

Using SLIST Catalog Entries to Store FRAME Entry Object Information

Randy C. Finch, Tennessee Valley Authority, Muscle Shoals, AL 35660-1010

Abstract

FRAME entries are powerful for application development, offering an object-oriented interactive environment to the programmer. For some applications, a way of storing and retrieving FRAME object attributes is needed. SCL lists and SLIST entries are convenient for this purpose. This article describes how SLIST entries can be used for storing and retrieving object information from a permanent user catalog as well as for synchronizing object information with multi-page catalog output such as GRSEG entries.

Introduction

Version 6.08 of the SAS® System introduced FRAME entries to SAS/AF®. FRAME entries are created in an object-oriented interactive graphical environment. Various types of user-interface objects can be added to a FRAME and, unlike fields in PROGRAM entries, rearranged easily. Each object has a number of attributes that can be set at design or run time. Each object also has a number of methods that can perform operations on the object. Sometimes it is useful to store object attributes for later use. SCL lists are convenient for this task. SCL lists are similar to arrays in that they are an ordered collection of data; however, the former can store many different data types and can even nest lists within lists. SCL lists can be easily stored in a catalog as an SLIST entry. Thus, an SCL list can be filled with the attributes of FRAME entry objects and then stored in an SLIST entry for later retrieval.

Creating a FRAME Entry

In order to create a FRAME entry within an existing SAS catalog, execute the BUILD command at the SAS command prompt or invoke the BUILD procedure (e.g. PROC BUILD; RUN;) from the Program Editor. Next, select the library and catalog to which the new entry will be added from the lists presented by the BUILD procedure. Finally, either execute the EDIT FRAMENAM.FRAME command at the SAS command prompt or select File/New/Entry from the BUILD window's menu. When using the latter method, a dialog box will appear. Enter the name of the entry and select the radio button next to the word FRAME to indicate that this is the type of entry to be created. A blank FRAME entry DISPLAY window appears with a large box delineating the area that objects can occupy. User interface objects can be added to the FRAME by selecting the Make pop-up menu item. (The pop-up menu is available by pressing the right mouse button.) A long list of available objects, such as Text Entry, Check Box, and Push Button, will then appear in another pop-up menu. After selecting one of these objects, an outline of the object appears on screen. It can be positioned at the appropriate place on the DISPLAY window. A mouse button click opens a window containing information about the object's attributes. If necessary, this information can be modified. When the attributes window is closed, the image of the

object appears on the screen.

Upon completing the layout of the user interface, functionality can be added to it by writing SCL code. Select the Locals/Edit SCL Code menu item to open the SCL editor. (Note that the Locals menu appears in either the menu bar or the pop-up menu displayed by clicking the right mouse button immediately above the large box in the FRAME.) As with PROGRAM entries, the SCL code can contain reserved label sections. The INIT label executes when the FRAME entry is first called; the TERM label executes when the FRAME entry terminates. Sections of code that are labeled with the name of an object execute whenever the user interacts with that object. For detailed information about creating FRAME entries see the manual SAS/AF Software: *FRAME Entry*.

Looking at a FRAME Entry Example

Figure 1 shows a FRAME entry named GRAPHIC.FRAME, which displays plots of data set variables. The user is allowed to select the library, the data set, the independent variable, and up to four dependent variables for the plot. In addition, the user can enter a title and a subtitle for the graph and specify whether the graph should be displayed in color or black and white (B&W). Printer output can also be generated.

GRAPHIC.FRAME contains several different types of objects including 10 push buttons, two graphic text objects, four list boxes, two radio boxes, two text entry objects, one SAS/GRAPH® output object, one scrollbar, and three containers. The push buttons appear at the top and in the lower right of the FRAME and are used for taking actions such as getting help, plotting or printing a graph, deleting a graph, etc. The two large titles near the top of the screen are graphic text objects. The four list boxes are located in the lower left corner of the screen and, from left to right, list available libraries, data sets, independent variables, and dependent variables. The radio boxes appear just to the right of the list boxes. One radio box is used to select either color or B&W graphs; the other is used to select whether the graph will appear within the SAS/GRAPH output object on the FRAME or full screen using the GRAPH1 command. The text entry objects below the radio buttons are used for entering the graph's title and subtitle. The SAS/GRAPH output object is the large rectangle in the middle of the screen and is used for displaying SAS/GRAPH generated GRSEG entries from a catalog. The vertical scrollbar to the immediate right of the SAS/GRAPH output object allows the user to scan through all the GRSEG entries in a SAS catalog. Finally, the containers are used to set apart the SAS/GRAPH output object, the radio boxes and text entry objects, and the six push buttons in the lower right corner of the FRAME.

