WHY USE SAS/GRAPH

The abstract provides some of the reasons to use SAS/GRAPH. If you just need a few, occasional graphs, you might want to consider GRAPH-N-GO in SAS. This choice does not provide code. Enterprise Guide and SAS/ASSIST provides code and have some types of graphs and options. There are also many non-SAS options.

NOTE ON CODE EXAMPLES

All code assumes you are submitting it interactively. You may have to add options when you are in batch mode.

GETTING STARTED

SAS/Graph runs on all platforms and can be displayed on many devices and in many formats. That's both good news and bad news. It's good because you aren't limited to a pc. It's bad because not all platforms/devices/formats are able to handle the same options so defaults may differ. If you're in a situation where you'll need code to work on multiple platforms/devices/formats, you may find yourself having to customize code. An obvious example is wanting to print graphs on a mainframe printer than only prints black and white as well as outputting the same graphs to a PDF that can handle color. Some of you may never need to be concerned with the following section but it will be essential for others. If you run a simple example graph and don't get what you expect, you'll

probably want to determine whether you need to specify a device and look into the defaults for the device.

The following code will provide you a list of all the devices available:

```
proc gdevice;
run;
```

The results will appear in a new window. Notice that some devices have different names for black/white versus color. PDF and PDFC is one such example. You can put a "B" or "S" beside a device to see some of the default values for options. To find all the defaults for a specific device, you can submit the following code:

```
proc gdevice nofs;
 list device_name;
run;
```

Because the proc is interactive, the nofs option will let the results write to the output window.

There are many options. Most people will be concerned with options such as hsize/vsize and the list of default colors. The following code will show you all the options available.

```
proc goptions;
run;
```

If you want to change an option, you can submit a graphic option statement. It's just like submitting an options statement in base SAS except you need to use GOPTIONS. Forgetting the "G" is a very common mistake – and people often think they've specified the option incorrectly based on the error message in the log. The following code will reset the horizontal size to 10. Ten what? Inches? Centimeters? You can specify the unit. If you don't, it will default to inches.

```
goptions hsize=10;
```

What if you really wanted 10 centimeters? Just use:

```
goptions hsize=10cm;
```

Options will stay set until you reset them. This can cause confusion when you are creating more than one graph in the same program and want different options. You can check specific options with a proc as well.

```
proc goptions option=option-to-check;
run;
```

This will give you the specific option. Some additional related options might be listed as well. If you wanted to check the horizontal size after setting it to 10cm, you would get the following in the log:

```
HSIZE=3.9370 in. VSIZE=7.5833 in.
```

Even though you specified 10 cm, you'll notice the result is shown in inches. In addition to specific options, you can specify TITLE, FOOTNOTE, PATTERN, SYMBOL, AXIS and LEGEND. The first two can help in base SAS as well. You can check titles with:

```
proc goptions TITLE;
run;
```

Since where you put your titles in relation to your proc and run statements can affect the results, you'll want to use this carefully.

The reset all options back to the default, you can use:

```
goptions reset=all;
```

