Paper 41-25

(Getting Even) Frame Your Base SAS® Programs

Andrew Rosenbaum, Trilogy Consulting, Kalamazoo, MI

ABSTRACT

Some programs are created with hard-coded values that need to be changed on a regular basis. This amounts to rewriting the program every time a change is made to one of the parameters. For example, a report of hours worked by employees for certain days. If the person requesting the report is not versed in the ways of SAS System programming, then a programmer must go into the program, find the parameters to be changed, make the changes, and run the program. Instead, the program can be written with variables in place of the hard-coded values. Any user can change the variables at run-time by way of a graphical user interface (GUI) also known as a FRAME. This frees the programmer to create programs instead of running them for others. This procedure is sometimes known as 'wrapping'. The base SAS program is 'wrapped' in a SAS/AF® program so that user interaction can be accomplished using visual objects (Text Entry boxes, Push Buttons, etc.) instead of re-writing the program each time a parameter needs to be changed.

INTRODUCTION

This paper is aimed at base SAS programmers with little or no experience with SAS/AF.

Assumptions:

- The SAS system 6.08 or higher
- Windows environment
- SAS/AF licensed and installed
- SAS/GRAPH® is needed for applications that use graphics objects.

NOTE: It is not necessary for the user of a SAS/AF application to have SAS/AF licensed. Only those who wish to develop or edit a SAS/AF application need to license SAS/AF.

THE PROBLEM

It's happened again! The boss wants you to run the weekly report. It's the same routine every week. Find the program. Change the start and end dates. Change a libname or data set name. Set a few other parameters. Submit the program. Get the output. Give it the boss. Oops! He gave you the wrong start date. Run the report again. Not very efficient, is it? A better way that would free up valuable programmer time and let the boss get the report he wants when he wants it would be to 'wrap' the base SAS program into a FRAME and let the boss run it for himself.

Here is a simple program that will be used to demonstrate the process of wrapping a base SAS program in a FRAME. It contains a data set that has information about employees, the date(s) that they worked, and the number of hours worked. Also, a base SAS program that is used to generate a simple report of hours worked for a given time period.

Incorporating the program into a SAS/AF application will allow any user to run this program and change the dates. Further modifications would allow the user to choose from different report styles, different sources of information, or to choose a subset of employees for which to report. The possibilities are endless!

Here is the data set WEEKDATA.EMPHOURS:

```
OBS    EMPNAME      DATE    HOURS
---    -------      ----    -----
  1    ASR         14703        8
  2    ASR         14704        7
  3    SMS         14705        4
  4    SMS         14706        5
  5    JKR         14707        7
  6    ASR         14710        6
  7    ASR         14711        8
  8    SMS         14712        7
  9    SMS         14713        9
 10    JKR         14714        9

/*----------------------------------------*/
```

This is the base SAS program that will be 'Framed':

```
data payday;
   set weekdata.emphours;
   *** Subset by date  ***;
   where date >= '09APR2000'd &
                 date <='15APR2000'd;
run;

*** Send output to file ***;
proc printto file='c:\reports\payday.txt' new;
run;

proc print data=payday;
  var empname date hours rate;
run;

proc printto;
run;

/*----------------------------------------*/
```

In this example the libname, the dates, the data set name, and the output file name have all been hard-coded into the program. To make the program more generic, these values can be represented by variables.

THE SOLUTION

In order to allow the user to select values for the variables at run-time, a visual interactive display needs to be created. It can be referred to in many ways: GUI (Graphical User Interface), FRAME, or front-end. The FRAME will contain objects that the user can type information into (Text Entry objects) and it will have buttons that the user can press to initiate action (Push Button objects). These objects are referred to as WIDGETS in Version 6 of the SAS System and as COMPONENTS in Version 8 of the SAS System.

In addition to the visual part of the application, there is SAS code 'behind' the FRAME that does many things. This code is known as SCL. (Version 6 definition is Screen Control Language – Version 8 definition is SAS COMPONENT LANGUAGE.) A section of the SCL code runs when the application is started. It also runs when anything is changed on the FRAME or when the user selects a Push Button. The SCL can obtain information from

the widgets and it can send information to a widget to be displayed.

