Use of SAS/AF® and the SAS/GRAPH® Output Class Object to Develop Applications That Can Return Scatterplot Information

Michael Hartman, Schering-Plough Corporation, Union, New Jersey

ABSTRACT

In today's time of 'interactive' analyses, users expect to be able to select data points of a graphic and obtain detailed information about a selected observation. SAS/AF Frame objects exist (e.g. GRAPHICS class object,) where a user can get some information about a selected observation. This class is good for basic use, but only the X and Y values can be obtained upon 'clicking' on a point and users can only get graphics within the scope of this class. For example, users can display confidence intervals, but are restricted to a 95% level. The SAS/GRAPH® Output class provides a mechanism where scatterplots created using line-code SAS can be displayed but, only limited information about the spot such as SpotID, color, etc. can be obtained.

This paper describes the SAS/ GRAPH® scatterplot object and how to develop SAS/AF® applications that allow the user to select a component/spot of any scatterplot graph and determine what part of the graph the spot is from. If the activated spot is determined to be an observation, then one can return detailed information about the selected observation.

Introduction

When a SAS/GRAPH is created, it remains 'separate' from the dataset that was used to obtain the data for the graph. That is, there was no linking of the data to the final graph.

A graph is created utilizing segments. Every segment corresponds to a piece of the graph and these segments can be activated (as HotSpots,) which can return information such as the segment number (SpotID,) color, etc. Since the SAS graph conforms to a structured design, the application developer can determine an offset value. This value would be subtracted from the SpotID value to obtain the correct observation number of the dataset used to create the graph.

The _GET_INFO_ method of the SAS/GRAPH output class can provide information about a hotspot (a user activated segment on the graphics object.) The method returns a SCL list that has information about the spot. Below is a summary of what is contained in this SCL list:

List Item Type Description	Type	Description
SPOT	N	A list defining the hotspot
SPOTID	N	The number of the graph segment of the graph.
SPOTTYPE	N	A number of the type of Graph segment selected.
TEXT	C	Contains the text of the graph segment, if text exists in segment
X	N	X coordinate of selected object (in Pixel units)
Y	N	Y coordinate of selected object (in Pixel units)

TABLE 1: Parameters returned by _GET_INFO_ method.

While the _GET_INFO method returns useful information when the user clicks on an observation, we are not able to tell which observation of the dataset it is from. But, since we can get the segment number (SpotID), one could get the actual observation number of an activated observation spot if they could relate the segment number to the observation number.

I will first discuss the SAS scatterplot's structure and then show an example application that can provide dataset information about the observation of an activated point.

(I'm sure that other graph types also have a structured format, but the focus of this paper is on scatterplots.)

The SAS/Graph® Scatterplot Object

The SAS/GRAPH object has a structured format. When a graph is created, every aspect of the graph (Titles, axis lines, tickmarks, observations, lines, etc.) is added by adding segments. The segments are added in a defined format, starting with segment number 1. Below is a flow of how the segments are added:

The SPOT ID settings are as follows:

Spot ID #	Contents
1 to a	TitleN (Title1= SpotID #1,Title2=SpotID#2,...
(a+1) to b	Footnote1 to b-a (Footnote1=SpotID#a+1)
b+1	Note from NOTE statement option.

[Any boxes around the Title(s), Footnote(s) and any Note (using the BOX= option, count as an additional segment unit.]

b+2	Legend 1 Label (e.g. IDGROUP variable name or label)
b+3 to b+c	Key groups 2 to c=number of groups
b+c+1 to b+c+d	Symbols or lines within each group, from left to right and top to bottom, (within each legend group any symbol types, 3 per group are counted first left to right, and then any joining line is counted)

The next Legend is done the same way, until all legend components are accounted for in the graph. Total number of segments (after any and all segments are added,) is now identified by **e**.

Now the Axis information is implemented into the graph; The primary (leftmost) Y-Axis is formatted first. The Major Tickmarks, minor tickmarks, and labels are all involved in the following manner.