You can also reset select options as in

```
goptions reset=title;
```

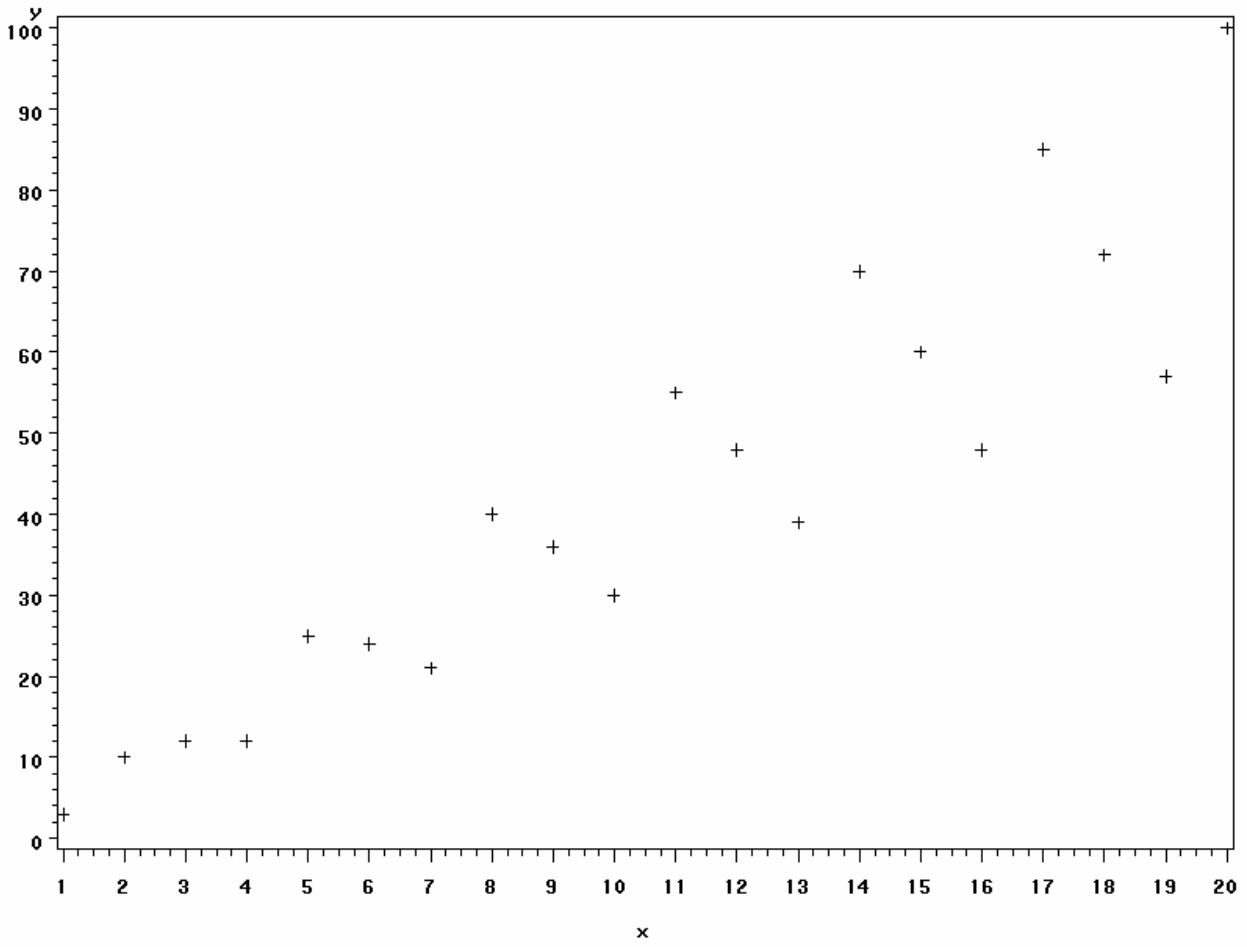
If you want to reset several options but not all the options, you will need to use separate goptions statements as in;

```
goptions reset=title;
goptions reset=hsize;
```

CREATING YOUR FIRST GRAPH

SAS/GRAPH offers many kinds of graphs including maps and your own custom graphs. This workshop used plots for simplicity.

```
*** Create some data ;
data mydata;
do x=1 to 20;
     if mod(x,3)=0 then  y=4*x;
else if mod(x,3)=1 then y=3*x;
else if mod(x,3)=2 then y=5*x;
 output;
end;
run;
*** plot data - y is the vertical axis (up and down);
***      x is the horizonal axis (goes across);
*** Remember to put the G in the name of the proc;
proc gplot data=mydata;
 plot y*x;
 run;
```



The basic graph isn't very impressive. There are many options that can be set. This is another advantage and disadvantage of SAS/GRAPH. You need options to improve the look of the graph but you don't want to use options that will be misleading given your data. You might want to change the type of graph as well. GPLOT was selected for ease of use in creating this workshop presentation but other graph types such as GCHART may be better for your data. You may also find summarizing your data in a different graph conveys the message best. There have been numerous SAS conference presentations as well as books on the topic that you should review before designing your graph. The rest of the presentation assumes you have the proper design for your graph.

SYMBOL Statements

The SYMBOL statement tell SAS how to display the values. This includes the size, color and shape to display. For plots, it also includes how to connect the dots. For example, the following will connect the data points with straight lines. The I stands for interpolation.

```
symbol i=JOIN;
proc gplot data=mydata;
 plot y*x;
 run;
```

Some of the other methods include NEEDLE, STEP, BOX, SPLINE. There are also several regression options.