Storing Object Information in an SLIST Entry

Once a user makes certain selections on a FRAME, he may want

to save the selections to disk and retrieve them later so he does not have to make the same selections every time he runs the application. In the case of GRAPHIC.FRAME, the settings of interest are the independent variable (one selection allowed), the dependent variables (four selections allowed), the graph title, and the subtitle. All of this information can be stored in an SCL list which can in turn be saved as an SLIST entry within a SAS catalog. A SAS data set could be used to store the object attribute information, but an SCL list is easier because data types do not need to be declared and related attributes can be grouped as nested sublists within the main list.

In addition to allowing the user to save object information in an SLIST entry within a permanent user catalog, it is also useful to store this information as an SLIST entry within the same temporary catalog used to store the GRSEG entries for display within the SAS/GRAPH output object. This allows the object attributes to change to match the currently displayed GRSEG as the user scrolls through the entries using the vertical scrollbar. If the object attributes were not changed to match the currently displayed graph, it could be very disconcerting to the user because the objects would not reflect the settings used to generate the graph. In addition to the variables and titles stored in a permanent SLIST entry (discussed above), temporary SLIST entries need to store the library name, the data set name, and the appearance of the graph (color or B&W). By giving the SLIST entry the same name as its corresponding GRSEG entry, the SCL code for the FRAME entry can easily match the SLIST entries with the GRSEG entries. GRAPHIC.FRAME uses the standard naming convention for entries generated by the GPLOT procedure (GPLOT, GPLOT1, GPLOT2, etc.).

The following section of code, labeled SAVE, is executed when the user selects the push button named SAVE.

Storing Object Information in a Permanent SLIST Entry

```
001 SAVE;
002 link GETFILE; /* GETFILE code NOT SHOWN */
003 if fname ^= _BLANK_ then do;
004     totalnam='GREPORT.GRAPHS.'
           ||fname||'.SLIST';
005     saveset=makelist();
006     ylist=makelist();
007     link SAVELIST;
008     rc=savelist('CATALOG',totalnam,saveset);
009     rc=dellist(ylist,'Y');
010     rc=dellist(saveset,'Y');
011     _msg_='Settings saved in file '
           ||totalnam;
012 end;
013 else do;
014     _msg_='Invalid filename. Settings not
           saved.';
015 end;
016 RETURN;
```

First, a name for storing the SLIST entry is obtained (line 2). Next, two SCL lists, named saveset and ylist, are created with the makelist function (lines 5-6), and another section of code labeled

SAVELIST is linked.

Building the SCL List to be Stored in an SLIST Entry

```
021 SAVELIST:
022 call notify('XVAR','_SELECTED_',1,rownum);
023 call notify('XVAR','_GET_TEXT_',
           rownum,rowtext);
024 saveset=insertc(saveset,GTITLE,-1,'GTITLE');
025 saveset=insertc(saveset,GSUBTITL,
           -1,'GSUBTITL');
026 saveset=insertc(saveset,rowtext,
           -1,'XVARSEL');
027 call notify('YVARS','_GET_NSELECT_',numvar);
028 do i=1 to numvar;
029     call notify('YVARS','_SELECTED_',
           i,rownum);
030     call notify('YVARS','_GET_TEXT_',
           rownum,rowtext);
031     ylist=insertc(ylist,rowtext,
           -1,'YVARSEL' || i);
032 end;
033 saveset=insertl(saveset,ylist,-1,'YLIST');
034 RETURN;
```

The SAVELIST code extracts information from objects on the frame and inserts it into the proper SCL list. Information is extracted from an object by sending a message to the object using the call notify function. For instance, to obtain the selected independent variable, the XVAR object is sent two messages (lines 22-23). In line 22, a _SELECTED_ message is sent to the XVAR list box object. This message asks the object to return the row number of the first selected item in the object to the rownum variable. The selection number (third parameter) can only be one for XVAR since it only allows a single selection, but it can be different from one for list boxes that allow multiple selections. In line 23, a _GET_TEXT_ message is sent to the XVAR list box. This message asks the object to return the text in the specified row number to the rowtext variable.

