

Creating a Dynamic Menuing System using SAS Frames

Dee Burgett, Tandy Information Services, Fort Worth, TX

ABSTRACT

The main menu is probably the single most important application in a company. If IT doesn't work, nothing works. Therefore, it is important to design a menuing system that can keep up with the company's changing needs. This paper addresses creating a menuing system that personalizes for each user, uses only one SAS frame, and doesn't require a copy of each user. This menu uses a "shell" frame that dynamically populates the icons with bitmaps, title, and call reference, based on data from a SAS data set. Access to an application is checked by allocating a libname set by security. The menu is controlled by SCL. Statistics are stored for user tracking. It uses flags to generate (more) and (goback) buttons. The menu is recursive with no nesting involved. This system also checks for application availability.

SYSTEM

This system was created on a Digital Alpha AXP machine running SAS for Open VMS security checking against a mainframe MVS SAS system.

DISPLAYING THE MENU

The menu that displays is unique for each user. There are no gaps where icons are hidden because the user does not have access. The (more) and (goback) button display only for those users with multiple screen access. This display looks clean and fresh for everyone. The statistics are captured behind the scenes, transparent to the user. Examples of its display are shown in Figure 1 and 2.

SHELL

A SAS frame was created and used as the shell. This frame, Figure 3, has a graphic label displaying the Main Menu name. Eight icons were created using generic bitmaps and named icon1, icon2, etc. through icon8. An exit push button was added. This produced a shell that can be used for a variety of menu levels.

MENU DATASET

A dataset was created which stores all the required information needed to start an application. Included were the icon number, icon label, label position, libname for security, application call name, and the level of menu at which this application resided. The level of menu indicates if the data goes on the main menu or a subsequent lower menu called by the main menu, such as the system maintenance menu.

This dataset is copied to the users work file immediately upon entering the init section of the menu, then deallocated. This allows for the updating of the dataset by the administrator, thus allowing new applications to be added during the day.

