Dynamic Hotspotting with the SAS/GRAPH® Output Object in SAS/AF® FRAME Entries

Art Alexander

Art Alexander is a senior technical support analyst in the Technical Support Division at SAS Institute. His areas of expertise include SAS/AF FRAME and SAS/GRAPH software. Art has a BS degree in economics from North Carolina State University and has completed his course work for an MA in communications from Regent University. He has been a SAS software user for 11 years.

Abstract

Although the SAS/GRAPH® Output object does not provide for dynamic hotspotting per se, within certain limitations, the object can be made to behave as if had them. By thoroughly understanding the data and knowing how various SAS/GRAPH procedure options affect the graphical output of the data, a link can be established between the data and the graph that behaves in a predictable fashion. This article provides how-to examples for creating and using this data-to-graph link to simulate dynamic hotspots on the SAS/GRAPH Output object.

Introduction

The SAS/AF FRAME® entry Graphics object is a useful tool for constructing graphical, 'point-and-click' interfaces to a user's data. However, it does not allow customization of the graphic to the degree that users of SAS/GRAPH software have become accustomed. The FRAME entry alternative, the SAS/GRAPH Output object, does not have the dynamic hotspotting capability of the Graphics object. So programmers have been faced with a dilemma. Do they use the Graphics object for its functionality or the SAS/GRAPH Output object for its customizability? "Isn't there a way," they ask, "to get dynamic hotspots on a SAS/GRAPH Output object?"

The answer is "Well, sort of." There are no dynamic hotspots for the SAS/GRAPH Output object; however, you can make the object behave as if it had dynamic hotspots, with certain limitations. Some types of graphical output are more conducive to this method than others. For the purpose of illustration here, we use a vertical bar chart. However, this basic approach has been used successfully with scatter plots, regression plots, maps, and even graphics made up entirely of annotation.

Tutorial

In a FRAME entry, the _get_value_ method for the Graphics object will return an SCL list containing information about the data point that was just selected by the user. When the underlying data changes, the values in the list returned by _get_value_ will reflect those changes. When used with the SAS/GRAPH Output object however, the _get_value_ method returns values for the associated segment hotspot. Segment hotspots can only be defined for static GRSEG entries because segment hotspots are tightly coupled to the position of the graphic segment within the stream of segments stored in the GRSEG entry. For this reason, hotspots are removed when the GRSEG entry changes. Therefore, if we want to maintain the dynamic relationship between the graphic and the underlying data, the _get_value_ method is of no use with the SAS/GRAPH Output object.

The only other method available to return information about a selected point on the SAS/GRAPH Output object is the _get_info_ method. At first glance, the items returned in the list do not seem to be of much use:

Item Name	Description
X	the X coordinate of the selection, in pixels
Y	the Y coordinate of the selection, in pixels
text	the text of the graph segment, if selected
spot	a list defining a selected hotspot, if selected
spotid	the number of the graph segment within the graph, beginning with 1
spottype	a number identifying the type of graph segment selected

But there is a little gem hiding there in the list, the spotid. With the spotid, every component (segment) of the graphic output has a sequential position (number) in the graphic output stream (grseg file). If we could somehow map that number back to the underlying data file, we would then have a key to retrieve the data related to that point (segment) from that file or from any other file that contains data related to that key value.

Now that we know there is a sequential relationship between the data and the graphical output, we have to do some detective work to see if the relationship is predictable enough to rely on. First, create a simple bar chart using PROC GCHART and the following code:

```
title 'Vertical Bar Chart Title #1';
title2 'Vertical Bar Chart Title#2';
footnote1 'Vertical Bar Chart Footnote #1';
footnote2 'Vertical Bar Chart Footnote #2';
pattern v=s c=green r=999;
proc gchart data=sasuser.crime(where=(state lt 20));
   vbar state/ sumvar=murder
               discrete
               name='vertbar';
run;
quit;
```

Next, create a frame entry containing a SAS/GRAPH Output object and name it 'Graph'. Finally, add three text entry fields, 'spotid', 'spottype', and 'text'. 'Spotid' and 'spottype' should be numeric and 'text' character. The FRAME entry screen should look something like the one in Figure 1:

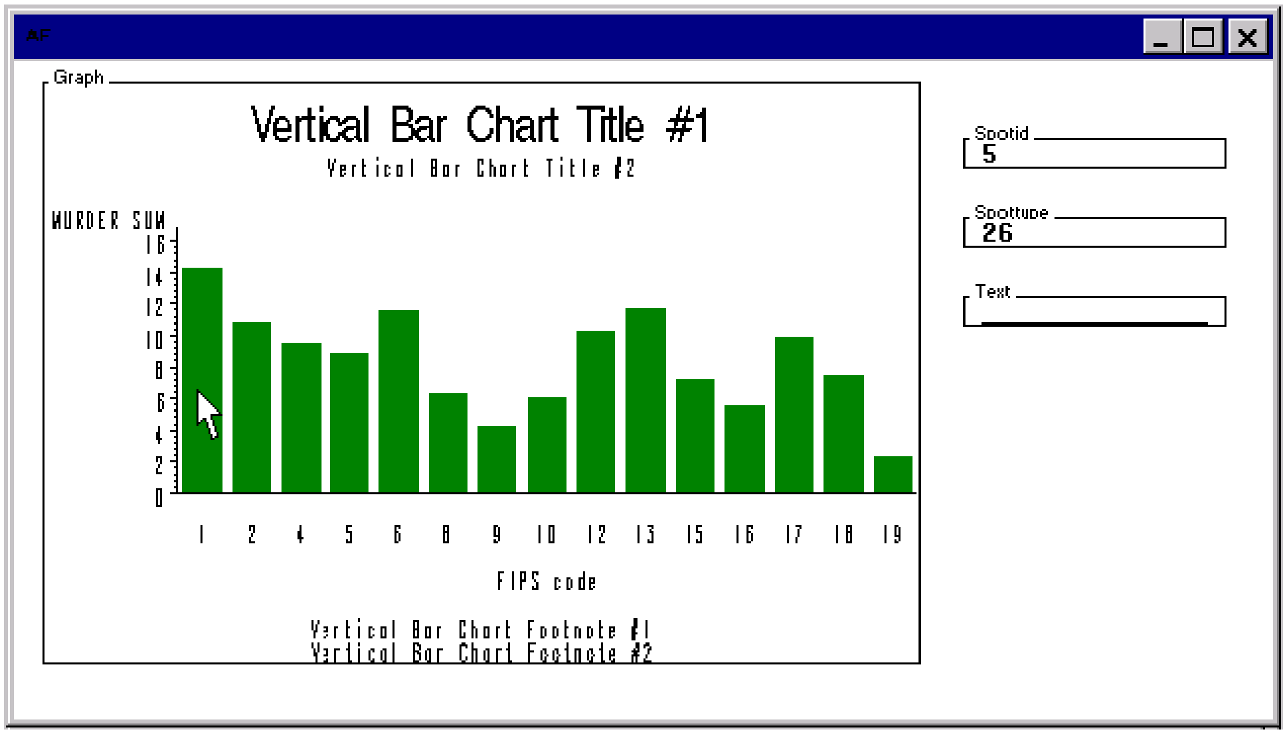

AF
Graph
Vertical Bar Chart Title #1
Vertical Bar Chart Title #2
MURDER SUM
0
2
4
6
8
10
12
14
16
1
2
4
5
6
8
9
10
12
13
15
16
17
18
19
FIPS code
Vertical Bar Chart Footnote #1
Vertical Bar Chart Footnote #2
Spotid
5
Spottype
26
Text

Figure 1

For the SCL source code, enter the following:

```
init:
   call notify('graph','_set_graph_','work.gseg.vertbar.grseg');
return;

graph:
   call notify('graph','_get_info_',infoid);
   spotid=getnitemn(infoid,'spotid');
   spottype=getnitemn(infoid,'spottype');
   text=getnitemc(infoid,'text');
return;
```

Now compile and run testaf on the frame. As you click on various points in the graphic area, you begin to get a feel for the arrangement of the segment information in the graphic output stream. You will find that segment (spotid) number 1 is the first title, number 2 is title 2, and so on. For the first bar, spotid=5. The remaining bars follow in sequence. So, if we created a series of vertical bar charts, each with two titles and two footnotes, would the first bar ALWAYS be segment 5? So it seems. Experiment with graphs of your own data within this framework.