The name of an object can also be used as a variable in the SCL code. The value of the variable is dependent on the type of object being referenced. For example, if a text entry object is named GTITLE, then the variable GTITLE contains the string that the user has typed into the object. This text can also be obtained by sending a _GET_TEXT_ message to the object. Thus, the following two lines of code are equivalent.

```
objtext = GTITLE;
call notify('GTITLE','_GET_TEXT_',objtext);
```

The name of the list box, GTITLE, is used to obtain the title in the SAVELIST code (line 24). The insertc function inserts character information into an SCL list. Notice that GTITLE is used as a variable. This line of code inserts the text in the GTITLE object (second parameter) at the end (third parameter) of the saveset SCL list (first parameter) with an item name of GTITLE (fourth parameter). The -1 causes the item to be inserted at the end of the list. (A positive number indicates the exact position from the

beginning of the list at which the item should be inserted; a negative number indicates the exact position from the end of the list at which the item should be inserted.) The item name is not required, but it is convenient. Without the name, items have to be referenced by position in the list. With the name, items can be referenced by name without regard to their position in the list.

The **SAVELIST** code inserts the graphics title (**GTITLE**), the graphics subtitle (**GSUBTITL**), and the selected independent variable (**rowtext**) into the **saveset** SCL list (lines 24-26). Then the selected dependent variables are inserted. However, since a variable number of selections (1-4) are possible, it is convenient to insert these selections into a separate list (**ylist**) and then insert this list into the **saveset** list. First, the number of selections is obtained by sending the **YVARS** object a **_GET_NSELECT_** message (line 27). Next, a **do** loop extracts the selected lines of text and insert them into the **ylist** SCL list with names of **YVARSEL1**, **YVARSEL2**, etc. (lines 28-32). Once **ylist** is complete, it is inserted into the **saveset** SCL list using the **insert1** (insert list) function with a name of **YLIST** (line 33). The **YVARSELn** variables could have been stored directly in the **saveset** SCL list rather than grouped as a sublist. However, the number of **YVARSELn** variables can range from one to four. It is more difficult to determine the number of these variables if they all appear in the main list. By creating **ylist**, the **listlen** function can be used to easily obtain the number of variables in the sublist.

When the **RETURN** statement is encountered at the end of the **SAVELIST** section, control returns to the statement following the **link** command in the **SAVE** section. Here, the **saveset** SCL list is saved to the designated catalog using the **savelist** function (line 8). This function saves the SCL list (**saveset**) to the SLIST entry (**totalnam**). The **CATALOG** parameter causes the data to be saved to a catalog entry rather than an external file. Finally, the two SCL lists are deleted using the **dellist** function and an appropriate message is displayed (lines 9-11).

As mentioned earlier, it is convenient to also store object information in SLIST entries along with the **GRSEG** entries generated by the **GLOT** procedure. This allows the objects on the **FRAME** to be updated with appropriate information as the user scrolls through the **GRSEG** entries using the vertical scrollbar. When the push button object named **PLOT** is selected, the section of code labeled **PLOT** executes.