The first task is to determine what parameters could the user be allowed to change:
- libname
- data set name
- start date
- end date
- destination of output report (file, printer, email)

(For simplicity's sake, only the data set name, start date, and end date will be used.)

Next, the FRAME needs to be designed that allows the user to make selections for each of the parameters. (A working knowledge of the available visual objects is a Good Thing.) For now, this discussion will use the Text Entry and the Push Button widgets.

Before starting, it will be necessary to understand the four-level naming convention that the SAS System uses:

libname.catalog.entry.type

LIBNAME is set prior to the start of work on the application. In the Windows environment, SAS catalogs are stored as files with an extension of .SC2. CATALOG contains the entries that go into a SAS/AF application. ENTRY is the name given to the entry. It can be any valid SAS name. TYPE is the type of entry: FRAME, SCL, IMAGE, HELP, CBT, etc. In this example, there will only be FRAME and SCL entries. (A SAS catalog can hold many other types of entries as well.) The FRAME entry holds the visual information. SCL entries hold the code that is used to manipulate the visual objects and the data. Therefore, SYSTEM.REPORTS.HOURS.FRAME is a FRAME entry called HOURS that resides in a catalog named REPORTS which is a file that is located in the directory which SYSTEM points to.

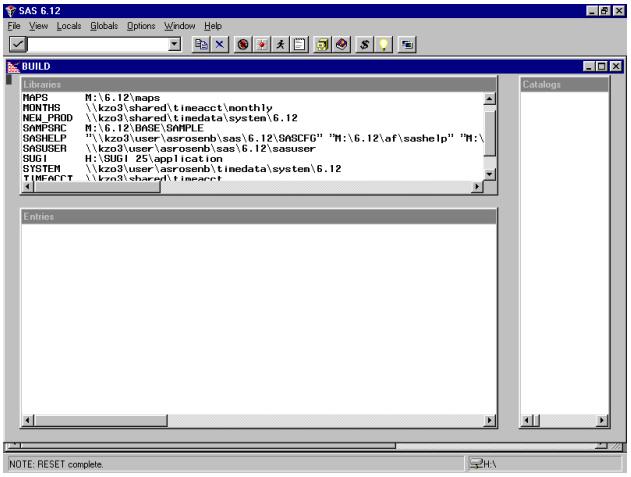
THE FRAME

Here are systematic instructions for creating a FRAME in version 6.12 of the SAS System for the sample program:

Assign the libname SYSTEM. A catalog called REPORTS will be created that will reside in SYSTEM and it will contain the FRAME and SCL entries for this application.

To enter the SAS/AF development area, type the word BUILD in the command box or on the command line. This will bring up the build window (figure 1), which contains three main areas. The first is called LIBRARIES and it contains the names of all of the existing libnames and the paths to them. When a libname is selected, the catalogs in that library will be displayed on the right side of the screen in the area called CATALOGS.

Click on the libname SYSTEM. Nothing significant happens because there are no catalogs in the SYSTEM library, yet. Go to the menu at the top of the screen and select FILE – NEW – CATALOG. Enter the name of the catalog (REPORTS) and select OK. The name REPORTS can now is seen in the CATALOGS area of the build window. When a catalog is selected, all of the entries within it are displayed in the ENTRIES area. REPORTS are highlighted since it is the only catalog in SYSTEM but there are no entries in it so nothing is displayed in the ENTRIES area, yet. Select FILE – NEW – ENTRY. Fill in the name of the FRAME (in this application it will be called HOURS) where it says ENTRY NAME. Make sure that the ENTRY TYPE says FRAME. (If it doesn't, then click on the down arrow and select FRAME from the pop-up list.) A blank FRAME should appear that says: BUILD: DISPLAY HOURS.FRAME in the title area.



