Object-Oriented Graphical User Interface (OOGUI) Using FRAME Entries in SAS/AF ® Software Part II

Jeff Cartier, SAS Institute Inc., Cary, NC

ABSTRACT

The SAS/AF® FRAME entry offers many widgets that can be creatively combined to implement nearly any GUI design. New SCL features allow widgets to communicate with one another in novel ways. This paper illustrates variations in FRAME design and new Screen Control Language features that control widget interaction.

INTRODUCTION

As the preceding tutorial has shown, creating and moving widgets in a frame is very simple. Our design of GRAPH1.FRAME is shown in Figure 1. The List Box widget is named LISTBOX and the large region to its right is a SAS/GRAPH® Output widget named GRAPH.

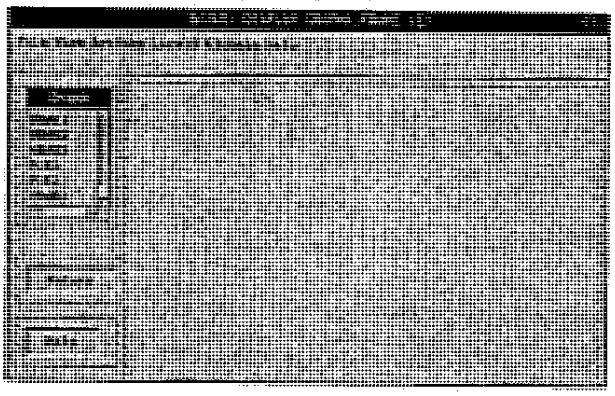

Figure 1 GRAPH1.FRAME in Build Mode

The SCL program GRAPH1.SCL is shown below.

```
length name $ 8;

LISTBOX:
  call notify('listbox','_get_last_sel_',
              row,issel,name);
  graph='SUGI.GRAPHS.'||trim(name)||'.GRSEG';
return;
```

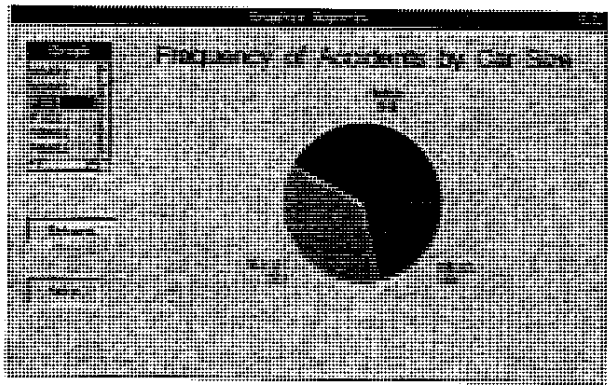

Figure 2 GRAPH1.FRAME in Execution Mode

INTERFACE DESIGN ISSUES

dWhen designing the interface, you often encounter a problem fitting in all the widgets needed to accomplish the task. In our initial design, you can scroll the List Box horizontally to reveal the description of the graph. If we made the List Box wider to reveal more text at all times, the area left for the graph would be too small.

The FRAME entry supports *overlapping* filled regions. In the GRAPH2.FRAME (Figure 3), a larger List Box widget has been positioned on top of an existing SAS/GRAPH Output widget. A new Push Button has been added that enables the user to toggle between viewing a selected graph and the List Box.

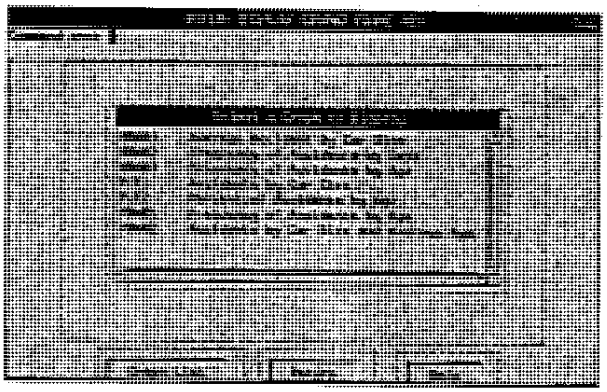

Figure 3 GRAPH2.FRAME with Overlapping Widgets

The region manager PUSH and POP commands enable you to control which overlayed widget is on top so you can set its attributes.

General Rule for Overlaying filled regions:

- A graphic widget can overlay or be overlayed by any widget.
- A non-graphic widget cannot overlay a non-graphic widget.