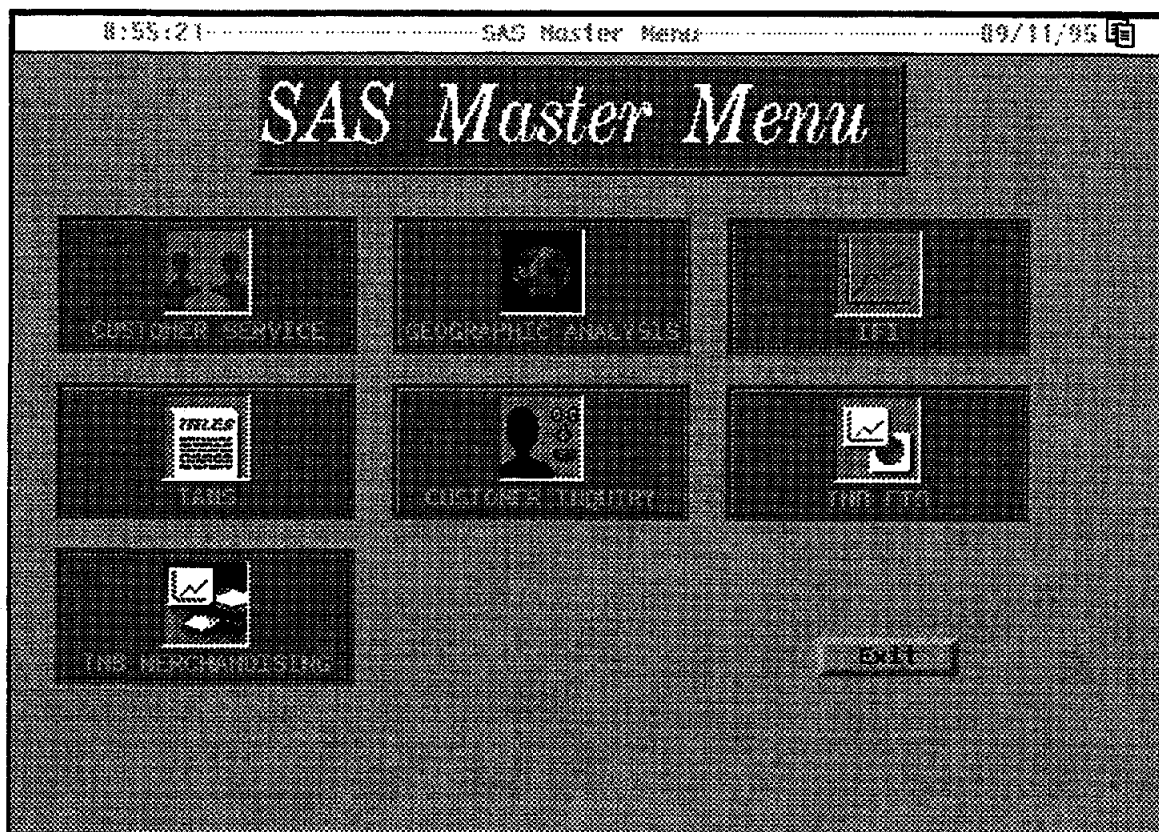


Figure 1

Figure2 - User with multiple screen access





MAINFRAME PASSWORD CHECK

TCP/IP client server applications require passing the mainframe password before connecting to the MVS. Users were set up with a generic name and password, controlled by security administration, to connect to the MVS system. This removed the need for modifying the mainframe password each month when it changed. The name and password were stored in a security dataset that was only accessed for the users. Programmers had the password checked. This system stored the mainframe password in the SASUSER file. Each month when the mainframe password changed, the stored password no longer matched and the programmers are prompted with a frame to change the stored password to the new mainframe password.

STATISTICS DATASET

A separate location was created to capture the usage of each user. Captured statistics were stored in a separate dataset from the menuing information, adding a new member each month. To avoid having to physically create a new member each month, the system checks to see if the current month member exists. If it does, the statistics, along with the userid, is appended to the file. However, if the member does not exist a new one has to be added. The system first checks to see if the libref exists, by checking for the return code from:

```
RC = LIBREF('LOGFILE');
```

If the reference does not exist the coding allocates the libname using the concurrent engine provided by Open VMS.

```
RC = LIBNAME('LOGFILE', '$DAT701:[PROD_USAGE]', 'CONCUR');
```

Immediately following this check for allocation is the check for the current month member. This is checked by the following code where toddate is today's month year as in SEP95:

```
LNAME = "LOGFILE." || TODDATE;
```

If the file does not exist, the system creates the new file with proper authority for all the other users. It uses a model dataset with the fields predefined. Since call execcmdi, is used the file is immediately created. The coding is as follows:

```
CALL NEW(LNAME, 'LOGFILE.MODEL', 0, 'N');  
SHOWIT = " X " || "SET FILE/PROT=(G:RWED,W:WRED) " ||  
         " $DAT701:[PROD_USAGE]" || TODDATE || ".**";  
CALL EXECCMDI(SHOWIT);
```

The statistics are appended to the correct month data for use by management.

SECURITY

Security personnel set up controlled datasets on the MVS side and granted access to the appropriate users. New users, or changes to existing users access is then easily modified

by the security department. The menuing system attempts to allocate a libname statement to these datasets. If a good return code is returned then this option is added to the users menu. If a bad return code is returned, the user does not have access and this icon will not display for the user.

Building the libname statement for the return code check uses the information from the dataset where all the menuing information is stored.

BUILDING THE FRAME

Arrays must be defined to keep track of the implied nested screen, the number of icons, and the statistics. SCL arrays are defined the same as base SAS arrays.

```
ARRAY ICONN{*} $ icon1-icon8;
```

The system appears to the user to have an infinite number of iterations that the screen can be redisplayed. However, the application does in fact have a finite number built into the array, in this instance 8, a number that the user should not reach. When the user clicks the (more) button the same screen rebuilds and displays. The screen counter increments based on whether the user select, more - plus or goback - minus. There is no need for multiple copies or nesting. This is achieved by use of the arrays and flag variables as described in this paper.

Variables are populated with the key data representing the first icon to be displayed on that screen. For instance, user A first has access to the application on row number 3. Screen 1 is displayed and the arrayed variable SROW{1} is 3. The pass number is 1 because this is the first screen to be displayed. The user wants to see what is on the second screen and presses the more icon. The user had access to applications listed on rows 3, 4, 5, 7, 8, 9, 10, 11, and 12. The user did not have access to the application on row 6. The system adds the (more) button if the user has access to at least 1 application past the last one displayed on the screen. The user presses the (more) icon and the variables SROW{2} is 12, the starting row of the second screen display. SROW{1} remains 3.

The maximum number of icons that can be displayed on a screen is dependent on which screen is displayed. The first screen has no need for a "goback" button so 8 icons are available for display. Subsequent screens need the addition of the "goback" button so 7 are available.

The next step in building the screen is the fetching of the rows in the menufile. This process uses the fetchobs function of SCL. Rownum is incremented after the security is checked.