<Fig 1- build window>

THE WIDGETS

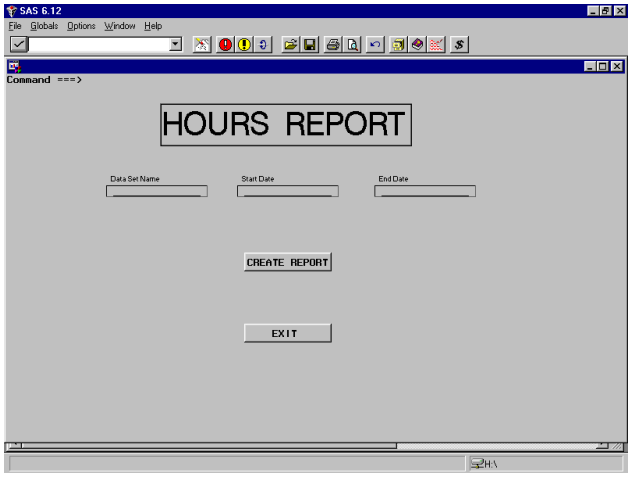
Next, populate the frame with the appropriate widgets. Create a Text Entry box for the data set name by right clicking anywhere in the FRAME and selecting MAKE from the pop-up list. Find TEXT ENTRY and select it. Press OK. After the pop-up window closes, move the cursor to any place in the frame where the Text Entry box is to appear. (It can be moved later.) Click once. A window opens that says TEXT ENTRY ATTRIBUTES. Change the name of the object from OBJ1 to DSNAME. Change the type to DSNAME. The type attribute allows the developer to specify what kind of data should be entered in the Text Entry box. When the user enters data, the Text Entry box automatically performs error checking. If the developer sets the type to NUM and the user enters character data, then a message would appear at the bottom of the screen alerting the user that incorrect data was entered. In this case, the DSNAME type means that SAS will check to see if a legitimate one- or two-level SAS data set name was entered. NOTE: DSNAME does not verify that the data set exists. The developer will have to do that using SCL code. Select OK from the bottom of the ATTRIBUTES screen.

Create another Text Entry box by repeating the previous procedure and call the new Text Entry box STRTDATE. Select NUM for the TYPE of data. Find the box that says, ADDITIONAL ATTRIBUTES and select INITIAL VALUE/FORMATS . . .
This will open another screen that has room on the right side to enter a format and an informat. Put MMDDYY10. (Or your favorite choice for a date format) in both boxes. Select OK for this box and then select OK again for the ATTRIBUTES box.

Create another Text Entry box by right clicking on the STRTDATE box and select COPY from the pop-up menu. Place the new Text Entry box anywhere on the screen and click once. Now select the new Text Entry box by clicking on it once and then right clicking on it. Select OBJECT ATTRIBUTES from the pop-up menu and change the name from OBJ1 to ENDDATE.

Now create two Push Buttons. One will say CREATE REPORT and one will say EXIT APPLICATION. Right click anywhere on the screen where there are not any objects. Select MAKE and then PUSH BUTTON. After placing the Push Button on the screen, change the name of the Push Button to CREATE. Change the label of the Push Button to CREATE REPORT. Select OK.
Create a new button in the same way or copy the CREATE button and set the name of the new button to EXIT. Change the label to EXIT APPLICATION. Find COMMAND PROCESSING under

additional attributes. Enter the word END in the box so that it says *EXECUTE SAS COMMANDS ON SELECTION.* This will execute the END command when the button is pushed and will allow the user to exit the application.



<Figure 2- finished frame>

THE CODE

Now that all of the visual objects are in place (figure 2), the SCL needs to be written to control the application. Find LOCALS – EDIT SCL SOURCE in order to open the text editor for SCL. You will see a blank screen that has line numbers down the left side. The SCL code is shown below. There are 3 labeled sections in the following code:

INIT – This section is run automatically when the FRAME is started. It is executed only once and usually contains instructions to initialize the FRAME. Variables can be initialized, lists created, visual attributes set, etc. In this case, a test data set is created within a SUBMIT block. SUBMIT blocks are used to run base SAS code within SCL. (See section labeled SUBMIT BLOCKS.)

CREATE – This is the name of a Push Button widget from the FRAME. Whenever a widget is modified, its labeled section is run first. In this case, whenever the user clicks on the CREATE button, the labeled section CREATE is run. Some error checking is done and then the code for the report is submitted to SAS. (Note: Since the user is entering information, error checking should be done. For example: did the user enter both the start date and end date? Are they legitimate dates? Does the data set exist? Etc. However, not all possible error checking is done in this example.)