Spot ID #	Contents
e+1 to e+f	Y1 VREF lines 1 to f
e+f+1	Y-Axis Vertical Line
e+f+2	Major TickMark closest to origin

Spot ID #	Contents
e+f+2+1 to e+f+g	One for each MINOR tickmark, where g= the number of ticks specified in the 'MINOR= ' component of the AXIS statement.
e+f+3+g+1	Second Major TickMark
.	
.	Follows same pattern until Last major
.	TickMark is used.

Last Major Tickmark of the Y axis is at SpotID:
e+f+1+(m+(m-1)*w) Where **m**=#Major Tickmarks and **w**=# Minor tickmarks for each segment.

For ease of notation, equate this SpotID to the symbol **Ω**.

Spot ID #	Contents
Ω+1	Y-Axis text Label, if one exists
Ω+1+1	Y-Axis Major Tickmark Label, starting with
to	the Minimum tickmark, ...,
Ω+1+v	for each of v tickmarks.
Ω+1+v	is now equated to the symbol **Λ**

Now the X-Axis information is implemented with SpotID's. The same format is followed as with the SpotID structure of the Y axis.

Spot ID #	Contents
Λ+1	X-Axis (HREF) reference lines 1 to t.
Λ+t+1	X-Axis Horizontal Line
Λ+t+2	Leftmost Major TickMark
Λ+t+3 to Λ+t+3+k	One for each MINOR tickmark, where k= the number of ticks specified in the 'MINOR= ' component of the AXIS statement.
Λ+t+3+k+1	Second Major TickMark
.	
.	Follows same pattern until Last major
.	TickMark is used.
.	

Last Major Tickmark of the X axis is at SpotID:
Λ+t+1+(x+(x-1)*y) Where **x**=#Major Tickmarks and **y**=# Minor tickmarks for each segment.

Equate this quantity to the symbol **∏**

Spot ID #	Contents
∏+1	X-Axis Label, if one exists
∏+2 to ∏+x	X-Axis Major tickmark labels from left to right.

Equate this final Major tickmark's SpotID to **Φ**.

Now if a second Y axis exists (called Y2,) then this information is implemented...

Spot ID #	Contents
Φ+1 to F+q	Y2 Vertical reference lines (VREF's) 1 to q
Φ+q+1	Y2 axis vertical line
Φ+q+2	First major tickmark closest to the x-axis (does not have to be minimum if VREVERSE option is used.)
Φ+q+2+1	One for each MINOR tickmark, where r= the
to	number of ticks specified in the 'MINOR='
Φ+q+r	component of the AXIS statement.
Φ+q+2+r+1	Second Major TickMark
.	
.	Follows same pattern until Last major .
.	TickMark is used.

Last Major Tickmark of the Y2 axis is at SpotID:
Φ+q+1+(mm+(mm-1)*ww) Where **mm**=#Major Tickmarks and **ww**=# Minor tickmarks for each segment.

For ease of notation, equate this segment # to the symbol **Ψ**

Spot ID #	Contents
Ψ+1	Y2-Axis Label, if one exists
Ψ+1+1	Y-Axis Major Tickmark Label, starting with
to	the Minimum tickmark,...,
Ψ+1+v	for each of v Major Tickmark labels.

Ψ+1+v is now equated to the symbol **∅**, the offset value.

Then the curves are added, observation symbols first, then any joining lines.

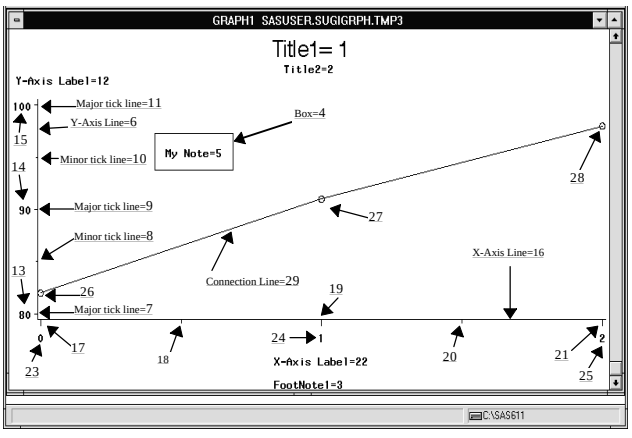
The Offset Value

This value ∅ is the offset needed to determine the correct data observation. Let's say we had a simple scatterplot with a square symbol used to designate each observation. If the user clicked on an observation, the observation number of that data point would be equal to the SpotID-∅.