You can change the shape shown for the data point. SAS/GRAPH can print many different shapes. Many shapes are not on a keyboard so there needs to be an alternative way to specify such as shape. You might think the following code would display a plus sign as the shape. Actually, it displays a plus sign with a circle around it. You can display plus sign by using the word "PLUS". All the many shapes available can be found in HELP.

```
 symbol i=join value=+;
 proc gplot data=mydata;
 plot y*x;
 run;
```

You can also control the line. Line can have values from 1 to 46. A value of 1 will display a solid line. Other values will display the line with varying widths of dashes, dots and blanks. Once again, you'll want to check HELP to see the wide range of choices. When you have more than one line, you'll want to ensure you pick line types that are easily distinguished from each. If the graph may end up copied and reduced, this will be become more important.

```
symbol i=join  value=plus width=5 line=45 ;
proc gplot data=mydata;
 plot y*x;
 run;
```

If you run the above code, you'll notice the data points get lost when you make the line wider. You'll want to change the size of the data point shape as well. This value can be a decimal and you can also specify the units. By default, the height will be in cells and the size of the cell depends upon the your values for several other options.

```
 symbol i=join value=plus width=4 line=45 height=3.43 cm;
proc gplot data=mydata;
 plot y*x;
 run;
```

CREATING A MORE COMPLEX GRAPH

You'll often want to have separate lines for another variable. The following will create separate lines for the different values of the third variable. In this case, there are 2 values of Z so 2 symbol statements were used.

```
*** Create some data with a third variable;
data mydata_3var;
do x=1 to 20;
if mod(x,2)=0 then y=4*x;
else y=3*x;
 z=mod(y,2);
  output;
 end;
run;

symbol1 i=join  value=PLUS width=1 line=1 height=1;
symbol2 i=join  value=STAR width=3 line=45 height=4;
proc gplot data=mydata_3var;
plot y*x=z;
run;
```

However, if you run the code as shown, you might not get what you expect. You probably thought you would get one narrow line with a small plus symbol for each data point and one wider line with a much larger star shape. When you run the code on most devices, you'll discover you get identical lines except for the color. The only time you get the expected lines is when the device is black and white only. Why? If you do not specify a color in a symbol statement, SAS will cycle through all the colors in the color list before going to the next symbol statement. The same is true for pattern statements used by some other procs. In the case of a black and white device, there is only one color so the symbol statements work as expected. Devices that are actually blank, white and shades of gray will treat each shade of gray as a color. In most cases, it is safer to always specify the color as part of the symbol statement. If you specify a color that is not available on your device, another color will be substituted. This will be noted in the log. It's one of the many reasons to always check your log!

```
symbol1 i=join  value=PLUS width=1 line=1 height=1 c=magenta;
symbol2 i=join  value=STAR width=3 line=45 height=4 c=green;
```

IMPROVING YOUR LABELS

Variable names are usually not sufficient. If you have labels in your dataset, they will be used. You can set a label or override the label in the dataset by specifying it with the proc.

```
proc gplot data=mydata_3var;
 plot y*x=z;
 label x='This is x'
       y='This is a long label for y to show what happens'
       z='This is a short Z label';
 run;
```

The label for Y was picked to show how the graph shrinks to accommodate the label. In most cases, you'll want to angle the label so it fits better. The following code will turn the label on its side.

```
axis1 label=(a=90);
proc gplot data=mydata_3var;
 plot y*x=z / vaxis=axis1;
 label x='This is x'
       y='This is a long label for y to show what happens'
       z='This is a short Z label';
 run;
```

The value for A can be anything from –360 to 360 although 90 is probably used most often. Unlike the symbol statement, just adding an axis statement will not make it be used with the graph. You need to specify an option on the plot statement. Each graph has at least 2 axes so you need to associate the axis statement with the appropriate axis.