If the first bar in our chart is always segment 5, and we arrange the records in our data in the same order that GCHART displays the bars in the graph, segment 5 (the first bar) should map to record 1 in the data set, segment 6 (the second bar) should map to record 2, and so on. In the simplest case, where there is a single value of the dependent variable for each value of the independent variable, it is easy to exploit this relationship. Simply subtract 4 (one for each title and one for each footnote) from the spotid to find the key to the ordered data set. This scenario can complicate, as we will see later. Save this frame and code for use anytime you want to explore new GRSEG output.

Make a second copy of the frame and add two fields, 'midpoint' and 'sum' (both numeric). For the SCL source, use the code following Figure 2 below. Because we will be changing the PROC GCHART code as we expand this example, the PROC GCHART has been added to the SCL inside a SUBMIT block in the INIT section. This is simpler than having to switch back to the program editor each time to edit and resubmit DATA step and procedure code. Remember, however, that SUBMIT blocks will not process from TESTAF. You must make your changes, compile, and run the code using the following syntax:

```
AF C=<libname>.<catalog>.<framename>.frame
```

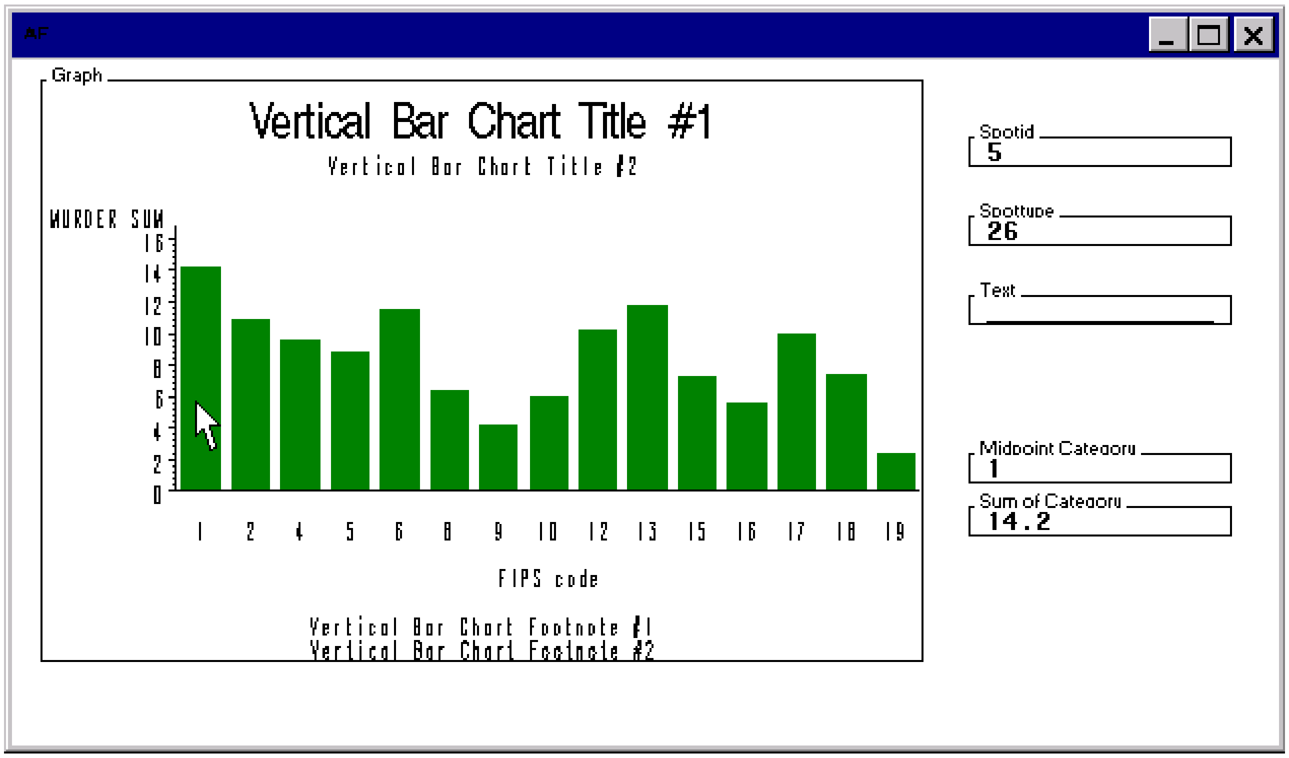


Figure 2

```
init:
   submit continue;
      proc sort data=sasuser.crime(where=(state lt 20))
           out=crimsort;
         by state;
      run;
      goptions goutmode=replace nodisplay;
      title 'Vertical Bar Chart Title #1';
      title2 'Vertical Bar Chart Title #2';
      footnote1 'Vertical Bar Chart Footnote #1';
      footnote2 'Vertical Bar Chart Footnote #2';
      pattern v=s c=green r=999;
      proc gchart data=sasuser.crime(where=(state lt 20));
         vbar state/ sumvar=murder
                     discrete
                     name='vertbar';
      run;
      quit;
      goptions goutmode=append display;
   endsubmit;
   call notify('graph','_set_graph_','work.gseg.vertbar.grseg');
return;

graph:
   call notify('graph','_get_info_',infoid);
   spotid=getnitemn(infoid,'spotid');