The information for this observation could be easily obtained from the dataset used to create the graph (using SCL functions such as FETCHOBS, VARTYPE, GETVARN, GETVARC, etc.)

The programmer could easily display the values of the other variables of the dataset for a selected observation. The activation of an observation can also trigger further processing, such as diagnostic checks. Thus, a truly interactive system can be developed in this manner.

A graphical representation of how the SAS graph is structured is provided below:



Scatterplots with Multiple Curves:

Each curve is actually a collection of SpotID's. The format of your graph needs to be known in order to correctly track additional SpotID offsets which may be needed to maintain integrity of observation recognition.

The general structure of the data with respect to SpotID's is as follows:

- SpotID's increment by curve in the order that they are obtained in the dataset (no special sorting is done).
- Each SYMBOL of a curve counts as one SpotID. A joining line is counted as one SpotID.
- The symbols take a segment id before the line (count the symbols first.)

A Simple application describing how this works:

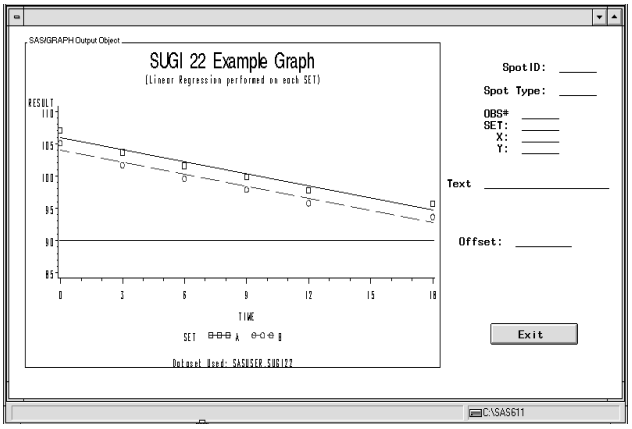
Let us build a simple application that will show us how this works. We are going to need to build a frame with an Exit button, which executes the CANCEL command when activated and an SAS/GRAPH output class object (Make the name of the SAS/GRAPH output object DISGRAPH.) Make this frame as large as you can relative to the display type that you are using. Place a protected CHARACTER text entry field labelled TEXT. Place protected numeric text entry fields named X, Y, OBSNO, and SPOTID on this frame. Also, place an UNPROTECTED numeric text entry field name OFFSET. This will be where the user can enter the appropriate offset value to use. Of course, for actual applications, the SCL code would be designed to determine the correct offset value, based upon the parameters used to create the graph. Place appropriate label fields next to each field.