There are many options for the axis as well. The following code would eliminate the little tick marks and only display large tick marks at every 4th value. In addition to angling the vertical axis label, it would change the font to Brush and make it .35 cells high. The only thing changed for the horizontal axis is to eliminate the little tick marks. Notice that the axis statements do not have to be in numerical order. You can have up to 99 axis statements.

```
axis1 label=(a=90 f=Brush  h=.35) minor=none
      order=(0 to 80 by 4);
axis7 minor=none;

proc gplot data=mydata_3var;
 plot y*x=z / vaxis=axis1 haxis=axis7;
 label x='This is x'
       y='This is a long label for y to show what happens'
       z='This is a short Z label';
 run;
```

CHANGING LEGENDS

The legend shows the value of your third variable represented by each of the individual lines. Once again, there are many options that you can change. The following code puts the legend at the top of the graph and inside the graph border. It also places the values on top of each other instead of beside each other. It removes the variable label. A colored border is added as well. The best placement of the legend really depends on the actual values in the plot. Placing the legend inside the graph will allow a little more room for the graph but you must be careful not to overlay a part of the graph. Legend statements are like axis statements in that you must specify which one to use with the specific graph.

```
symbol1 i=join  value=PLUS width=3 line=6 height=1.7 c=yellow ;
symbol2 i=join  value=a width=2 line=40 height=2.4 c=blue ;
axis1 label=(a=90 f=Brush  h=.35) minor=none
      order=(0 to 80 by 4);

legend1 down=2 label=none position=(INSIDE TOP) cborder=PURPLE;
```

```
proc gplot data=mydata_3var;
 plot y*x=z / legend=legend1 vaxis=axis1 haxis=axis7 ;
 label x='This is x'
       y='This is a long label for y to show what happens'
       z='This is a short Z label';
 run;
```

UNEXPECTED RESULTS

Once you've gotten the idea of specifying options, you might want to try your own data. Let's assume this was your data. If you run the graph as shown, you might be a bit surprised at the results. You don't get the nice simple lines as you did in the previous example. Why? The problem isn't with the graph statement. The PROC SORT needs changed. Your data must be sorted correctly before it is used in the graph proc. This is probably even more important the more variables you use in the graph. Results such as this graph may also indicate that you should summarize your data or use a different type of graph. I am not listing the "correct" sort here because the "correct" sort really depends upon your data. There are cases where the odd-looking results are really what you want to display!

```
data mixed_up;
    do x=1 to 10;
       do z ='a','b';
         do y=1 to 10 by 2;
          output;
          end;
          end;
          end;
          run;

 proc sort data=mixed_up;
     by z;
     run;

proc gplot data=mixed_up;
 plot y*x;
run;
```

CREATING MULTIPLE GRAPHS IN ONE PROC

The previous examples have been very simple. The following shows some of the real power for using SAS/GRAPH. This data contains a "group" variable so you can get one graph per group. Notice the data are sorted by that variable. When you run this code, you'll end up with 100 graphs.

This example has also become more complex. The symbol statements show different colors for the line and the data point shape. Title statements are specified and show how different fonts can be used in the same title. It also shows what happens if you specify a variable label in both the proc and axis statements.

The other important thing to notice about this code is the use of the “NAME” option. By default, the name of a graph will be based on the proc. You can provide a different name. If the name is already used, SAS will add a number. If the name of the graph becomes longer than 8 characters, it will start truncating the name. Each of the 100 graphs in this example will have its own name. A later example will show why the name of the graph is important.

```
*** create some data ;
data mydata_bygroup;
do group=1 to 100;
   do x=1 to 20;
   if mod(x,2)=0 then y=4*x;
else y=3*x;
 z=mod(y,2);
  output;
 end;
 end;
run;

proc sort data=mydata_bygroup;
  by group x y z;
run;

symbol1 i=join v=star c=blue l=1 h=3 cv=orange;
symbol2 i=spline v=square c=red l=43 h=2 cv=green;

axis1 minor=none label=('This is from axis1');
axis2 minor=none label=(a = 90 'This is from axis2');

proc gplot data=mydata_bygroup;
 by group;
 plot y*x=z / haxis=axis1 vaxis=axis2 name='Group';
 label group='Grouping Label'
       z="This is Z label"
       x="This is X label";
title f=simplex "This is the " f=triplex "first title "
      f=math "line";
title2 f=script "This is the second title in SCRIPT";
run;
quit;
```

SENDING GRAPHS TO OTHER DESTINATIONS

When you are working interactively, you can export you graphs. You can also do this programmatically. The following code saves thegraph as a GIF file. The device specifies the format of the graph. The filename statement shows where to place the graph. The GSFNAME option associates the graph with the filename.

```
filename grafout 'mygraph.gif';
goptions reset=all
         device=gif
         gsfname=grafout;
```

```
proc gplot data=mydata;
 plot y*x;
 run;
```

What happens when you have more than one graph created in a proc? It actually depends on the device you've specified. If you specify a device such as GIF, you will just get the last graph. The following example will create a folder with individual files with each graph. Other devices will create one file with all the graphs. Which one you use depends upon what you want to do.