   spottype=getnitemn(infoid,'spottype');
   text=getnitemc(infoid,'text');
   if spottype=26 then do;
```

```
         dsid=open('work.crimsort');
         recnum=spotid-4; * since the first bar starts @5;
         rc=fetchobs(dsid,recnum);
         midpoint=getvarn(dsid,varnum(dsid,'state'));
         sumval=getvarn(dsid,varnum(dsid,'murder'));
         dsid=close(dsid);
      end;
      else do;
         midpoint=_blank_;
         sumval=_blank_;
      end;
   return;
```

The PROC SORT ensures that the data set is sorted in ascending order by 'state', which is the VBAR variable in the PROC GCHART. Because we are also using the DISCRETE option in the VBAR statement, GCHART will create a separate bar for each discrete value of state. By default, GCHART will arrange the bars in ascending order by state, and because there is only one value of 'murder' (the SUMVAR) for each value of 'state', the data file order will now match the bar order in the chart.

We are also checking spottype in the code to insure that the selected spotid is actually a bar. Because only bars have data in our data file, the program would halt without this check if we clicked anywhere other than on a bar. Referring to the table on page 8 of the SAS/GRAPH Output Class in the *SAS/AF® Software: FRAME Class Dictionary* you will find that a spottype of 26 is 'polygon fill'. The only areas on our graph that consist of polygon fill are the bars. Therefore, if the user clicks on a spot that returns a spottype of 26, it must be one of the bars. All we then have to do is subtract the offset for the 2 titles and 2 footnotes and we have the key to the corresponding record in the sorted data set.

After fetching the proper record, the code assigns the selected value of the independent variable 'state' to the screen variable 'midpoint' and the corresponding value of the dependent variable 'murder' to the screen variable 'sumval'. As a result, when you select a bar on the screen, you will see the data values retrieved from the data set displayed in these two fields. You can then cross verify the data by comparing the values in these fields with the value on the bar chart.