TERM – This section is run automatically when the FRAME ends. Usually, closing and/or deleting data sets, deleting SCL lists, save operations, and other clean-up operations are performed here. In this example, the test data set is deleted.

```
/*=============================================
Demonstration program: Generate report of
employee hours for a user-specified time period
from a user-specified data set.
       SAS Version 6.12 --- Windows9x
=============================================*/
INIT:
 /* CREATE DEMO DATA SET */
 submit continue;
  libname demo 'c:\SUGI demo';
  data demo.hours;
   input empname $3. +1 date mmddyy10. +1 hours;
   cards;
ASR 04/03/2000 8
ASR 04/04/2000 7
SMS 04/05/2000 4
SMS 04/06/2000 5
JKR 04/07/2000 7
ASR 04/10/2000 6
ASR 04/11/2000 8
SMS 04/12/2000 7
SMS 04/13/2000 9
JKR 04/14/2000 9
;
  run;
 endsubmit;
return;

/*-----------------------------------------*/

CREATE:
  /* CHECK THAT THE DATA SET DSNAME EXISTS. */
  /* IF NOT, THEN ISSUE AN ERROR MESSAGE.   */
  rc=exist(dsname);
  if rc = 0 then do;
    /* DATA SET DOES NOT EXIST */
    _msg_='Data set does not exist';
    return;
  end;
  /* CHECK TO BE SURE THAT ENDDATE>STARTDATE */
  /* If IT ISN'T, ISSUE ERROR TO THE SCREEN. */
  if enddate < strtdate then do;
    _msg_='End Date must be after Start Date';
    return;
  end;

  /* RUN REPORT */
  submit continue;
    data payday;
      set &DSNAME;
      **SUBSET BY DATE AND EMPLOYEE NAME(S)**;
      where date >= &STRTDATE and date
                                 <=&ENDDATE;
    run;

    title 'DEMO REPORT: HOURS FOR ACME SAS'
                           ' CONSULTING COMPANY';
    proc print data=payday noobs;
      format date mmddyy10.;
      var empname date hours;
    run;

endsubmit;

  /* SEND MESSAGE THAT REPORT HAS BEEN RUN */
  _msg_='Report has been generated';
  alarm;
return;

/*-----------------------------------------*/

TERM:
  /* DELETE TEST DATA SET */
  rc=delete('demo.hours');
return;

/*-----------------E N D--------------------*/
```

To compile the code, select LOCALS – COMPILE. If there are no errors, the message, "Code generated for HOURS.FRAME" will appear on the message line at the bottom of the screen.

To run the application, close the SCL window and the FRAME. They will automatically be saved. In the command window, type the following:

```
AF C=SYSTEM.REPORTS.HOURS.FRAME
```

After pressing ENTER the application will run.

SUBMIT BLOCKS

In order to allow base SAS code to be run from SCL, it must be placed into a SUBMIT block. Any valid base SAS code can be placed within a SUBMIT block and multiple data steps and procedures may be used as well as MACRO code. There are a few things to take note of:

1. Code within a SUBMIT block is not checked at compile time. That means that if there is a syntax error within a SUBMIT block, no warning or error message will appear until the program is executed.
2. The user will not see any error messages issued during the execution of the SUBMIT block. Any messages issued will be sent to the log. (See the section on ERROR CHECKING later in this paper.)
3. There are two ways of executing a SAS/AF program: The first is to issue the AF command with the name of the entry to execute.
   (AF CATALOG=SYSTEM.REPORT.HOURS.FRAME)
   The second is to use TESTAF mode. TESTAF is the faster method to use during the development of an application. However, SUBMIT blocks are not executed in TESTAF mode. The programmer must use the AF command to execute an application that contains SUBMIT blocks.

Before the first line of base SAS code, the keyword SUBMIT is used. After the last line of base SAS code, the keyword ENDSUBMIT is used. This defines a SUBMIT block. The keyword CONTINUE is used to cause the SUBMIT block code to be executed immediately.

