A Maintenance-Free Menu Driven Closure System

by Stephen M. Noga,
Rho, Inc.

Introduction

As a clinical trial nears closure, a series of data validation programs are run, sometimes individually, and sometimes in combination with one another. This paper describes a menu driven closure system that was developed using SAS® Frame, and takes advantage of what were once exclusively SCL functions, but which are now available in the data step. This menu system is maintenance-free because you can add new closure programs to it without ever having to update the menu program.

Although this system was developed for use in a CRO environment, I demonstrate how it can be adapted for use in any organization. The examples in this paper are for a fictitious company, Joe's Garage, and any similarity between an actual company and this one is purely coincidental.

The Concept

The SAS department at Joe's Garage is composed of many creative programmers. They are constantly writing newer and better programs. One of their tasks, in conjunction with the car maintenance managers, is to write a series of programs that verify an automobile is up to Joe's standards before it leaves the shop.

So, following the normal organizational flow, the following three steps occur: the car managers decide what they want the programs to check, and then relay this information to the SAS department; the programmers write the programs to accomplish these checks; and the car processing department runs these programs from an interactive menu which has descriptions of the closure programs.

This system works great until the car managers, in their never-ending quest for quality-assurance, develop ideas for new checks. Then they present these requests to the programmers who write the new programs, and upon completion, inform the car processing people that they can now run these new checks. Of course when the car processing people try to run some of these new checks, they can't see them on the menu because nobody updated the closure menu, and all the SAS programmers who know where the menu code was located were on vacation, or at a conference, or otherwise indisposed. Another possible complication is that the menu may have descriptions for a number of checks that are no longer needed (i.e., cars no longer need a crank in front to start them, so no need to check that a crank is included with the car).

The Solution

The garage manager decided that a menu system was needed that would be maintenance-free once the initial development was completed. In other words, new programs can be added to the closure menu and obsolete programs can be deleted without modifying the menu program, and whenever the car processing people brought up the closure menu, the information contained in it would be current. And so it happened that this system was created and Joe's Garage was a harmonious work-place again.

Maintenance-Free Code: Step 1

This menu system is created by a three step process.

- Gather current information
- Select programs to run
- Run selected programs

In the first step, a program is run which gathers information about which closure programs are currently considered in 'good standing'. To accomplish this simple task, three constraints were applied: all current closure programs would reside in the same, designated directory (i.e., R:\base\closure); all closure programs would have the same file extension; and the top line of each closure program would contain a brief description of what the program was designed to do. This description is enclosed in ^^ so the program is able to distinguish the description from comments or anything else.

```
**Verify Headlights And High Beams Work Properly*;
```

Because of these three constraints, all the system needs to do to construct a menu is look in one directory and get the names and associated descriptions of the programs residing in it. The code to accomplish this task has been made relatively short and sweet thanks to some SCL functions which were made available to the data step in version 6.12.

A temporary data set is created that contains three variables: the program name (FILE); the program description (DESC); and a flag variable (CHKBOX) which will be used in the FRAME application.

```
*****************************************
 Get Description Of Current Closure Programs
*****************************************;
DATA progs (keep=file desc chkbox);
 length txt $89 desc $85 ;
 retain chkbox . ;
 txt = ' ';

  fn = filename("fileref","R:\base\closure\sesug98");
  dirid = dopen("fileref");

  filecnt = dnum(dirid);
  do i = 1 to filecnt;
    file = dread(dirid,i);
    if index(file,'.PGM') > 0 then do;
      fn= filename("fileref2",
                   "R:\base\closure\sesug98\"||file);
      fid = fopen("fileref2");
      if fread(fid) = 0 then do ;
        txt = ' ';
        rc = fget(fid,txt,89);
        desc = substr(txt,
                   index(txt,'^')+1,
                   index(substr(txt,index(txt,'^')+1),'^')-1);
        output ;
      fcl = fclose(fid);
      end;
    end;
  end;
RUN:
```

A file reference id (FILEREF) is assigned with the filename function which points to the directory where the closure programs reside, and then the dopen function 'opens' this directory (information can be gathered from it) and a numeric id is assigned to the directory. The dnum function gets the number of files within this directory.