Graphic Widgets	Non-Graphic Widgets
Graphic Text	Text Entry
SAS GRAPH Output	Text Label
Graphics	Control
CSF	Check Box
Hotspot	Radio Box
Container Box	List Box
Extended Table	Icon
Block	
Push Button	
Slider	
Scrollbar	

Figures 4 and 5 show the GRAPH2.FRAME during execution. Also note that the Graph List button is *grayed* (disabled) when the List Box is visible.

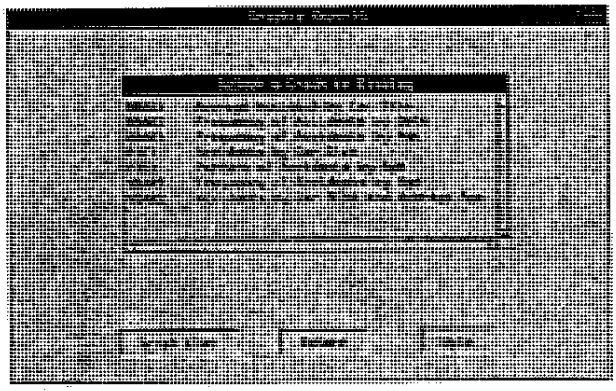

Figure 4 GRAPH2.FRAME in Execution Mode

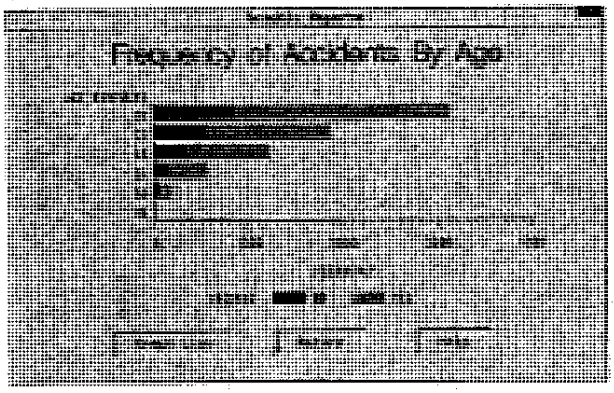

Figure 5 GRAPH2.FRAME in Execution Mode

The FRAME entry's source program GRAPH2.SCL is shown below.

```
length name $8;

INIT:
  rc=gray('gbutton');
return;

LISTBOX:
  call notify('listbox','_get_last_sel_',
              row,issel,name);
  graph='SUGI.GRAPHS.'||trim(name)||'.GRSEG';
  call notify('listbox','_hide_');
  call notify('graph','_unhide_');
  rc=ungray('gbutton');
return;

GBUTTON:
  call notify('listbox','_unhide_');
  call notify('graph','_hide_');
  rc=gray('gbutton');
return;

HELP:
  call wregion(2,10,21,60);
  call display('sugi.frame.help.frame');
return;
```

The _HIDE_ and _UNHIDE_ methods allow the program to send a message to a widget to control whether it is visible or not. These methods apply to any widget. There is also an _IS_HIDDEN_ method that asks a widget to return its *state* (1=hidden, 0=not hidden).

Another possibility for a window design is shown in GRAPH3.FRAME (Figure 6). This design uses a Text Entry widget named TEXT to allow the user to type the name of the graph or to click on a Control widget named ARROW to get a selection list. The SAS/GRAPH Output widget is named GRAPH. Figure 7 shows GRAPH3.FRAME during execution, after clicking on ARROW.