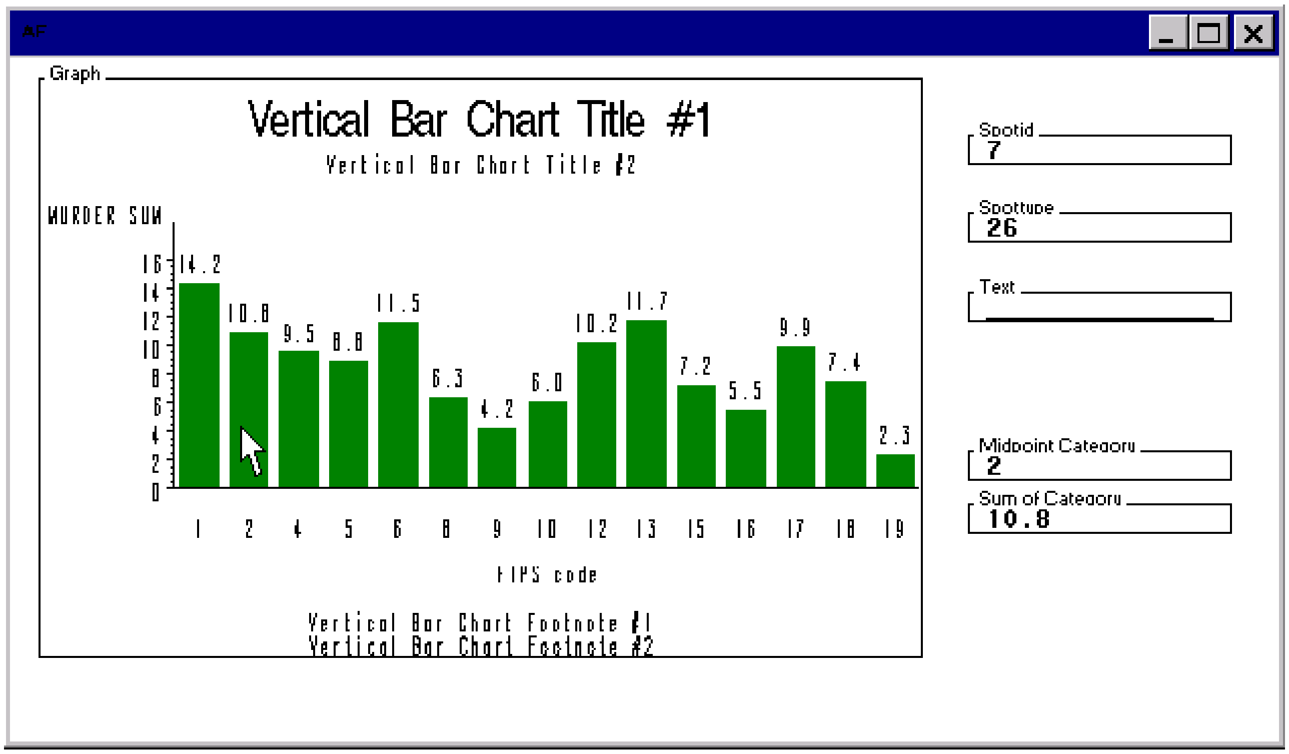
As mentioned earlier, there are factors that can complicate the scenario presented above. If you change the options in the PROC GCHART, you must understand their effect on the graphic output and compensate for the changes in your SCL code. Suppose we want to add the numeric SUM value above each bar on the graph using the SUM option in the VBAR statement. Each annotated number will be an additional segment or spotid in the resulting GRSEG. Going back to our basic frame used in Figure 1 to explore the graph, we find that the number above each bar follows the respective bar in the output sequence. So, instead of a neat, sequential ordering of the bars following segment (spotid) 5, we discover that we now have the following sequence:

spotid #	graphic item	spottype	description
5	bar #1	26	filled polygon
6	sumvalue	30	text
7	bar #2	26	filled polygon
8	sumvalue	30	text
etc...			

The pattern is still predictable, we just have to modify our formula to work with the alternating bar, value, bar, value… ordering of the spotid. If we change the line in the previous example from

```
   recnum=spotid-4;
```
to
```
   recnum=int(spotid/2)-1;
```

we can adjust for the new sequence and still find the key for the corresponding data record. The screen graphic with the SUM values added would look like the one in Figure 3.