Now that the directory is opened, and the program knows how many files are in it, then all the code needs to do is loop through the files and save the file name and description. The dread function gets the filename associated with file #i . The code then checks to see if the filename has the required extension (.PGM). This is a necessary check since files other than the closure programs may be in the directory. In fact, the function dnum considers ., .., and any other directories that are contained in the closure directory to be files. If the filename has the required extension then get the description.

A file reference id is assigned to the file this time, and the file is opened with the fopen function (same idea as dopen). Read the first record (line #1) from the file into the file data buffer with the fread function, and if this was successful (a 0 is returned), then copy the first 89 characters into the data step variable TXT. Now strip out the file description (DESC) from TXT, output the observation, and close the file with the fclose function.

Once the program has processed all of the files in the closure directory, the data set PROGS is ready to be used by step two of the closure system.

Maintenance-Free Code: Step 2

The second step is to use the information gathered in step one to construct an interactive menu with the current closure descriptions displayed. Once the data set PROGS has been created, call the FRAME application using PROC DISPLAY.

```
**********************************************
 Call FRAME For User To Select Closure Programs
**********************************************;
PROC display cat=dmscat.closure.select.frame;
RUN;
```

This produces **Figure 1.** As you can see, descriptions of the current closure programs are displayed, along with three push buttons (for demonstration purposes, two of the selections have been checked to show the reader what the menu would look like if one or more of the areas had been selected).

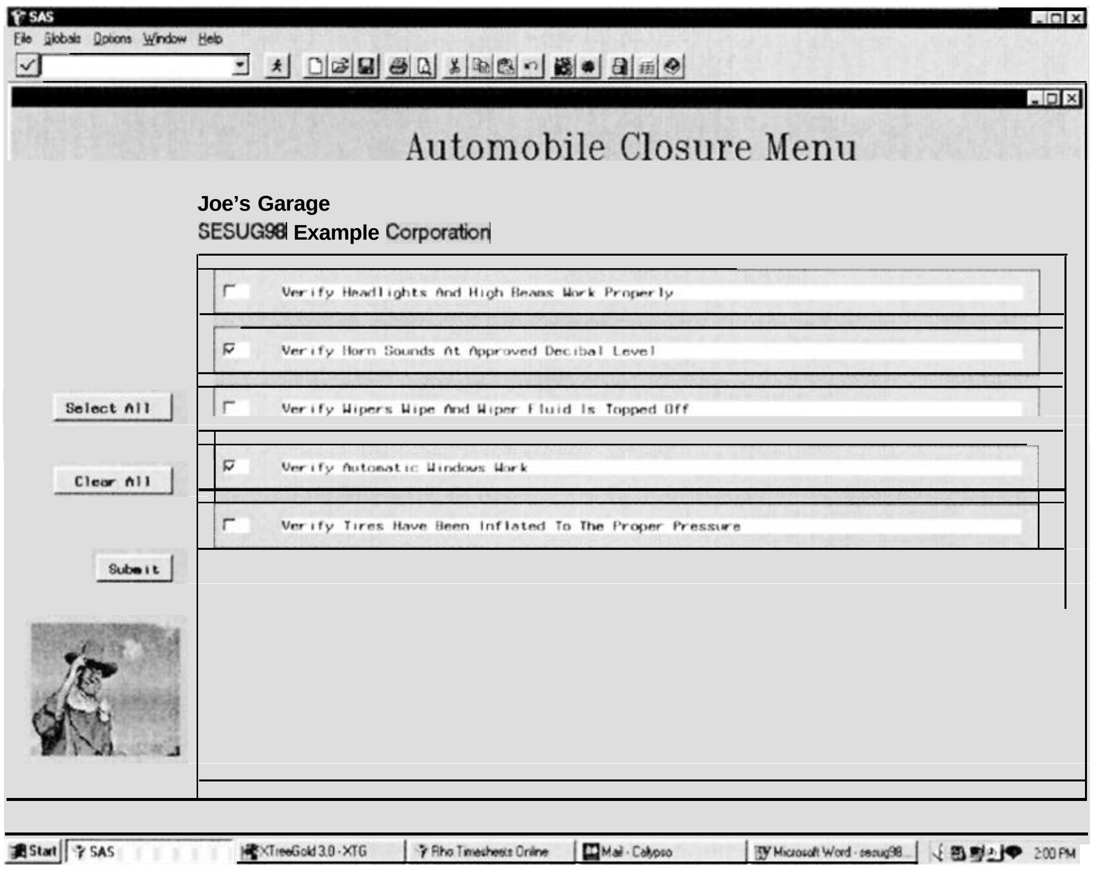
This menu was created using FRAME. There are only four main components to this application (one table of descriptions and three push buttons). The table is an Extended Table and actually contains three pieces within it: the container box (the rectangle that the checkbox and description are contained in); a Checkbox (named CHKBOX - remember that the data set PROGS has a variable named CHKBOX with all values set to missing); and a Text Entry Field (named DESC - remember that the data set PROGS has a variable named DESC whose values are the descriptions of the current closure programs). The Select All (named MARKALL), Clear All (named CLEARALL), and Submit (named SUBMIT) buttons are Push Buttons. The "Automobile Closure Menu", "Joe's Garage", and "SESUG98 Example Corporation" are all Graphic Text fields, while the image in the lower left corner is an Image field.