Enter the below SCL code for the above frame (if the frame's name is WIDGET, then the SCL code program will have the same name as the frame object (ie. WIGET.SCL):

```
/* This SCL Code will manage the DISGRAPH Frame  */
/*   The FRAME associated with this SCL member consists of  */
/*   the following objects:                                  */

/*   a SAS/GRAPH output object named 'GRAPH'                 */
/*          TEXT ENTRY OBJECTS:                              */
/*          SPOTID   - numeric   (from _GET_INFO_ list)      */
/*          SPOTTYPE - numeric   (from _GET_INFO_ list)      */
/*          TEXT     - character (from _GET_INFO_ list)      */
/*          X        - numeric   (X value from the data set) */
/*          Y        - numeric   (Y value from the data set) */
/*      OFFSET  - numeric   (user input value)               */
/*     OBSNO   - numeric   (observation number; spotid-offset  */

 INIT:
   hsflg=0;  /* indicates if a hotspot is currently activated */

/* specify name of graph.  A user input graph name can be */
/* easily implemented by adding a text entry class object     */
/* to the frame, with the name 'CURGRAPH'                     */

CURGRAPH = 'SASUSER.SUGIGRPH.TMP1.GRSEG';

/* set this graph to the name of the SAS/GRAPH output object */

CALL notify('DISGRAPH','_set_graph_',CURGRAPH);

RETURN;

DISGRAPH:
/* this section runs whenever the user activates a spot of  the */
/* graph within the DISGRAPH SAS/GRAPH object by clicking */ /*
the left button of the pointing device with the pointing marker */
/* (e.g. Arrow,) is over the segment to be activated.              */

obsno=0;  /* set the initial observation number equal to zero */

            /* get the information of the spot */
           call notify('pgngraph','_get_info_',infoid);

            /* get total number of spots for this graph */
            totspot=listlen(infoid);

            /* get the number of the graph ID segment */
             spotid=getnitemn(infoid,'spotid');

/* 0 is returned if no segment existed where the user clicked */

 if spotid ^= 0 then do;

/* now get the type of spot that was activated and any possible */ /*
text in the segment.  The spot types are summarized  below */

spottype=getnitemn(infoid,'spottype');
text=getnitemc(infoid,'text');
   end;

  else do;  /* otherwise, tell the use to click on a spot */
  _msg_ = 'Click on an observation for detailed information.';
     spottype=0;
     end;


/* now make sure that a spot on the graph is for a data symbol */
/* the list of possible symbols is summarized below           */

/*          25  dot
            26  rectangle fill
            27  pic
            28  polygon fill
            29  symbol      -  THIS IS THE ONE THAT WE WANT
                                    TO DETECT
            30  text
            31  dashed lines
            32  polylines
            33  polymarkers
            34  rounded rectangle fill
            35  ellipse                              */

if spottype=29 then do;  /* this indicates a symbol */

/* if a current hotspot exists, then delete it */
   if hsflg=1 then call notify('disgraph','_delete_hotspot_','hot1');

/* add a hotspot at this location */
   attrlist = makelist();    /* make an attributes list */

   rc=setnitemc(attrlist,'hot1','name');

   rc=setnitemc(attrlist,'blue','color');
   rc=setnitemn(attrlist,2,'filltype');
   rc=setnitemn(attrlist,spotid,'spotid');
```

```
/* add a hotspot where this activated point is using the values of the
attributes list */

        call notify('disgraph','_add_hotspot_', attrlist);


/* GET THE OBSERVATION NUMBER */
 obsno=spotid-offset;


      if obsno>0 then do;  /* then an observation exists, so lets get
info */

      dsid=open('sasuser.sugi22');
      rc=FETCHOBS(dsid,obsno);

     xvar=varnum(dsid,'time');
     yvar=varnum(dsid,'result');
     setvar=varnum(dsid,'set');

   x=getvarN(dsid,xvar);
   y=getvarN(dsid,yvar);
  set=getvarC(dsid,setvar);

           end;

/* activate the flag which indicates that a hotspot exists */
        hsflg=1;

       END;

 else do;  /* otherwise clear the other fields */
  x=.; y=.; set='';
  end;

 RETURN; /* END OF DISGRAPH SECTION OF CODE */
```

An example of what the frame should look like is listed below:



Note: The Graph will not be displayed until the frame is run and the graph exists.

Below is an example dataset of which we will make a scatterplot of:

```
data sasuser.sugi22;
input obs set $ time result;
cards;
 1 A  0   107.1
 2 A  3   103.7
 3 A  6   101.6
 4 A  9   99.9
 5 A  12  97.8
 6 A  18  95.7
 7 B  0   105.1
 8 B  3   101.7
 9 B  6   99.6
 10 B  9   97.9
 11 B  12  95.8
 12 B  18  93.7
;
run;
```

We want to make a graph using this dataset. Least-Square regression lines need to be calculated for each SET of data.