The parameters that were chosen earlier, STRTDATE, ENDDATE, and DSNAME will now become variables. Just place an ampersand in front of the variable name. This does NOT turn the variables into MACRO variables. However, the result is similar. Whenever an ampersand is found within a SUBMIT block, SAS looks for an SCL variable of the same name. If one is found then the SCL variable value is substituted. The code is then submitted just as one would submit it from the program editor.
If an SCL variable is not found, then the code is submitted and the variable is treated as a MACRO variable.

Where did the values for STRTDATE, ENDDATE, and DSNAME come from? Notice that the Text Entry widgets have the same names. When the user fills in DEMO.HOURS in the Text Entry widget called DSNAME, the SCL variable called DSNAME now has the value DEMO.HOURS. This value is then substituted for &DSNAME in the SUBMIT block.

There is an alternate way to run base SAS code from an SCL entry:

Leave the base SAS program in an external file and use %INC in the submit block:

```
submit continue;
  %inc 'C:\SUGI demo\hours report.sas';
endsubmit;
```

The second method allows changes to be made to the base SAS code without recompiling the FRAME code. In addition, someone that doesn't know how to access SCL code could modify the base SAS code. This might be a benefit or not, it depends on your situation.

ERROR CHECKING

As mentioned earlier, messages are sent to the log if an error or warning is generated during execution of a SUBMIT block. How will the user know if an error has occurred? The error must be trapped and then it can be reported to the user. The author uses the automatic macro variable SYSERR to detect errors and warnings within a SUBMIT block. SYSERR is 0 if a data step or procedure ran without errors or warnings. It is 4 when there is a warning. Anything else signifies an error. The following is an example if there is a single data step within a SUBMIT block:

```
submit continue;
   .
   .
   (data step or procedure)
   .
   .
endsubmit;

rc=symgetn('syserr');
if rc = 4 then put 'WARNING';
else if rc ^= 0 then put 'ERROR';
else put 'OK';
```

If multiple data steps or procedures are used then a different approach must be used because SYSERR is reset before a new data step or procedure is run. For example, the first data step may have an error and the second data step does not. SYSERR will be 0 after the second data step. The error from the first data step will not be trapped. There are two solutions to this problem:

1. Separate each data step and procedure into its own SUBMIT block and test SYSERR after each SUBMIT block.
2. Test SYSERR after each data step and procedure within the SUBMIT block and store any errors or warnings in a new macro variable. Then this new macro variable can be checked after the SUBMIT block for errors or warnings.

CONCLUSION

This paper has demonstrated the usefulness of SUBMIT blocks within SCL and has hopefully served as a gentle introduction to SAS/AF. A SUBMIT block is a powerful tool to extend the use of base SAS programs into a SAS/AF application.

An important attribute of good SCL programming is a consistent and easy to read style. The author presents the following as a suggested guideline. Feel free to develop and use your own set of standards.

- Indent two or three spaces.
- Use color:
  - Regular code = black
  - SUBMIT blocks = blue
  - Comments = green
  - Code that requires attention = orange
- Labeled sections are capitalized.
- Thorough description of program at top.
- Liberal use of comments.
- Write code that is easy to read. Remember: YOU may be the person who has to change this program next year!

Another element of programming style is variable naming. A common approach among programmers is to use one or more lowercase letters to prefix a variable name. This prefix alerts the reader to how the variable is used. For example, a character

variable is preceded by the letter 'c' as in **`cLastName.`**
A numeric variable is preceded with an 'n' as in **`nPaperLength.`**
Take note that SCL non-window variable names can be up to 32 characters long. Take advantage of this and give your variables useful names. Just remember that you may have to type them repeatedly so don't make them too long.

Since this paper only touches on the basics of SAS/AF development, further reading and/or formal instruction are recommended. See the REFERENCES section for recommended reading.

REFERENCES

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

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

SAS Institute Inc. (1995), Building SCL Applications Using FRAME Entries Course Notes, Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The author would like to thank the following people for their assistance:

Charles Bininger
Dean Clous
Manuel Rosenbaum
Brian Varney

CONTACT INFORMATION

Contact the author at:

Andrew Rosenbaum
Trilogy Consulting
5278 Lovers Lane
Kalamazoo, MI 49002
U.S.A.
Phone: 616.344.4100
FAX: 616.344.6849
Email: Andrew_Rosenbaum@trilogyusa.com