Storing Object Information in a Temporary SLIST Entry

```
041 PLOT:
042 * PLOT submits SAS code to generate a graph.
    This graph is then
043 * displayed in the SAS/GRAPH object and the
    objects' info is stored;
044 /* Plotting code goes here (NOT SHOWN) */
045 totalnam='WORK.MYSEG.'||curgrph||'.SLIST';
046 saveset=makelist();
047 ylist=makelist();
048 call notify('USERLIB','_SELECTED_',
    1,rownum);
```

```
049 call notify('USERLIB','_GET_TEXT_',
    rownum,rowtext);
050 saveset=insertc(saveset,rowtext,
    -1,'USERLIB');
051 call notify('USERDS','_SELECTED_',1,rownum);
052 call notify('USERDS','_GET_TEXT_',
    rownum,rowtext);
053 saveset=insertc(saveset,rowtext,
    -1,'USERDS');
054 call notify('APPEAR','_GET_TEXT_',apptxt);
055 saveset=insertc(saveset,apptxt,-1,'APPEAR');
056 link SAVELIST;
057 rc=savelist('CATALOG',totalnam,saveset);
058 rc=dellist(ylist,'Y');
059 rc=dellist(saveset,'Y');
060 RETURN;
```

The code that generates the plot is not shown; only the code relevant to the present discussion is shown. The **PLOT** code is essentially the same as that in the **SAVE** section. It differs in that three pieces of information are inserted into the **saveset** SCL list before linking the **SAVELIST** code (lines 48-55). They are the selected library name, data set name, and appearance (color or B&W).

Retrieving Object Information from an SLIST Entry

When a user wants to retrieve object information that has been stored previously using the **SAVE** push button, she can do so by selecting the **LOAD** push button. This causes the section of code labeled **LOAD** to execute.

Retrieving Object Information from a Permanent SLIST Entry

```
061 LOAD:
062 link GETFILE; /* GETFILE code NOT SHOWN */
063 if fname ^= _BLANK_ then do;
064     totalnam='GREPORT.GRAPH.'
        ||fname||'.SLIST';
065     loadset=makelist();
066     rc=filllist('CATALOG',totalnam,loadset);
067     if rc=0 then do;
068         link LOADLIST;
069         _msg_='Settings loaded from file '
            ||totalnam;
070     end;
071     rc=dellist(loadset);
072 end;
073 else do;
074     _msg_='Invalid filename. Settings not
        loaded.';
075 end;
076 RETURN;
```

First, the name of the SLIST entry is obtained from the user (line 62). Next, an SCL list named **loadset** is created (line 65). This list is filled with the information stored in the SLIST entry using the **filllist** function (line 66). This function copies the SLIST entry

(totalnam) to the SCL list (loadset). If filllist is successful, it will return a value of zero. Next, the section of code labeled LOADLIST is linked.

Extracting Object Information from an SCL List

```
081 LOADLIST:
082 GTITLE=getnitemc(loadset,'GTITLE');
083 GSUBTITL=getnitemc(loadset,'GSUBTITL');
084 call notify('XVAR','_UNSELECT_ALL_');
085 call notify('YVAR','_UNSELECT_ALL_');
086 call notify('XVAR','_SELECT_TEXT_',
              getnitemc(loadset,'XVARSEL'));
087 call notify('XVAR','_SET_TOPTXT_',
              getnitemc(loadset,'XVARSEL'));
088 ylist=getniteml(loadset,'YLIST');
089 numitems=listlen(ylist);
090 do i=1 to numitems;
091   call notify('YVAR','_SELECT_TEXT_',
               getnitemc(ylist,'YVARSEL' || i));
092   link YVAR;
093 end;
094 rc=dellist(ylist,'Y');
095 RETURN;
```

The LOADLIST routine extracts information from the loadset SCL list and passes this information to the appropriate objects on the frame. Named character information is obtained from an SCL list as follows (line 82):

```
GTITLE = getnitemc(loadset,'GTITLE');
```

This getnitemc (get named character item) function call returns the character item within the loadset SCL list with an item name of GTITLE. Again, the GTITLE variable is used directly in the code in lieu of using a call notify statement as follows:

```
call notify('GTITLE','_SET_TEXT_',
            getnitemc(loadset,'GTITLE'));
```

GSUBTITL is also assigned using the former method (line 83).