Even if a device can have multiple graphs, your graphs will be overwritten unless you use the GSFMODE=APPEND option as shown below. The ROTATE option is also used here because the graphs look better as landscape with this device.

You might also notice I put "quit" after the run. Why both? The RUN tells SAS to create the graph. When you are running interactively, you can keep submitting graph statements. If you were running the above examples and tried to delete your results, you would get a message saying the proc was still active. The QUIT says to end the graph proc so you can do something else such as delete the results. If you leave out your RUN, you won't see your last graph until you issue another proc or data step. For clarity in the log, you always want to use RUN anyway. For this reason, RUN is important even in batch mode. If you leave out the QUIT, you just won't be permitted to do certain actions. QUIT has no impact in batch mode.

```
filename grafout 'mygraph_multi.html';

goptions reset=all
         device=html
         gsfname=grafout
         gsfmode=append
         rotate=landscape;

symbol1 i=join v=star c=blue l=1 h=3 cv=orange;
symbol2 i=spline v=square c=red l=43 h=2 cv=green;
axis1 minor=none label=('This is axis1');
axis2 minor=none label=(a = 90 'This is axis2');

proc gplot data=mydata_bygroup;
 by group;
 plot y*x=z / haxis=axis1 vaxis=axis2 name='bygroups';
 label group='My grouping Variable'
       z="This is z"
       x="This is x";
title f=simplex "This is the " f=triplex "first title "
      f=math "line";
title2 f=script "This a the second title in SCRIPT";
run;
quit;
```

The following is an example of a device that puts all the graphs in one file.

filename grafout 'mygraph_multi.pdf';

```
goptions reset=all
         device=PDF
         gsfname=grafout
         gsfmode=append
         rotate=landscape;

symbol1 i=join v=star c=blue l=1 h=3 cv=orange;
symbol2 i=join v=square c=red l=43 h=2 cv=green;
axis1 minor=none label=('This is axis1');
axis2 minor=none label=(a = 90 'This is axis2');

proc gplot data=mydata_bygroup;
 by group;
 plot y*x=z / haxis=axis1 vaxis=axis2 name='bygroups';
 label group='My grouping Variable'
       z="This is z"
       x="This is x";
 Title “This is from one PROC”;
run;
quit;
```

Caution: When creating these examples, I encountered problems with some fonts I tried to use in the title statement. The graphs would run successfully but opening the PDF would give an error and only part of the graph would be shown. I am not sure if this is a SAS/GRAPH issue or in issue with my version of Acrobat Reader. If you encounter the problem, you'll want to change fonts.

Many SAS users are using ODS to control their SAS output. ODS is also useful with SAS/GRAPH. The following code is very similar to previous examples except for the use of ODS statement. If you aren't familiar with ODS, there are numerous conference presentations and books as well as HELP within SAS.

```
ods listing close;
*** assign graphics options for ODS output ;
*** GIF is the "plain" output - JAVA and ACTIVEX have more;
*** functionality but have other requirements to be able to work;

goptions device=gif transparency noborder;

*** open html destination for ODS output ;
ods html body='odsexample.html'
         contents='contents.html'
         frame='frame.html'
         path="c:\workshop\ws112"
         newfile=output;
```

```
*** Run the graph procedure ;

symbol1 i=join v=star c=blue l=1 h=3 cv=orange;
symbol2 i=spline v=square c=red l=43 h=2 cv=green;
axis1 minor=none label=('This is axis1');
axis2 minor=none label=(a = 90 'This is axis2');

proc gplot data=mydata_bygroup;
 by group;
 plot y*x=z / haxis=axis1 vaxis=axis2 name='bygroups';
 label group='My grouping Variable'
       z="This is z"
       x="This is x";
title f=simplex "This is the " f=triplex "first title "
       f=math "line";
title2 f=script "This is the second title in SCRIPT";
run;
quit;

*** close the html destination ;
ods html close;

*** open the listing destination ;
ods listing;
```