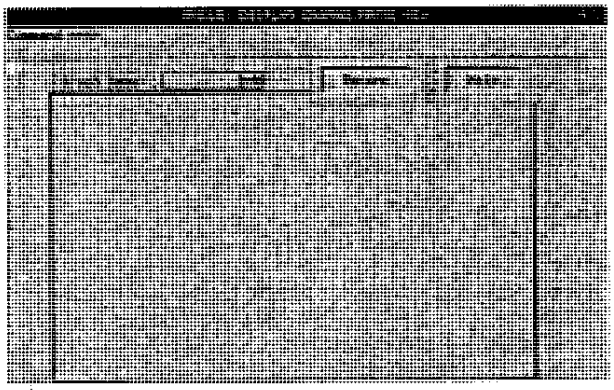

Figure 6 GRAPH3.FRAME in Build Mode

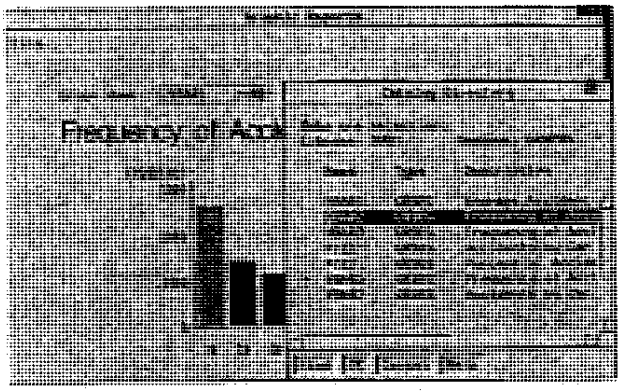

Figure 7 GRAPH3.FRAME in Execution Mode

This design has the advantage that the Text Label, Text Entry and Control widgets are small regions and the user who knows the name of a graph can type it in without having to choose from a selection list.

The supporting program GRAPH3.SCL is shown below.

```
length gname $ 35;

INIT:
  control error;
return;

TEXT:
  erroroff text;
  _msg_=_blank_;
  graph=_blank_;
  if text=_blank_ then return;
  gname='SUGI.GRAPHS.'||trim(text)||'.GRSEG';
  if cexist(gname) then graph=gname;
  else do;
     erroron text;
     _msg_='Press ARROW for selection.';
  end;
return;

ARROW:
  gname=catlist('SUGI.GRAPHS','GRSEG',1,'y');
  if gname ne _blank_ then do;
     graph=gname;
     text=scan(gname,3,'.');
     if error(text)=1 then erroroff text;
  end;
return;
```

The program must validate any value typed in the Text Entry object named TEXT because there is no Type attribute for catalog entries. Figure 8 shows the Object Attribute window for TEXT. Notice the Type attribute is set to NAME to enforce that the entered value conforms to SAS® naming convention.

The CATLIST function is used to present the selection list and return the four-level name of the selected entry.

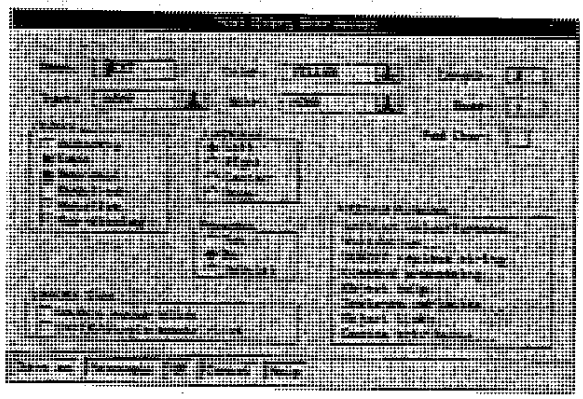

Figure 8 Attributes for Text Entry Widget

The design for REPORT.FRAME is consistent with that of GRAPH3.FRAME. In this case, the Text Entry widget named REPORT is used in conjunction with a Control object named ARROW to obtain the name of an existing printed report stored in a catalog entry of type OUTPUT. The stored output is then displayed in an Extended Table widget (Figure 9).

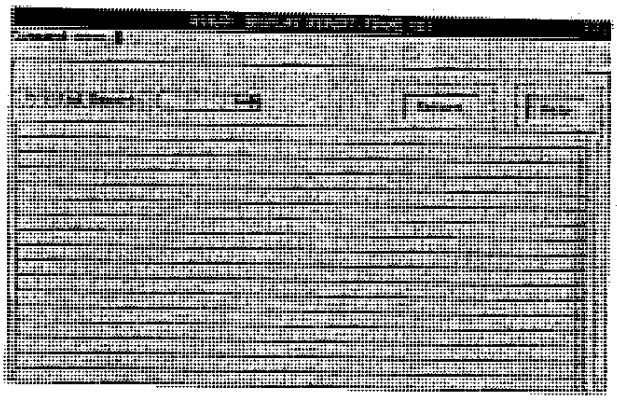

Figure 9 REPORT.FRAME in Build Mode

This FRAME entry uses a different strategy to present a selection list of reports. Notice that a List Box is presented during execution (Figure 10), but there is no List Box widget in the Frame Entry.