All selected items in the XVAR and YVAR objects are cleared by sending them an _UNSELECT_ALL_ message (lines 84-85), and then new selections are made based on the information in the SCL list. The list of dependent variables are referenced by creating an SCL list named ylist (line 88). The getniteml (get named list item) function call returns the SCL list identifier of the nested SCL list named YLIST from the loadset SCL list. Since the number of items in this list can vary, the list length is determined via the listlen function (line 89). Next, a do loop extracts the dependent variables in the ylist SCL list and selects these variable names in the YVAR object (lines 90-93). Finally, ylist is deleted (line 94).

When the RETURN statement is encountered at the end of the LOADLIST section, control returns to the statement following the link command in the LOAD section. Here, the loadset SCL list

is deleted and an appropriate message is displayed (lines 69-71).

When the vertical scrollbar is manipulated by the user, the GSCROLL section of code is executed.

Retrieving Object Information from a Temporary SLIST Entry

```
101 GSCROLL:
102 * GSCROLL updates the SAS/GRAPH object with
   a GRSEG entry
103 * and updates the objects' information with
   an SLIST entry;
104 call notify('GSCROLL','_GET_VALUE_',gnum);
105 if gnum=0 then curgrph='GPLOT';
106   else curgrph='GPLOT' || gnum;
107 call notify('GRAPH','_SET_GRAPH_',
              'WORK.MYSEG.' || curgrph || '.GRSEG');
108 call notify('OUTPUT','_SET_BORDER_TITLE_',
              'OUTPUT: ' || curgrph);
109 totalnam='WORK.MYSEG.' || curgrph || '.SLIST';
110 loadset=makelist();
111 rc=filllist('CATALOG',totalnam,loadset);
112 if rc=0 then do;
113   call notify('USERLIB','_SELECT_TEXT_',
               getnitemc(loadset,'USERLIB'));
114   link USERLIB;
115   call notify('USERDS','_UNSELECT_ALL_');
116   call notify('USERDS','_SELECT_TEXT_',
               getnitemc(loadset,'USERDS'));
117   link USERDS;
118   call notify('APPEAR','_GET_STATION_',
               getnitemc(loadset,'APPEAR'),row);
119   call notify('APPEAR','_ACTIVATE_',row);
120   link LOADLIST;
121 end;
122 rc=dellist(loadset);
123 RETURN;
```

A _GET_VALUE_ message is sent to the GSCROLL scrollbar object in order to obtain its current value (line 104). This scrollbar is designed to return a value ranging from zero to the number of GRSEG entries in the catalog minus one. (Notice the position of the thumb on the scrollbar in Figures 1 and 2, where GPLOT1 and GPLOT3 are shown from a catalog containing GPLOT through GPLOT3.) Thus the return value can be concatenated to the string 'GPLOT' to obtain the name of the GRSEG to display (except when the value is zero, in which case the name is just 'GPLOT', lines 105-106). Messages are then sent to the SAS/GRAPH output object and the OUTPUT container so that the appropriate GRSEG entry is displayed and the label for the container surrounding the SAS/GRAPH output object reflects the current graph name (lines 107-108). Next, the loadset SCL list is created and filled with the SLIST entry corresponding to the displayed GRSEG (lines 109-111). After successfully filling the SCL list, the extra information stored in the SLIST entry by the PLOT routine is extracted and used to adjust objects on the FRAME (lines 113-119). Finally, the LOADLIST routine is linked to extract the remaining information from the loadset SCL list before deleting it (lines 120-122).

Summary

SCL lists and SLIST catalog entries are very useful for storing information about FRAME objects that needs to be retrieved at a later time. SLIST entries can also be used in temporary catalogs to store information about other entries in the catalog. This article shows how SLIST entries can be stored in a catalog along with GRSEG entries. As the user scrolls through the catalog viewing different GRSEG entries, the corresponding SLIST entries can be

retrieved and FRAME objects updated to reflect the settings for the GRSEG being viewed. However, this method can be used for any application that stores multiple output files in a catalog and allows the user to scroll through the entries.