The SCL code which runs behind this frame follows.

```
INIT:
  * enable processing - custom commands;
  control enter;
```

The following code takes away the ability for someone to press a function key and have something happen that the programmer did not intend to happen.

(Figure 1)


SAS
File Globals Options Window Help
Automobile Closure Menu
Joe's Garage
SESUG98 Example Corporation
Verify Headlights And High Beams Work Properly
Verify Horn Sounds At Approved Decibal Level
Verify Wipers Wipe And Wiper Fluid Is Topped Off
Verify Automatic Windows Work
Verify Tires Have Been Inflated To The Proper Pressure
Select All
Clear All
Submit
Start
SAS
XTreeGold 3.0 - XTG
Rho Timesheets Online
Mail - Calypso
Microsoft Word - sesug98
2:00 PM

```
. Store current function key defs;
  array pfkeys{53} $40;
  do i=1 to 53;
     pfkeys(i) = getfkey(fkeyname(i));
     call setfkey (fkeyname(i),'');
  end ;

. Define screen specific fnctn keys;
  call setfkey (fkeyname(3),'END');

* Open transactions dataset;
  link open;

call notify('progs' , '_cursor_') ;
RETURN ;

MAIN:
RETURN ;

TERM:
   * Close data set:
   rc=close(progid);
```

```
   * Restore function keys ;
   do i=1 to 53;
      call setfkey (fkeyname(i),pfkeys{i});
   end ;
RETURN ;
```

Open the data set PROGS created in step one which holds the closure program descriptions

```
OPEN :
   progid = open('work.progs','u');
   if (progid eq 0) then do;
      * open failed;
      link alarm;
      _msg_=sysmsg();
      put _msg_=;
      refresh ;
      return;
   end ;

   * Fill the screen with data set info:
   call set(progid);
RETURN ;
```

Get the data from data set PROGS

```
GETPROGS:
  rc = fetchobs(progid,_currow_);
  if rc ne 0 then do;
    call notify ('progs','_endtable_');
    if (rc ne -1) then do;
      link alarm;
      _msg_ = 'Error reading data set:
'|| sysmsg();
    end;
  end;
RETURN;
```