AF
Graph
Vertical Bar Chart Title #1
Vertical Bar Chart Title #2
MURDER SUM
0
2
4
6
8
10
12
14
16
14.2
10.8
9.5
8.8
11.5
6.3
4.2
6.0
10.2
11.7
7.2
5.5
9.9
7.4
2.3
1
2
4
5
6
8
9
10
12
13
15
16
17
18
19
FIPS code
Vertical Bar Chart Footnote #1
Vertical Bar Chart Footnote #2

Spotid: 7

Spottype: 26

Text:

Midpoint Category: 2

Sum of Category: 10.8

Figure 3

A more complicated scenario consists of data that contains multiple dependent variable values for each independent variable value. In this case, each bar on the chart would represent a summation of the dependent values for each midpoint value. This forces us to modify our approach somewhat. We must first decide what we are interested in displaying or retrieving when the user selects a bar. Do we simply want to use the sum of the midpoint category, or do we need to retrieve all individual dependent values for the selected midpoint value?

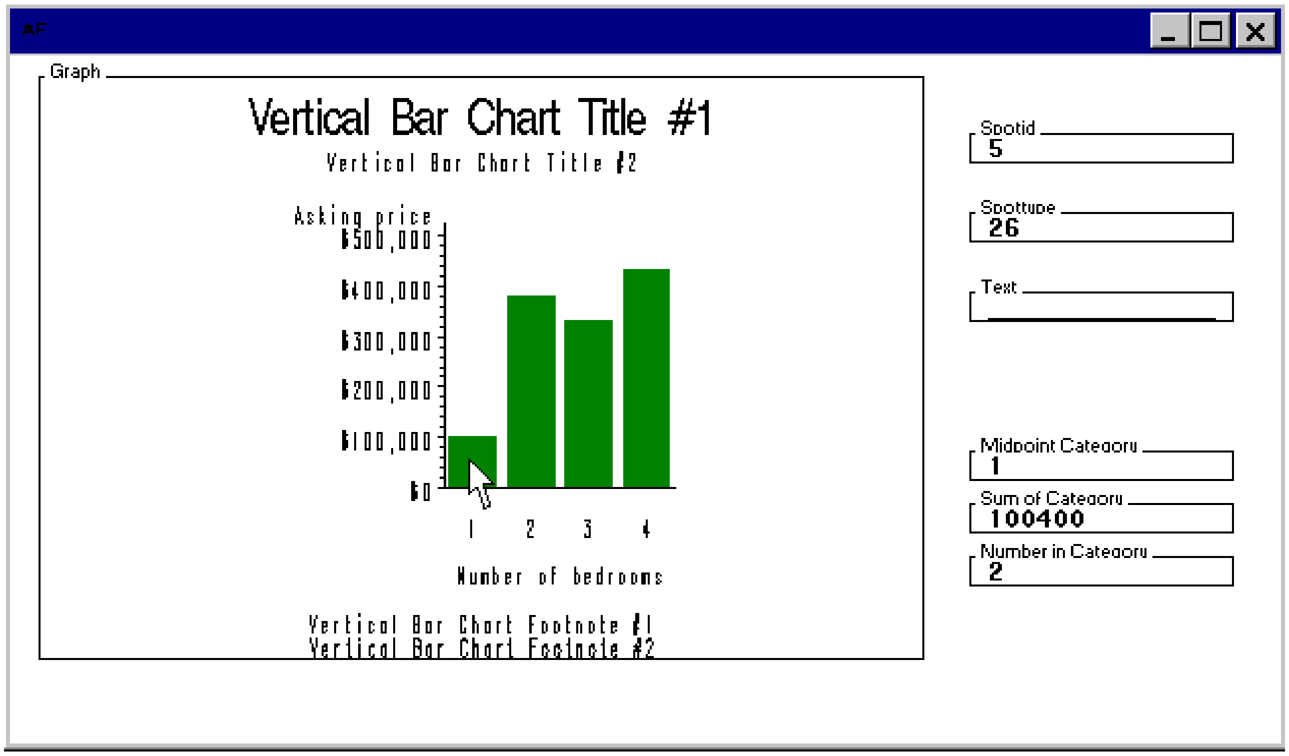
Either way, we must first perform a summation of the underlying data and order that data in the same way that PROC GCHART will sum and order the graphics output to provide ourselves with the graphic-to-data link. In the next example, we use the simplest case, that of retrieving the summation value. First, change the data set to one that contains multiple dependent values for each midpoint value. We use SASUSER.HOUSES. On the frame add a numeric text entry field 'numobs', which will be used to display the number of dependent values comprising the midpoint total. Modify the SCL code as follows:

```
init:
   submit continue;
      proc sort data=sasuser.houses
           out=housort;
         by bedrooms;
      run;
      proc means data=housort noprint;
         var price;
         by bedrooms;
         output out=sumdata sum=pricesum;
      run;
      goptions goutmode=replace nodisplay;
      title 'Vertical Bar Chart Title #1';
      title2 'Vertical Bar Chart Title #2';
      footnote1 'Vertical Bar Chart Footnote #1';
```

```
         footnote2 'Vertical Bar Chart Footnote #2';
         pattern v=s c=green r=999;
         proc gchart data=sasuser.houses;
            vbar bedrooms/ sumvar=price
                           discrete
                           name='vertbar';
         run;
         quit;
         goptions goutmode=append display;
      endsubmit;
      call notify('graph','_set_graph_','work.gseg.vertbar.grseg');
   return;

   graph:
      call notify('graph','_get_info_',infoid);
      spotid=getnitemn(infoid,'spotid');
      spottype=getnitemn(infoid,'spottype');
      text=getnitemc(infoid,'text');
      if spottype=26 then do;
         dsid=open('work.sumdata');
         recnum=spotid-4; * since the first bar starts @5;
         rc=fetchobs(dsid,recnum);
         midpoint=getvarn(dsid,varnum(dsid,'bedrooms'));
         sumval=getvarn(dsid,varnum(dsid,'pricesum'));
         numobs=getvarn(dsid,varnum(dsid,'_freq_'));
         dsid=close(dsid);
      end;
      else do;
         midpoint=_blank_;
         sumval=_blank_;
         numobs=_blank_;
      end;
   return;
```