SAS, SAS/AF, and SAS/GRAPH are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

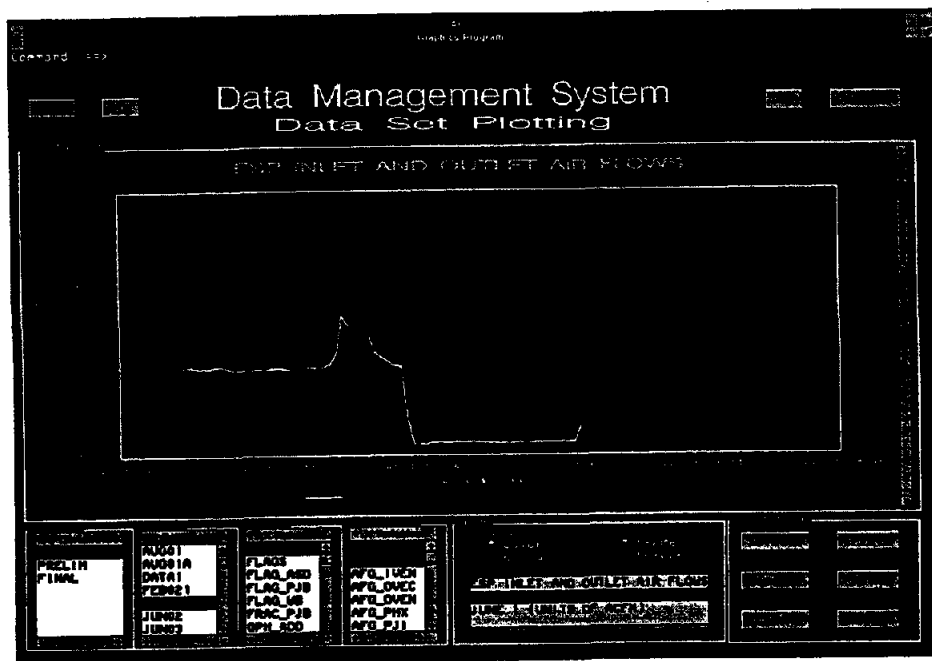


Figure 1. Second (GPLOT1) of Four Graphs

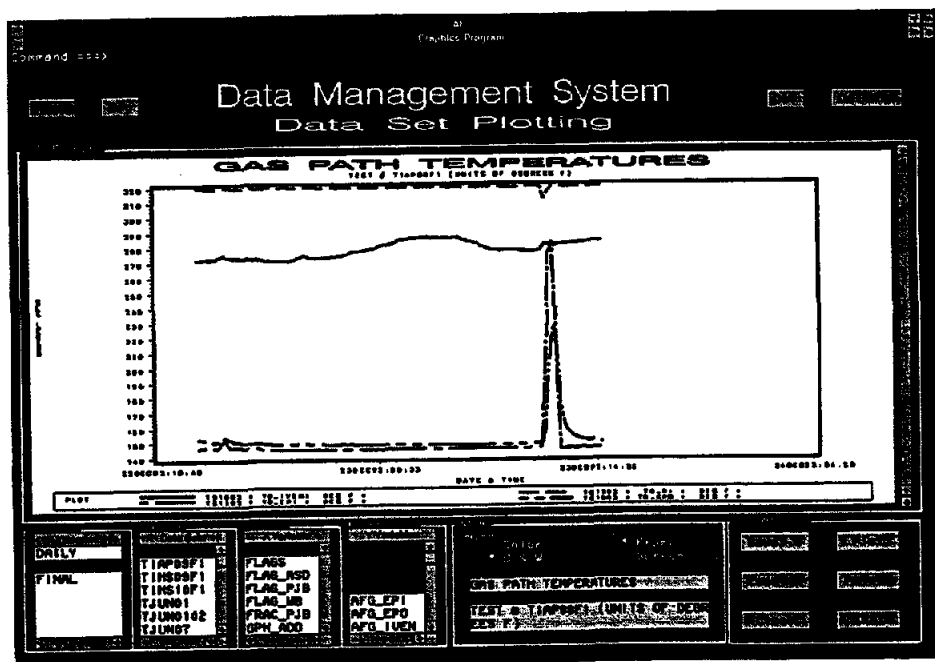


Figure 2. Fourth (GPLOT3) of Four Graphs