The system checks to see if the user has access to the SAS libname. The libname statement is built based on the information returned from the table as shown below:

```
RC = FETCHOBS(MENFILE,ROWNUM);  
IF (RC = 0) THEN  
DO;
```

```

LNAME = '$SYSTEM:[PROD_SECURITY.';
LNAME = LNAME||IC_NM
LNAME = LNAME||)';
TESTRC = LIBNAME("TESTSEC",LNAME,"ACCESS=READONLY");

```

If the user has access the variable testrc is 0. If the user does not have access a number other than 0 is returned. If the user has access, the following code is performed and the icon is populated with that applications information and displayed to the user when the screen is refreshed. The mainframe password is passed along as the passnm variable as shown in figure 3.

Figure 3

```

IF (TESTRC = 0) THEN
DO;
  X = X + 1;
  PASSNM{X} = LOGIN||". "||PASS;
  IF (X < MAXNUM) THEN
  DO;
    ICONX = 'ICON'||X;
    CALL NOTIFY(ICONX,'_SET_ICON_',IC_NUM);
    CALL NOTIFY(ICONX,'_SET_LABEL_',IC_LABEL);
    CALL NOTIFY(ICONX,'_SET_LABEL_POSITION_IC_POS);
    CALL NOTIFY(ICONX,'_UNHIDE_');
    CDISPA{X} = CDISP_RF;
    CDISPB{X} = CDISP_NM;
  END;

```

This coding sets the icon number, label and label position. It tells the system to unhide this icon and display it. The CDISPA{X} variable if the application name as recognized by the system. The CDISPB{X} variable is the name of the application used in the call display statement.

Adding the (more) and (goback) icons coding is added just after the last end statement in figure 3, as shown in figure 4.

Figure 4

```

ELSE IF (PASSNO = 1) THEN
DO;
  ICONX = 'ICON'||X;
  CALL NOTIFY(ICONX,'_SET_ICON_',164);
  CALL NOTIFY(ICONX,'_SET_LABEL_', 'MORE');
  CALL NOTIFY(ICONX,'_SET_LABEL_POSITION_'_B');
  CALL NOTIFY(ICONX,'_UNHIDE_');
  PASSCK = 'NO';
END;
ELSE IF (PASSNO > 1) THEN
DO;
  ICONX = 'ICON'||X;

```

```

CALL NOTIFY(ICONX,'_SET_ICON_',164);
CALL NOTIFY(ICONX,'_SET_LABEL_', 'MORE');
CALL NOTIFY(ICONX,'_SET_LABEL_POSITION_'B');
CALL NOTIFY(ICONX,'_UNHIDE_');
X = X + 1;
ICONX = 'ICON'||X;
CALL NOTIFY(ICONX,'_SET_ICON_',116);
CALL NOTIFY(ICONX,'_SET_LABEL_', 'GOBACK');
CALL NOTIFY(ICONX,'_SET_LABEL_POSITION_'B');
CALL NOTIFY(ICONX,'_UNHIDE_');
PASSCK = 'NO';
END;
END;

```

Prior to the screen displaying, the system checks to see if this user has access to only one application. If he/she has just the one, the system automatically links to the label for icon1 as shown in figure 5.

Figure 5

```

PASSCK = 'NO';
RC = FETCHOBS(MENFILE,1);
IF (X = 1) AND (PASSNO = 1) THEN
DO;
    LINK ICON1;
    CALL EXECCMD('END');
    RETURN;
END;

```

The SCL code contains labels for the 8 generic icons. The labels are identical except the numbers refer to the number of the icon. Using these allows the program to display any icon that user has access to and standardizes the place for statistics capturing. The coding for this label, shown in figure 6, links to another section called forback. This section gets the label value, checking to see if the (goback) or (more) icons have been selected, and increments the flags accordingly. See figure 7 for this coding.

Other labels are linked in the icon coding. For those applications that access mainframe data from the MVS system the value of TCPSEC must contain the userid and the mainframe password. A call display to a program is added in this example for a check of the application availability, in this case APPLTEST.SCL. Passed to this call display, is the application name, cdispa#, and a display application flag. Since a user can stay in an application for an extended period of time, the system is designed to check the availability when he/she exits the application.

If new messages are available, for the main menu or the application, the system displays this on the screen. The availability coding displays a message indicating this. If maintenance is being performed on an application and the application is temporarily off

line, the availability coding displays a message indicating this. The user is returned to the main menu and may continue in some other capacity.