Update the data in data set PROGS

```
PUTPROGS:
  chkhold = chkbox;
  rc = fetchobs(progid,_currow_);
  chkbox = chkhold;
  rc = update(progid);
  if (rc ne 0) then do;
    link alarm;
    _msg_=sysmsg();
    refresh;
    put _msg_;
     return;
  end;
RETURN;
```

If the Select All (named MARKALL in it's attribute window) button is pressed, then set variable CHKBOX to 1 for all observations in PROG

```
MARKALL:
  call notify
('progs','_get_maxrow_',maxr);
  do i = 1 to -(maxr);
    rc = fetchobs(progid,i);
    chkbox = 1;
    call notify
('progs','_select_row_',i);
    rc = update(progid);
    if (rc ne 0) then do;
      link alarm;
      _msg_=sysmsg();
      refresh;
      put _msg_;
      return;
    end;
  end;
  call notify('progs','_refresh_');
RETURN;
```

If the Clear All (named CLEARALL in it's attribute window) button is pressed, then set variable CHKBOX to 0 for all observations in PROG

```
CLEARALL:
  call
notify('progs','_get_maxrow_',maxr);
  call notify('progs','_unselect_all_');
  do i = 1 to -(maxr);
    rc = fetchobs(progid,i);
    chkbox = 0;
    rc = update(progid);
    if (rc ne 0) then do;
      link alarm;
      _msg_=sysmsg();
      refresh;
      put _msg_;
      return;
    end;
  end;
  call notify('progs','_refresh_');
RETURN;

ALARM:
  call sound(523,260);
  call sound(423,460);
RETURN;
```

The way the menu is setup, the user can select any or all closure programs to run by either selecting each one individually, or pressing the Select All button. If the user made a mistake, Clear All resets all the check boxes to not selected. In this example, there are only five closure programs. If your company had 200 closure programs, all 200 would be displayed and the user would have the ability to scroll down the list.

Processing: Step 3

After the user presses the Submit button (the value on selection attribute is set to END) on the menu, the FRAME application closes and control is passed back to the program which called it. The last step in this closure system is only running those programs that the user wanted to run. The following code accomplishes that task using a combination of call symputs and call executes.

```
********************************#*xx***********
 Only Run Closure Programs Which Were Selected
*********************************.a**********;
DATA _null_;
 set  progs;
if chkbox = 1 then do;
  call symput('filen',trim(left(file)));
  call execute('DM "log; clear;";');
  call execute('data _null_;
                  put /"&filen will be called." ; run;');
  call execute('DM "log; print
file=R:\base\progoutUog\CLOSE.LOG  form=PLP
append;";');
  call execute('%include
"R:\base\closure\sesug98\&filen" / nosource2;');
end;
RUN;

**********
 Close SAS
**********;
ENDSAS;
```

Conclusion

My goal in writing this paper was to demonstrate to the reader a real life SAS FRAME application in a generic way so that he/she could take some of the ideas presented in this paper and apply them to a situation at his/her workplace. The SCL functions which are now available in BASE offer an additional set of tools to accomplish this.

Acknowledgements

I would like to thank Caroline Bahler for her series of papers from the SESUG '97 Proceedings. This series of three papers introduced me to the SCL functions that are now available in BASE.

References

Bahler, Caroline (1997), "New Data Step Functions in SAS 6.12 - Part I - Data Set Functions", SESUG '97 Conference Proceedings.

Bahler, Caroline (1997), "New Data Step Functions in SAS 6.12 - Part II - External File Functions", SESUG '97 Conference Proceedings.

Bahler, Caroline (1997), "New Data Step Functions in SAS 6.12 - Part III - Special Functions", SESUG '97 Conference Proceedings.

Author Contact

Stephen M. Noga
Manager, Statistical Programming
Rho, Inc
121 S. Estes Drive, Suite 100
Chapel Hill, NC 275 14
(919) 932-6500 x 235
snoga@rhoworld.com