EVEN MORE CHOICES IN SAS 9

All the code shown above works in Version 8.2. In SAS 9, you can now use some ODS styles with SAS/GRAPH. Run the following code to see the built-in styles.

```
proc template;
     list styles;
run;
```

To use one of the styles, just add the following option to the ODS HTML option shown above.

style=your-selected-style

MULTIPLE GRAPHS ON ONE PAGE

The above examples have all been writing the graphs to the default graphics catalog of GSEG. You can also specify the name of a different code by using the GOUT option on the PROC statement. If you are running you code multiple times in the same interactive session until you get the desired output, it is often useful to delete the earlier attempts. This also means the naming of the graphs will occur as though you had just started so you have more control over the names. The firt PROC GREPLAY will delete the existing graphs programmatically. You have several ways to do this manually as well.

You can easily put multiple graphs on one page. Notice there are some additional options being used. The NODISPLAY option means the graphs are written only to the catalog so you can select which onesto view instead of having to page through all the

graphs created. HSIZE and VSIZE reduce the size of the graphs so they fit better in the final graph. On some systems, you may need to use XMAX and YMAX instead.

```
proc greplay igout=temp nofs;
   delete _all_;
run;
quit;

goptions reset=all device=pdf hsize=6.5 in vsize=9 in nodisplay
     gsfname=output gsfmode=replace;
filename output 'c:\workshop\ws112\template_example.pdf';

proc gplot data=mydata_bygroup gout=temp;
 by group;
 plot y*x=z / name='bygroups';
 label group='My grouping Variable'
       z="This is z"
       x="This is x";
 Title "First Title";
 title2 "Second Title";
run;
quit;
```

Once you've created your graphs, you need to create another graph with the desired graphs on one page. Even though 100 graphs are created above, only the first four are shown on a single page. Reset the options to display the graph and reset the size to the full size. There are several built-in templates or you can create your own. PROC GREPLAY does what is says – it replays your existing graphs. The TC or template catalog option points to the built-in templates that are found in SASHELP.TEMPLT. The IGOUT option specifies the graphics catalog to use for the input graphs. The TEMPLATE statement shows the name of the template. L2R2 is 2 boxes left and 2 boxes right. (You can click on VIEW in the catalog to find the description and coordinates of each template.). The TREPLAY statement shows which graph to put in which box based on the names of the graphs. This is why specifying the names of the graphs can be important.

```
goptions display hsize=8 in vsize=10 in;
proc greplay igout=temp nofs tc=sashelp.templt;
template l2r2;
   treplay 1:bygroups 2:bygroup1 3:bygroup2 4:bygroup3;
run;
quit;
```

SUMMARY

SAS/GPAPH has the ability to create any graph you can imagine. If you also know other functions of SAS, you can write code that will adjust the graphs as needed based on your actual data. The above shows just a few examples of the many options you can set using SAS/GRAPH. HELP provides details on these options. Until you become familiar with options, applying just one option at a time as I did in the examples lets you see what changes in the graph. You should also read the log because it can tell you

when SAS made a substitution of a value that you specified. Whether a substitution occurred will depend on the device you used.

I hope this workshop gets you comfortable with the basics so you can harness the power that is available.

ADDITIONAL READING

There are many resources to learn more about SAS/GRAPH and the wide range of graphs that can be created.

You can search SGF (formerly SUGI) proceedings at SUPPORT.SAS.COM. You can also search the proceedings and those from some other SAS conferences at www.lexjansen.com.

SAS also has technical support documents that provide more details on specific topics. If you want to use SAS/GRAPH with Microsoft Office products, you'll definitely want to read TS-674 "An Introduction to Exporting SAS/GRAPH Output to Microsoft Office SAS Release 8.2 and higher." This document will provide significant detail about selecting the appropriate devices and options for your needs.

CONTACT INFORMATION

Deb Cassidy
Deborah.Cassidy@rtp.ppdi.com

Trademark