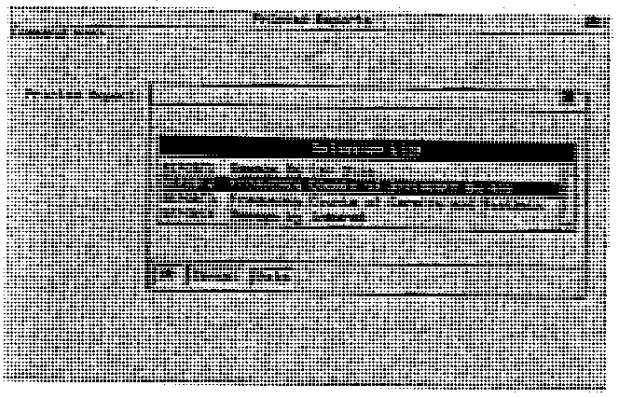

Figure 11 REPORT.FRAME in Execution Mode

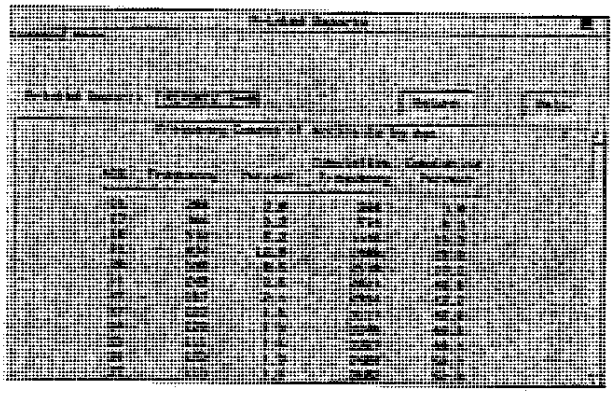

Figure 11 REPORT.FRAME in Execution Mode

A portion of REPORT.SCL is shown next. These coding segments provide an alternative to using functions like DIRLIST, VARLIST, CATLIST, and DATALISTC/DATALISTN functions for displaying selection windows.

The coding also illustrates how *list functions* can be used in SCL programs. An SCL list is like a dynamic array that can grow or shrink during execution. It is stored in memory and persists for the duration of the application.

```
INIT:
  dsid=open('sugi.oview');
  replist=makelist();
  numval=0;
  rc=lvarlevel(dsid,'descrip',numval,replist);
  replist=revlist(replist);.
  rc=close(dsid);
return;

ARROW:
  output=_blank_;
  call notify('report','_column_',repcol);
  call notify('report','_row_',reprow);
  item=popmenu(replist,4,reprow+3,repcol-2);
  if item gt 0 then do;
     report=getitemc(replist,item);
     output='SUGI.OUTPUT.'||
            trim(report)||'.OUTPUT';
     link report;
  end;
   /* REPORT is the labeled section for
      Text Entry widget. This section and
      the section for the Extended Table
      are not shown. */
return;
```

This idea behind this code is to create an SCL list named REPLIST and populate it with unique values of a variable (DESCRIP) in a SAS data set (SUGI.OVIEW). The LVARLEVEL function does this and reports the number of unique values (NUMVAL) that were placed in the list. The REVLIST function reverses the order the items in the list.

Once the list is populated, other list functions can present selection windows based on the list, eliminating repeated disk I/O required by functions like DATALISTC.

The _COLUMN_ and _ROW_ methods can be used to locate the upper lefthand corner of any widget region in the FRAME entry. The POPMENU function displays a selection list based on items in an SCL list. If the number of list items does not exceed the depth of the FRAME entry, a list similar in appearance to the Resource list is displayed. When the number of items is larger, the POPMENU function automatically converts the selection list to a List Box presentation style. If you include optional arguments to the POPMENU function, you can explicitly control the depth of the list (4) and where it should appear. The POPMENU function returns the position of the selected item in the list. The GETITEMC function returns the value of a list item based on its position in the list.

This technique requires a little more coding than placing a List Box widget in the Frame entry, but offers a good alternative when you want customized selection lists or when you do not have enough space in the Master Region for a lot of widgets.

Summary

The FRAME entry offers many possibilities for common graphical user interface designs. Widget methods and SCL functions are the application developer's tools for controlling the communication among widgets.

The next tutorial will show how you can create generic software units like a Text Entry widget that accepts only name of catalog entries.

REFERENCES

SAS/AF Software: Frame Entry Usage and Reference, Version 6, First Edition

SAS Technical Report P-216, *SAS/AF Software, SAS/FSP Software, and SAS Screen Control Language: Changes and Enhancements, Release 6.07.*