Use the following SAS line code to produce this graph:

```
goptions goutmode=replace nodisplay;

Title1'SUGI 22 Example Graph';
Title2'(Linear Regression performed on each SET)';
Footnote1'Dataset Used: SASUSER.SUGI22';

SYMBOL1 value=square interpol=RL COLOR=RED LINE=1 R=1;
SYMBOL2 value=CIRCLE interpol=RL COLOR=GREEN LINE=2
R=1;

AXIS1 order=(0 to 18 by 3) major=(number=7)
minor=(number=2);

AXIS2 order=(85 to 110 by 5) major=(number=7)
minor=(number=4);

proc gplot data=SASUSER.SUGI22
gout=SASUSER.SUGIGRPH;
plot RESULT*TIME=SET /NAME='TMP1' haxis=axis1 vaxis=axis2
vref=90 lvref=1 cvref=blue;
run;
quit;   run;
```

This will produce a graph with the name SASUSER.SUGIGRPH.TMP1.GRSEG.

Using the Application

Based upon our counting system to determine the correct offset, the offset value would be 77. So, subtracting this value from the returned SpotID of our frame will give us the correct observation number.

Run the SAS/AF application, specifying the frame that was just built:

(command line command:
AF c={sasuser.sugifrme.graphtst.frame})

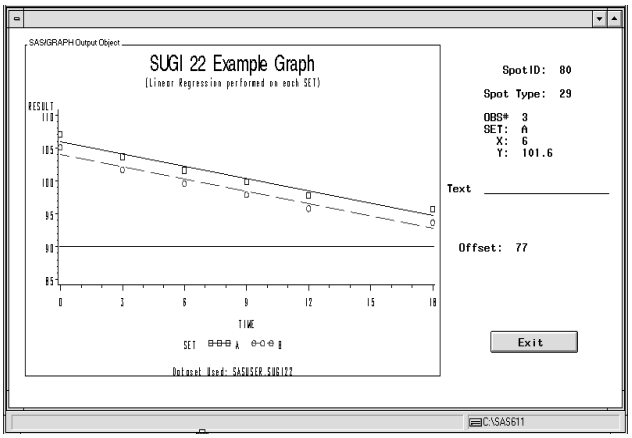
Put the offset value of 77 into the appropriate field.

The graph will be displayed within the frame. Any time that the user selects a valid component of the graph, the frame's text entry fields will be appropriately updated.

If a point on the graph is not activated, then the message bar will tell the user to select an observation on the graph. If an observation is selected on the graph, then the appropriate observation ID value will be returned.

Note that the observation variable within the dataset is not required for this to work. Also, the order of the observations is based upon the order in the actual dataset (i.e. no additional sorting is performed.)

Below is an example with the third observation activated:



Conclusion:

The SAS/GRAPH is definitely not as GUI as other window-based programs. But, through application development using SAS/AF Frame entry objects and SCL, one can develop comprehensive systems to induce a much more GUI graphical environment. Since information can be obtained for every segment within a SAS/GRAPH, one can develop applications that can activate other frames, programs, etc. depending upon what segment has been activated.

Once the correct observation can be identified, the programmer is able to obtain values for any variable in the dataset for that observation number. The most difficult part for the programmer is to develop applications that can effectively determine the correct offset, based upon the *changing* parameters used to develop the displayed graphs. Effective design and development can ensure that the end-user will be given an application that is capable of meeting their needs.

The sample application given in this paper can be used to 'experiment' with a number of different graph types. The programmer can click on any component of the graph and get the corresponding spot id segment number. Once a firm understanding of the graph's structure has been obtained, the developer can expand on this basic code for use in their applications.

Acknowledgments:

I would like to say thanks to all of the extremely knowledgeable and helpful SAS employees that work with the SAS Technical Support team. They are an invaluable resource for working with the end-users for finding solutions to difficult problems. A special thanks goes to Art Alexander, who has given me insight to the SAS graphics structure that would have been difficult to obtain otherwise.

Author Contact Information:

Michael Hartman
Schering-Plough Corporation
1011 Morris Avenue
Union, New Jersey, 07083
Phone: 908-820-6610
e-Mail: michael.hartman@spcorp.com