The PROC MEANS is used to sum all prices for each midpoint value of the independent variable 'bedrooms'. After the original data has been sorted and summed, the output data set from the PROC MEANS is in the proper order to use the simple single-bar-to-single-record mapping scheme. When the program is running, the screen chart will look like the one in Figure 4 below.


AF
Graph
Vertical Bar Chart Title #1
Vertical Bar Chart Title #2
Asking price
$500,000
$400,000
$300,000
$200,000
$100,000
$0
1
2
3
4
Number of bedrooms
Vertical Bar Chart Footnote #1
Vertical Bar Chart Footnote #2

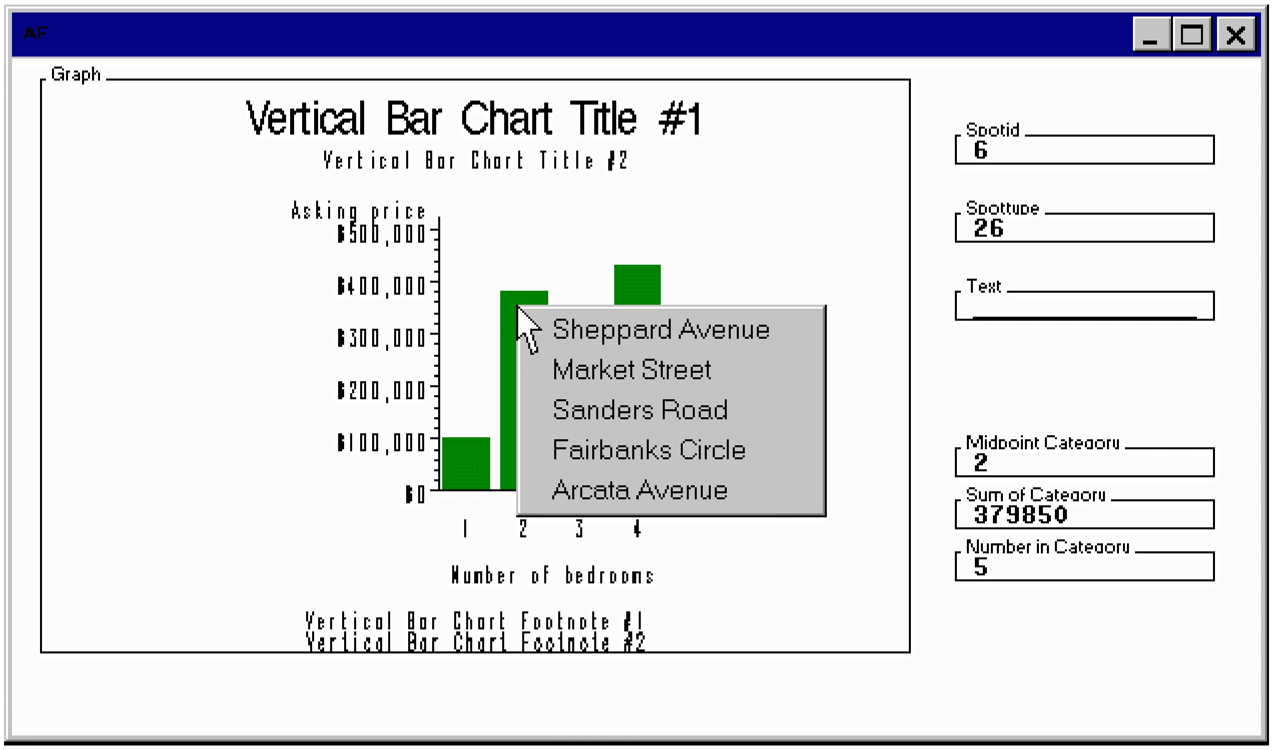
Spotid: 5
Spottype: 26
Text:
Midpoint Category: 1
Sum of Category: 100400
Number in Category: 2

Figure 4

If we needed to retrieve all of the individual values that contributed to the sum for a particular midpoint value, we could go a step further. Using the midpoint value retrieved from the list returned by the _get_info_ method to construct a WHERE clause, apply the WHERE clause to the original data set and fetch those observations. Suppose we wanted a quick look at the addresses of all houses in the data set, which had the selected number of bedrooms. Take the section labeled 'graph:' in the previous SCL example and insert the following lines immediately before the first 'end' statement, inside the 'if spotid=26 then do' section:

```
newid=open('sasuser.houses');
poplist=makelist();
whrc=where(newid,'bedrooms='||midpoint);
do while(fetch(newid)=0);
   rc=insertc(poplist,getvarc(newid,varnum(newid,'street')),-1);
end;
newid=close(newid);
refresh;
rc=popmenu(poplist);
poplist=dellist(poplist);
```

When the user selects a bar, this modification will display a pop-up list of the addresses of all houses in the data set that have the number of bedrooms represented by the selected bar, as in Figure 5 below.



Spotid: 6

Spottype: 26

Text:

Midpoint Category: 2

Sum of Category: 379850

Number in Category: 5

Figure 5

Conclusion

There are many variations on this basic theme. The main point is that by thoroughly understanding your data and knowing how various SAS/GRAPH procedure options will affect the graphical output of the data, it is possible to form a link between the data and the graph that behaves in a predictable fashion. If you then rigidly adhere to the procedures and options used in setting up that link, you can use the SAS/GRAPH Output object for all of its inherent aesthetic appeal, without giving up the dynamic selectability of the Graphics object.

References

SAS/AF® Software: FRAME Class Dictionary, Version 6, First Edition, Cary, NC: SAS Institute Inc., 1995.

SAS/AF® Software: FRAME Application Development Concepts, Version 6, First Edition, Cary, NC: SAS Institute Inc., 1995.

SAS® Screen Control Language: Reference, Version 6, Second Edition, Cary, NC: SAS Institute Inc., 1994.

SAS/AF® Software: FRAME Entry Usage and Reference, Version 6, First Edition, Cary, NC: SAS Institute Inc., 1993.

SAS/GRAPH® Software: Reference, Version 6, First Edition, Cary, NC: SAS Institute Inc., 1990.