A link to the statistics section is done before and after the execution of the application. The first link captures the beginning time and values while the second captures the time and values after the user exits the application.

| Figure 6:

```
ICON1:
  INUM = 'ICON1';
  LINK FORBACK;
  IF (TEXT ^= "GOBACK") AND (TEXT ^= "MORE") THEN
  DO;
    IF (PASSNM ^= ' ') THEN
      CALL SYMPUT('TCPSEC',PASSNM1);
    CALL DISPLAY("APPLTEST.SCL",CDISPA1,DSPLAPP);
    IF (DSPLYAPP NE 'NO ' THEN
    DO;
      APPLNAME = CDISPA1;
      PASSNUM = 3;
      CURRDTM = DATETIME();
      STTIME{2} = PUT(TIMEPART(CURRDTM),11.4);
      LINK APPLSTAT;
      CALL CISPLAY(CDISPB1);
      PASSNUM = 4;
      CURRDTM = DATETIME();
      ETTIME{2} = PUT(TIMEPART(CURRDTM),11.4);
      LINK APPLSTAT;
      CALL DISPLAY("APPLTEST.SCL",CDISPA1,DSPLYAPP);
    END;
    DISPLYAPP = 'YES';
  END;
RETURN;
```

| Figure 7:

```
FORBACK:
  CALL NOTIFY(INUM,'_GET_LABEL_',TEXT);
  IF (TEXT = "GOBACK") THEN
  DO;
    PASSNO = PASSNO - 1;
    ROWNUM = SROW{PASSNO};
  END;
  ELSE IF (TEXT = "MORE") THEN
  DO;
    PASSNO = PASSNO + 1;
  END;
RETURN;
```

If the menu is ready to display but the 8 icons are not filled the system hides the unused ones as indicated in figure 8.

Figure 8

```

IF (X < 8) AND (PASSCK = 'NO ') THEN
DO;
  DO I = (X + 1) TO 8;
    ICONX = 'ICON'||I;
    CALL NOTIFY(ICONX, '_HIDE_');
  END;
END;

```

CAPTURING the APPLICATION STATISTICS

The statistics dataset was created to capture the user statistics during the day. This allowed management to track the usage amounts of a particular application. A new member was created each month so statistics could be kept over time. Statistics captured were the buffered io, direct io, page faults, elapsed cpu time, user id and time of day, and elapsed time.

Open VMS has a function called "GETJPI" that returns the DCL function statistic. The value is captured before a user goes into an application, thus creating a starting point. The value is then captured again when he/she comes out of the application. The difference is calculated and the statistics are appended to the tracking dataset. A label was created called, applstat, keeping track of the pass number, first or second, and the values of each statistic. Figure 9 shows this example.

Figure 9

```

APPLSTAT:
*-----*;
* Passnum 2 and 4 are used for the end of an application *;
*-----*;
A = PASSNUM;
BUFFIO{A} = GETJPI("BUFIO", "");
DIRRIO{A} = GETJPI("DIRIO", "");
PAGEFLT{A} = GETJPI("PAGEFLTS", "");
ELAPCPU{A} = GETJPI("CPUTIM", "");
IF PASSNUM = 2 OR PASSNUM = 4 THEN
DO;
  *-----*;
  * ELAPSED CPUTIME IS REPORTED IN HUNDREDTHS OF A SECOND *;
  * DIVISION BY 100 IS NECESSARY TO PUT TIMES AND CPU TIME IN SYNC*;
  *-----*;
  B = PASSNUM - 1;
  BUFF_IO = (BUFFIO{A} - BUFFIO{B});
  DIR_IO = (DIRRIO{A} - DIRRIO{B});
  PAGE_FLT = (PAGEFLT{A} - PAGEFLT{B});
  ELAPCPUX = ((ELAPCPU{A} / 100) - (ELAPCPU{B} / 100));
  IF PASSNUM = 2 THEN

```

```
DO;  
    STIME = STTIME{1};  
    ETIME = ETTIME{1};  
END;  
ELSE  
DO;  
    STIME = STTIME{2};  
    ETIME = ETTIME{2};  
END;  
CON_TIME = (ETIME - STIME);  
END;  
RETURN;
```

CONCLUSION

A very important case for companies is speed, accuracy, and standardization. The main menu system gives companies the power for speed, eliminates the creation and maintenance of multiple menus, and provides a standardized place for upgrades. This application was designed and coded in a short amount of time and is easy to maintain. This application is ideal for many companies needs.